Journal Pre-proof

An efficient hybrid genetic algorithm for the traveling salesman problem with release dates

Gabriel Soares, Teobaldo Bulhões, Bruno Bruck

PII: S0377-2217(24)00347-3
DOI: https://doi.org/10.1016/j.ejor.2024.05.010
Reference: EOR 19009

To appear in: *European Journal of Operational Research*

Received date : 23 February 2023
Accepted date : 5 May 2024

Please cite this article as: G. Soares, T. Bulhões and B. Bruck, An efficient hybrid genetic algorithm for the traveling salesman problem with release dates. *European Journal of Operational Research* (2024), doi: https://doi.org/10.1016/j.ejor.2024.05.010.

An efficient hybrid genetic algorithm for the traveling salesman problem with release dates

Gabriel Soares[a], Teobaldo Bulhões[b], Bruno Bruck[b]

[a] *Universidade Federal da Paraíba, Centro de Informática, Programa de Pós-graduação em Informática, Rua dos Escoteiros s/n, Mangabeira, 58055-000, João Pessoa, Brasil*
[b] *Universidade Federal da Paraíba, Centro de Informática, Departamento de Computação Científica, Rua dos Escoteiros s/n, Mangabeira, 58055-000, João Pessoa, Brasil*

Abstract

This paper deals with a generalization of the classic traveling salesman problem where each customer is associated with a release date, defined as the time at which the desired product becomes available at the depot. In this problem, a single uncapacitated vehicle is allowed to perform multiple trips in order to satisfy all demands. However, the vehicle cannot start a route unless all the products associated with the demands in the route are released. As a consequence, there might be a waiting time before starting the next route. The objective of the problem is to minimize the completion time of the service, defined as the time at which the vehicle returns to the depot after satisfying all demands. We propose a hybrid genetic algorithm that incorporates more advanced mechanisms to evaluate individuals and to ensure population diversity. We also introduce a novel dynamic programming splitting algorithm that converts a sequence of visits to customers, the so-called *giant-tour*, into the best set of routes that respects the sequence. Computational experiments performed on 522 benchmark instances show that our approach is able to find all 154 known optimal solutions from the literature. In addition, we were able to improve the best-known upper bounds for 364 instances in significantly shorter computational times when compared to the state-of-the-art.

Keywords: traveling salesman, vehicle routing, release dates, genetic algorithms

1. Introduction

The class of *vehicle routing problems* (VRPs) was proposed more than 60 years ago by Dantzig & Hamser (1959) and still remains one of the most studied in the area of

*Corresponding author
Email addresses: soaresggm@gmail.com (Gabriel Soares), tbulhoes@ci.ufpb.br (Teobaldo Bulhões), bruno.bruck@ci.ufpb.br (Bruno Bruck)

combinatorial optimization. These problems concern finding optimized vehicle routes that depart from one or more depots and that serve a set of customers.

In VRPs, it is common to assume that all goods that should be delivered to customers are available at a depot at the beginning of the distribution. In practice, though, there are scenarios in which this is not the case, and it is necessary to consider that certain products become available during the distribution process. In the *traveling salesman problem with release dates* (TSP-rd), a single uncapacitated vehicle is in charge of performing a series of deliveries to customers. However, in this problem, products may only become available after a certain time (referred to as the *release date*), the vehicle is allowed to return to the depot any number of times, and there might be a waiting time before starting the next route. In particular, we deal with a variant of the TSP-rd in which the objective is to minimize the completion time of the service, defined as the time in which the vehicle returns to the depot after satisfying all demands. This problem is $\mathcal{NP}$-hard as it generalizes the classic *traveling salesman problem* (TSP) (Archetti et al., 2015).

The concept of release date in VRPs was first used by Cattaruzza et al. (2016), which introduced the *multi-trip vehicle routing problem with time windows and release dates* (MTVRPTWR). The importance of this concept typically arises in practical contexts such as fast parcel delivery, in which products are continually being delivered at a distribution center during the day, or when the products are already in the depot and orders are dynamically placed throughout the day. Other examples are in less-than-truckload distribution systems, and in next-day delivery services, where orders arrive continuously and customers expect the deliveries to be performed either in the same day or at most in the next one. In these scenarios, it is often the case that the same vehicle performs several routes on the same day and, consequently, timing decisions have to be taken when planning the routes. Mor & Speranza (2022) refer to the class of VRPs in which the starting time of routes have to be decided as *VRPs over time*. This class includes not only the classical *VRP with time windows* (VRPTW) (Toth & Vigo, 2014) and the TSP-rd, but also some other variants that have been introduced in the last years. For instance, the *vehicle routing problem with release dates* (VRP-rd) is a generalization of the TSP-rd where a fleet of vehicles is available to perform multiple routes simultaneously (Archetti et al., 2015). Furthermore, generalizations and variants of the VRP-rd have been proposed recently (see, e.g., Shelbourne et al. 2017; Yang et al. 2021; Liu et al. 2017; Li et al. 2020; Reyes et al. 2018). The reader is referred to the survey by Mor & Speranza (2022) for further details on VRPs over time.

In the literature, the TSP-rd was introduced by Archetti et al. (2015), which studied the complexity of two variants of the problem. The first one, called TSP-rd(time), aims

at minimizing the completion time of the routes, whereas in the second one, named TSP-rd(distance), a deadline for the completion time is imposed and the objective is to minimize the total traveled distance. The authors also introduced the VRP-rd. These problems were later generalized by Reyes et al. (2018), which included a deadline for each customer, defined as the latest time a delivery can be made.

Archetti et al. (2018) studied the TSP-rd(time) by proposing a mathematical model and an *iterated local search* (ILS) that uses a destroy-and-repair method in its perturbation phase. The authors proposed two repair operators, one based on a mathematical formulation and another based on a heuristic, giving rise to two variants of the ILS, called *MathTSPrd* and *HeuTSPrd*, respectively. Furthermore, the authors introduced a set of benchmark instances derived from the classic VRPTW instances of Solomon (1987) and the TSPLIB, presented by Reinelt (1991). Their approaches were able to find optimal solutions for instances with up to 20 customers and upper bounds for instances with up to 500 customers.

Montero et al. (2021) proposed a new mathematical formulation for the TSP-rd(time) and developed a *branch-and-cut* approach to solve it. The authors were able to find optimal solutions for all instances with up to 30 customers and for some instances with up to 50 customers. In addition, they also introduced new benchmark instances derived from those of Solomon (1987) using the same procedure described in Archetti et al. (2018). The proposed approach was also applied to some TSP variants.

A dynamic version of the problem, called the *dynamic traveling salesman problem with stochastic release dates* (DTSP-srd), was proposed by Archetti et al. (2020). The decisions were represented as a Markov Decision Process and solved using a re-optimization algorithm. A similar problem, named the *dynamic dispatch waves problem* (DDWP), was studied by Klapp et al. (2018a,b), in which no information regarding customers nor their release dates are known *a priori*, and the distribution process is divided into periods of time, called *waves*. At the beginning of each wave, the algorithm must decide whether to deliver the products that are already available or wait until the next wave, when a new subset of products is released and the vehicle capacity could be better utilized. As a consequence, it is possible that some demands are not served, in which case a penalty is applied for each unit of product that is released but not delivered.

Recently, Pina-Pardo et al. (2021) introduced the *traveling salesman problem with release dates and drone resupply* (TSPRD-DR), in which the vehicle is allowed to receive newly available products en route, delivered by a drone sent from the depot. The authors proposed a *mixed integer linear programming* (MILP) model that is able to solve small

instances with 10 to 15 customers. For larger instances, they presented a solution approach that decomposes the problem into truck-routing and drone-routing decisions.

In particular, the presented literature reveals a research gap for the TSP-rd(time) both in terms of exact and heuristic approaches as there are only a couple of papers approaching the problem. In this sense, the main goal of this work is to provide an improved heuristic algorithm for the TSP-rd(time). In what follows, we describe the main contributions of the paper:

- We propose a *hybrid genetic algorithm* that incorporates more advanced mechanisms to evaluate solutions and to ensure population diversity. This algorithm, originally proposed by Vidal et al. (2012, 2014) for other VRP variants, relies on the so-called *giant-tour* chromosome (Prins, 2004), which is a sequence of visits to customers without visits to the depot. The proposed algorithm is able to find improved upper bounds for several benchmark instances;

- We develop a splitting procedure by means of a dynamic programming algorithm, which is capable of efficiently converting a giant-tour into the best set of routes associated with that tour. In addition to splitting a giant-tour, this procedure can be used as local search operator to optimize depot placement decisions. In this regard, the proposed procedure can be seen as a generalization of some operators defined by Archetti et al. (2018), as discussed in Section 3.4.3.;

- We derive a new property that is satisfied by at least one optimal solution of the problem, and we show how it can be exploited in order to speed up the splitting procedure in some cases;

- Finally, we propose a new neighborhood structure called *Divide&Swap* specifically tailored for the TSP-rd(time).

The remainder of this paper is organized as follows. In Section 2 we present the problem definition and the new property. The proposed solution approach is described in Section 3, while the results of the computational experiments are presented in Section 4. Finally, in Section 5 conclusions are drawn.

2. Problem definition and properties

We define the problem over a complete directed graph $G = (V, A)$, where $V = \{0\} \cup N$. Vertex 0 stands for the depot, while vertices in $N = \{1, 2, ..., n\}$ represent the customers. A

travel time $t(i,j)$ is given for each arc $(i,j) \in A$, and it is assumed that such values satisfy the triangle inequality. The release date of customer $i \in N$ is denoted by $r(i)$. A single uncapacitated vehicle is available to perform a sequence of routes. Each route starts at the depot, visits a sequence of distinct customers and returns to the depot. The duration and the release date (i.e., the minimum time at which all products served in that route are released) of a route $R = (0, c_1^R, c_2^R, \ldots, c_{n_R}^R, 0)$ visiting n_R distinct customers are denoted by, respectively, $t(R)$ and $r(R)$, where:

$$t(R) = t(0, c_1^R) + \left(\sum_{k=1}^{n_R - 1} t(c_k^R, c_{k+1}^R) \right) + t(c_{n_R}^R, 0) \tag{1}$$

$$r(R) = \max_{k=1}^{n_R} r(c_k^R) \tag{2}$$

A solution is a sequence of up to n routes such that each customer is visited exactly once, and the goal is to find a solution s whose *completion time* $t(s)$, i.e., the finishing time of its last route, is minimum. Note that there can be a waiting time between two consecutive routes in the solution due to the release date constraints. More formally, the completion time $t(s)$ of a solution s composed of routes $(R_1, R_2, \ldots, R_k)$ is defined as:

$$t(s) = \text{Min} \quad \sigma_k + t(R_k) \tag{3}$$

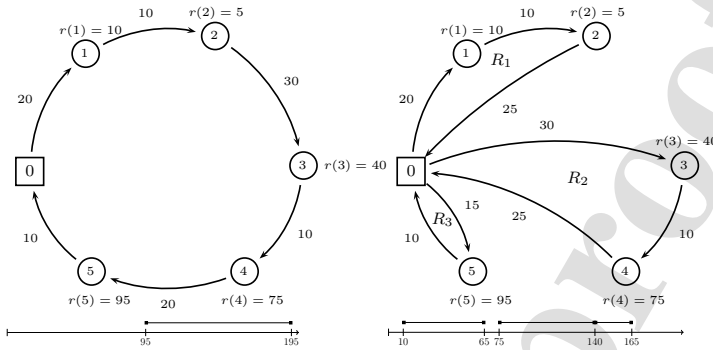
$$\text{s.t.} \quad \sigma_1 \geq r(R_1) \tag{4}$$

$$\sigma_j \geq \max\{\sigma_{j-1} + t(R_{j-1}), r(R_j)\} \qquad j = 2, \ldots, k \tag{5}$$

where σ_j, $1 \leq j \leq k$, is a variable representing the starting time of route R_j in solution s. Hereafter, let $\mathcal{R}(s)$ denote the route set of solution s.

Figure 1 shows two different solutions serving the same set of 5 customers. In the first solution (Figure 1a), a single route visits all customers and, thus, its start time cannot be smaller than the largest release date, equal to 95. In the second solution (Figure 1b), the vehicle performs three routes R_1, R_2 and R_3. The first route, R_1, starts at time 10, which is its minimum feasible start time since $r(1) = 10$, and finishes at time 65. Then, the vehicle must wait at the depot given that route R_2 cannot start before time $r(4) = 75$. On the other hand, route R_3 can be immediately started after route R_2 since the latter finishes at time 140 and $r(R_3) = 95$.

Let d_{TSP} be the value of the optimal solution of the TSP defined over graph G and $r_{\max} = \max_{i=1}^{n} r(i)$ be the largest release date. Also, for a TSP-rd(time) solution s, let $d(s)$ be the sum of the travel times of the arcs traversed in s. Finally, let s^* be an optimal



(a) Solution with a single route and cost 195 (b) Solution with three routes and cost 165

Figure 1: Two solutions for the TSP-rd(time)

TSP-rd(time) solution. Then, the following properties hold.

Property 1. *(Archetti et al., 2018) There exists an optimal solution with no waiting time after the departure of the first route.*

Property 2. *(Archetti et al., 2018) There exists an optimal solution with exactly one route starting not earlier than the largest release date.*

Property 3. *(Archetti et al., 2018) Inequality* $r_{\max} + d_{\text{TSP}} \leq 2t(s^*)$ *holds and the ratio is tight.*

Property 4. *(Archetti et al., 2018) Inequality* $d(s^*) \leq nd_{\text{TSP}}$ *holds and the ratio is tight.*

We now state a new property that will be useful in Section 3.1.

Property 5. *(New property) There exists an optimal solution whose routes are sorted in increasing order with respect to their release dates.*

Proof. Let s be a TSP-rd(time) solution in which there exists a route R_2 that immediately follows a route R_1 such that $r(R_1) \geq r(R_2)$. Define a new solution s' by replacing R_1 and R_2 with a new route $R_3 = R_1 \oplus R_2$, i.e., R_3 corresponds to the concatenation of R_1 and R_2 in this order. Given that $r(R_3) = r(R_1)$, the starting time of R_3 in s' can be the same as R_1 in s. In addition, we have that $t(R_3) \leq t(R_1) + t(R_2)$ as we suppressed a visit to the depot and the travel times satisfy the triangle inequalities. Therefore, $t(s') \leq t(s)$. □

3. Proposed hybrid genetic algorithm

The overall approach, presented in Algorithm 1, implements the population-based metaheuristic *hybrid genetic search* (HGS) developed by Vidal et al. (2012, 2014), with the exception that there is no need to handle infeasible solutions in our case. For the sake of completeness, in this section, we provide a detailed description of the method. The methodological contribution of our work lies in tailoring some building blocks of this method to the TSP-rd(time), namely the splitting (Section 3.1) and the local search (Section 3.4) procedures.

The procedure starts by generating an initial population. Then, at each iteration, the offspring is produced based on the crossover of two parent individuals from the current population. Before being inserted into the population, the offspring goes through an *education phase*, where a local search procedure is used to refine it. When the population reaches the maximum size, a *survivors selection procedure* is triggered, after which only the best individuals survive. In this procedure, an individual is evaluated not only based on its cost, but also on its contribution to population diversity. Once the algorithm goes through It_{div} iterations without improvement, a *diversification procedure* is called to insert new individuals into the population to try exploring other regions of the solution space. However, in case the algorithm reaches It_{NI} iterations without improvement, it stops and returns the best solution found. The algorithm also stops in case the total computational time exceeds the time limit T_{max}.

Algorithm 1: HGS for TSP-rd(time)

```
Initialize population;
while number of iterations without improvement < It_NI and time < T_max do
    Select parent individuals P_1 and P_2;
    Generate offspring C from P_1 and P_2 (crossover);
    Educate offspring C (local search procedure);
    Insert C into the population;
    if maximum population size reached then
        Select survivors;
    if best solution not improved for It_div iterations then
        Diversify population;
return best solution
```

Each individual in the population is represented by a giant-tour chromosome (Prins, 2004), defined as a sequence of visits to the customers without considering the depot. Note

that this representation is incomplete, given that the visits to the depot are necessary to properly evaluate the individuals. This information can be obtained by a splitting procedure that computes the best set of routes that can be generated from a giant-tour with the same sequence of visits. The idea of splitting a giant-tour by solving a shortest path problem was introduced by Beasley (1983) and first used in the context of genetic algorithms by Prins (2004). Later, Vidal (2016) developed a more efficient splitting algorithm for the same problem that runs in $O(n)$ time. For the TSP-rd(time), we developed a novel splitting algorithm, which is described in detail in the following subsection.

3.1. Split algorithm

The novel *Split* algorithm for the TSP-rd(time) presented in this section determines a set of optimal routes respecting the visiting sequence defined by the giant-tour chromosome $(c_1^P, c_2^P, \ldots, c_n^P)$ of an individual P. The algorithm works over a collection of routes $\mathcal{R} = \{R_{i,j} : i, j \in \{1, \ldots, n\}, i \leq j\}$, where $R_{i,j}$ represents the route $R_{i,j} = (0, c_i^P, c_{i+1}^P, \ldots, c_j^P, 0)$ visiting $j - i + 1$ customers. Recall that the duration $t(R)$ and the release date $r(R)$ of a route R are defined by Equations (1) and (2), respectively.

Let $\sigma(R_{i,j})$ denote the minimum feasible starting time for route $R_{i,j}$ in a solution that respects the visiting sequence defined by P. To compute such value, one must consider that all previous customers $c_1^P, \ldots, c_{i-1}^P$ have already been served (possibly by several routes) in an optimal manner. Note that $\sigma(R_{1,j}) = r(R_{1,j})$ for all routes $R_{1,j}$. Also, let $\phi(j)$, where $j \in \{1, \ldots, n\}$, denote the minimum feasible completion time of a partial solution serving the first j customers $c_1^P, \ldots, c_j^P$ of P. Thus, we have:

$$\phi(j) = \min_{i=1}^{j} \Big\{ \sigma(R_{i,j}) + t(R_{i,j}) \Big\} \tag{6}$$

$$\sigma(R_{i,j}) = \begin{cases} \max\{\phi(i-1), r(R_{i,j})\} & \text{if } i > 1 \\ r(R_{i,j}) & \text{if } i = 1 \end{cases} \tag{7}$$

Therefore, the minimum completion time of a solution respecting the visiting sequence defined by P is given by $\phi(n)$. Algorithm 2 presents a dynamic programming approach to compute, in $O(n^2)$ time, all values $\phi(\cdot)$. In its j-th iteration (for loop in lines 3-16), the algorithm determines the value $\phi(j)$ based on the values $\phi(i)$, $i < j$, computed in its previous iterations. For doing so, the minimum completion time ϕ' of a partial solution visiting the first j customers of P that contains R_{ij} as its last route is computed for each $i \leq j$ (for loop in lines 6-14). Remark that the minimum feasible starting time σ' of

route R_{ij} in such a partial solution (lines 7-10) corresponds to the value $\sigma(R_{i,j})$ defined in Equation (7). The value ϕ' (line 11) is then compared against the value ϕ'_j (the current best estimation on $\phi(j)$), and the latter is updated if it is greater than the former (lines 12-14). The algorithm also keeps track, for each $j \in \{1, \ldots, n\}$, of the value $i = \theta'_j$ associated with the current best estimation on $\phi(j)$ (line 14). Clearly, ϕ'_j equals $\phi(j)$ at the end of the iteration (lines 15-16).

Algorithm 2: *Split*

```
1  φ(j) ← ∞, j ∈ {1, ..., n}
2  θ(j) ← null, j ∈ {1, ..., n}
3  for j ← 1 to n do
4      φ'_j ← ∞
5      θ'_j ← null
6      for i ← 1 to j do
7          if i > 1 then
8              σ' ← max{φ(i − 1), r(R_{i,j})}
9          else
10             σ' ← r(R_{i,j})
11         φ' = σ' + t(R_{i,j})
12         if φ' < φ'_j then
13             φ'_j ← φ'
14             θ'_j ← i
15     φ(j) ← φ'_j
16     θ(j) ← θ'_j
17 return φ(·), θ(·)
```

The correctness of Algorithm 2 does not depend on the assumption that the triangle inequality is satisfied. However, one can take advantage of it to speed up the algorithm as follows. For each $1 \leq j \leq n$, let β_j denote the minimum position $1 \leq k \leq j$ such that the release date of customer c^P_k is maximum. In light of Property 5, we know that there exists an optimal solution that does not contain any route $R = R_{i,j}$ such that $i > \beta_j$, otherwise R would be preceded by another route whose release date is not smaller than that of R. Therefore, we can replace j by β_j in line 6 of Algorithm 2. Although this trick does not change the worst-case time complexity of the algorithm, it allows for a linear time complexity in some cases — e.g., when a customer with a maximum release date is placed in the first position of the giant-tour, implying that $\beta_j = 1$ for all values of j. Of course, the linear time complexity is guaranteed in these cases because (i) the β_j values can be

computed in linear time; (ii) the duration $t(R_{i,j})$ can be retrieved in constant time given that accumulated travel times can be precomputed in linear time; and (iii) all the release dates that will be needed in the modified version of the algorithm satisfy $r(R_{i,j}) = \beta_j$.

Figure 2 illustrates the execution of the *Split* algorithm over a 4-customer instance. The $\theta(\cdot)$ values computed by Algorithm 2 feed the backtracking approach described in Algorithm 3 that retrieves the actual set of routes $\mathcal{R}'$ of the solution. In the example shown in Figure 2b, this algorithm begins by examining $\theta(4) = 3$, indicating that the route $R_{3,4}$ is included in the solution. Next, it considers the element representing the route that ends immediately before $\theta(2) = 2$, signifying that the route $R_{2,2}$ is also part of the solution. Finally, the algorithm examines the element $\theta(1) = 1$, which corresponds to the inclusion of the route $R_{1,1}$ in the solution. Throughout the paper, we will denote the solution (set of routes) computed by *Split* for an input individual P as $s(P)$.

Note that the proposed *Split* algorithm is not comparable to the one presented in Vidal et al. (2012) given that the underlying problem that is solved by the procedure is significantly different in the context of the TSP-rd(time), as it must take into account the release dates and completion times. On the other hand, in Vidal et al. (2012), the splitting problem corresponds to finding the set of routes with minimum total cost (sum of route costs), which can be solved as a shortest-path problem on an auxiliary graph.

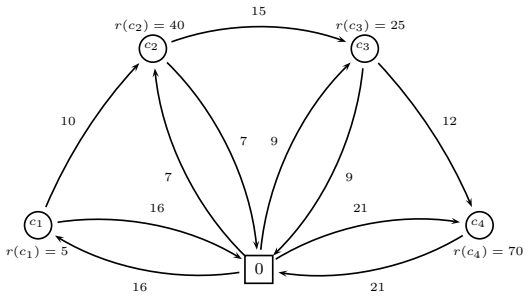
Algorithm 3: Retrieving the routes

```
1 R ← ∅
2 j ← n
3 while j ≥ 1 do
4 |   R ← R ∪ {R_{θ(j),j}}
5 |   j = θ(j) − 1
6 return R
```

3.2. Individuals evaluation

Genetic algorithms employ a *fitness measure* to evaluate individuals as the population evolves. In principle, one could use the objective function of the problem itself as a fitness measure — e.g., in the TSP-rd(time), the fittest individuals would be those whose associated solutions have the earliest completion time. However, as highlighted by Vidal et al. (2012), this might not lead to an efficient genetic approach as the contribution of an individual to population diversity is not taken into account, rendering the algorithm vulnerable to premature convergence.

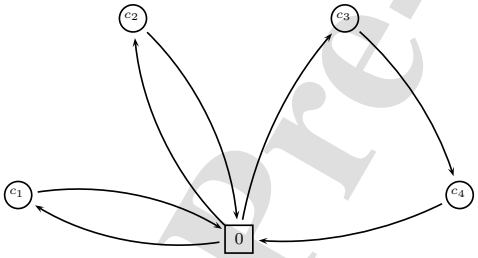
Figure 2: An execution of *Split* for a giant-tour chromosome (c_1, c_2, c_3, c_4)



(a) Travel times (only those needed by *Split*) and release dates

$i \backslash j$	1	2	3	4
1	**5+32**	40+33	40+50	70+74
2		**40+14**	**40+31**	70+55
3			54+18	**70+42**
4				71+42
$\phi(j)$	37	54	71	112
$\theta(j)$	1	2	2	3

(b) $\sigma(\cdot) + t(\cdot)$ (upper table), $\phi(\cdot)$ and $\theta(\cdot)$ values



(c) Solution with cost 112

To circumvent this, Vidal et al. (2012) introduced a *biased fitness measure* that evaluates an individual both with respect to its objective function value and its contribution to population diversity. In this work, we adopted the distance function $\delta(s_1, s_2)$ defined by Bulhões et al. (2018) and shown in Equation (8), where $A(s)$ denotes the set of arcs traversed by the routes of solution s. Remark that the minimum distance value 0 is attained when s_1 and s_2 have the same set of arcs, while the maximum value 1 is attained when the corresponding sets of arcs are disjoint.

$$\delta(s_1, s_2) = 1 - \frac{|A(s_1) \cap A(s_2)|}{|A(s_1) \cup A(s_2)|} \tag{8}$$

Given a solution s, its contribution to population diversity, denoted as $\Delta(s)$, is defined as in Equation (9):

$$\Delta(s) = \frac{\sum_{s' \in \mathcal{N}(s)} \delta(s, s')}{n_{close}}, \tag{9}$$

where $\mathcal{N}(s)$ is the set of n_{close} solutions in the population that are closest to s according to

the distance function (8). In other words, $\Delta(s)$ is the average distance between s and the solutions in $\mathcal{N}(s)$.

Now let $fit(P) \in \{1, \ldots, nbIndiv\}$ be the rank of an individual P in a population of $nbIndiv$ individuals with respect to the completion time of its associated solution. That is, the larger the completion time of $s(P)$, the larger the value $fit(P)$. Similarly, define $dc(P) \in \{1, \ldots, nbIndiv\}$ as the rank of P in the population with respect to its contribution to diversity, i.e., the larger the value $\Delta(s(P))$, the smaller the rank $dc(P)$. The biased fitness measure introduced by Vidal et al. (2012) is then defined as in Equation (10):

$$BF(P) = fit(P) + \left(1 - \frac{nbElit}{nbIndiv}\right) dc(P), \tag{10}$$

where $nbElit$ controls the elitism of the measure. For instance, if $nbElit = 0$, then both $fit(P)$ and $dc(P)$ affect equally the final fitness value $BF(P)$. On the other hand, if $nbElit = nbIndiv$, the contribution of P to the population diversity will have no impact on the measure. The value $BF(P)$ is used by the algorithm in its fundamental procedures such as parent and survival selections, as will be shown in the next subsections. As its definition depends on the current population, it must be recomputed whenever necessary during the course of the optimization.

3.3. Parent selection and crossover

In the reproduction phase, two parents P_1 and P_2 are selected by means of *binary tournaments*. More precisely, two distinct individuals are randomly selected in the population, and the one with the best value according to the biased fitness measure is selected as the first parent P_1. The same procedure is repeated in order to select the second parent P_2, ensuring that P_2 is different from P_1. Finally, the classical *order crossover* (OX) (Davis, 1991) is applied to P_1 and P_2 to generate the offspring C.

3.4. Education

During the education phase, we perform a local search on a given solution s with the route set $\mathcal{R}(s) = \{R_1, R_2, \ldots R_{|\mathcal{R}(s)|}\}$. As shown in Algorithm 4, this procedure is divided into three steps, called *intra-route*, *inter-route* and *depot assignment* improvements. The procedure is repeated until no improvement is found. In the remainder of this subsection, each step of the education phase is described in detail.

3.4.1. Intra-route improvement

This procedure is composed of a set of intra-route moves that operate on each route $R_i \in \mathcal{R}(s)$. The objective is to improve the completion time of the solution without changing

Algorithm 4: Education

```
function Education(s):
    repeat
        IntraRouteImprovement(s);
        InterRouteImprovement(s);
        DepotAssignmentImprovement(s);
    until no improvement is found;
```

the set of customers served by each route. In the following, we present the list of moves considered in the procedure.

- **Exchange(1, 1)**: Swap of two customers from the same route.
- **Exchange(1, 2)**: Swaps a customer with a set of two adjacent customers on the route.
- **Reinsertion**: One customer is removed and reinserted into a different position of the route.
- **Or-opt2**: A set of two adjacent customers is removed and inserted into a different position of the route.
- **2-opt**: Two nonadjacent arcs are removed, disconnecting part of the route. Then, two other arcs are inserted in such a way that the disconnected part is reconnected in the inverse order.

The complete procedure is implemented as a *randomized variable neighborhood descent* (RVND) (Subramanian et al., 2010; Subramanian, 2012), an extension of the well-known *variable neighborhood descent* (VND) procedure of Mladenović & Hansen (1997) in which the neighborhoods are explored in a random order. The procedure is called for each route and the neighborhoods are explored using the *best improvement* strategy. Considering that in this step we are optimizing the duration of a route and the moves involve the exchange of a constant number of arcs, the evaluation of a single move can be performed in constant time.

3.4.2. Inter-route improvement

This procedure is based on the following inter-route neighborhoods. As for the previous procedure, these moves are part of the modified RVND described in Algorithm 5. However,

here is used the *first improvement* strategy. The pairs of routes are considered in a random order during the exploration of a neighborhood.

1. **Swap(1,1)**: Permutation between two customers from different routes.
2. **Shift(1)**: Remove a customer from a route and insert it into a different route.
3. **Divide&Swap**: Divide a route into two new routes and swap their visiting order in the solution. This move is explained in detail later in this section.

Algorithm 5: Local search from a solution s over neighborhood set N and objective function $f(.)$.

```
N' ← shuffle(N)
k ← 1
while k ≤ |N| do
    m ← 0
    while true do
        Find a neighbor s' in the k-th neighborhood of N' using the first
         improvement strategy
        if s' is found then
            s ← s'
            m ← m + 1
        else
            break
    if m > 0 then
        N' ← shuffle(N)
        k ← 1
    else
        k ← k + 1
return s
```

The evaluation of solutions obtained by inter-route moves is calculated based only on the completion time of the last route involved in the move. Note that, this does not ensure that the completion time of the solution is improved. However, decreasing the completion time of internal routes may lead to better solutions later. For instance, suppose we have a solution with four routes and an inter-route move was applied to routes R_1 and R_3. In case

the yielded solution has a better completion time for R_3, it is accepted as a better neighbor even if the completion time of R_4 remains unaltered. This opens the possibility that another move might be able to decrease the release date of route R_4 and improve the solution. The complexity of this evaluation is $O(|\mathcal{R}(s)|)$ given that, to obtain the completion time of the last route in the move, we need to perform a reevaluation starting from the first route.

Divide&Swap. The so-called *Divide&Swap* move initially divides a route $R = (0, c^R_1, c^R_2, \ldots, c^R_{n_R}, 0) \in \mathcal{R}(s)$ into routes $R' = (0, c^R_1, c^R_2, \ldots, c^R_{d-1}, 0)$ and $R'' = (0, c^R_d, c^R_{d+1}, \ldots, c^R_{n_R}, 0)$, for some $d \in \{2, 3, \ldots, n_R\}$. Then, it swaps routes R' and R'' in the solution, thus, changing the order in which these routes are performed.

To understand the motivation behind this move, consider the solution depicted in Figure 3a, with a completion time equal to 100, and suppose we decide to divide route R_2. Note that $r(R_2) = 50$ because of customer 3, which has the largest release date in the route. In case we divide R_2 just before customer 3, we obtain the solution shown in Figure 3b, which is clearly better than the original one without needing to swap the newly created routes. However, this is equivalent to adding an additional visit to the depot before visiting customer 3, a scenario that is already considered by the move described in Section 3.4.3.

Alternatively, we may first divide route R_2 right after customer 3, resulting in the solution of Figure 3c. Even though this solution has a larger completion time, the release date of route R_3 is smaller than that of the new route R_2. This allows us to swap routes R_2 and R_3, yielding the solution depicted in Figure 3d, which has a completion time of 90.

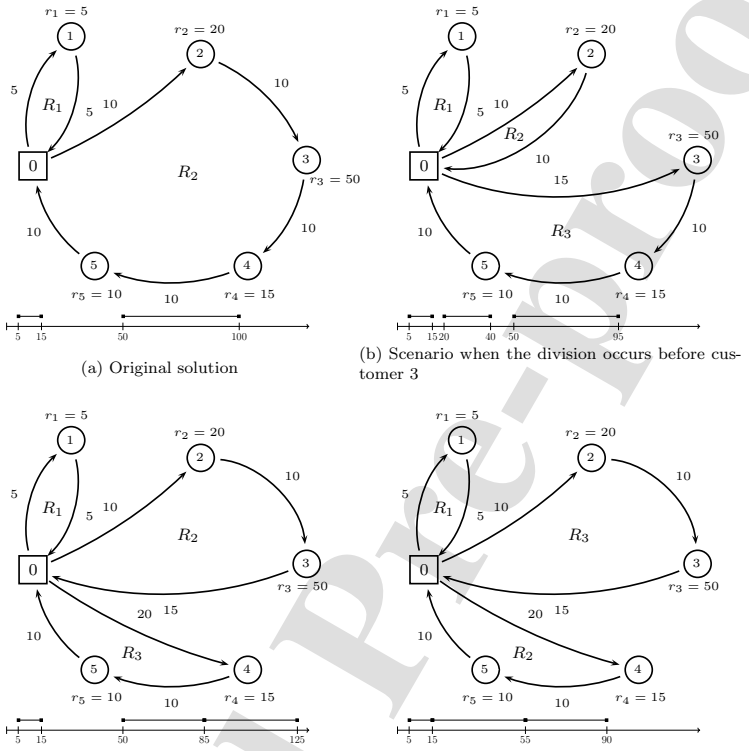
3.4.3. Depot assignment improvement

The depot assignment improvement phase tries to find better solutions by delaying or expediting each visit to the depot without altering the sequence of visits to the customers. Archetti et al. (2018) proposed three moves focused on changing when the vehicle returns to the depot. These moves are depot insertion, shift and removal. In our algorithm, we do not use these moves, instead, we use the *Split* procedure presented in Section 3.1. This procedure runs in $O(n^2)$ time and is able to find the best assignments for the depot, thus, yielding the optimal set of routes associated with the given sequence of visits. As a consequence, the procedure generalizes the moves proposed by Archetti et al. (2018) that deal with depot assignment.

3.5. Population management

The population management mechanism is composed of a set of procedures that, together with the algorithms presented in the previous sections, seek to identify and maintain

Figure 3: Example of the Divide&Swap move



(a) Original solution

(b) Scenario when the division occurs before customer 3

(c) Solution obtained by dividing route R_2 after customer 3

(d) Solution after swapping routes R_2 and R_3

the characteristics of the best solutions. The minimum population size is defined by μ. Individuals are generated by means of reproduction and added to the population until it reaches its maximum size, defined as $\mu + \lambda$, where λ denotes the generation size. Once it reaches its maximum size, the population goes through a survivors selection phase, in which λ individuals are removed.

The survivors selection phase is implemented as in Algorithm 6 and tries to preserve the best individuals, while maintaining a certain level of diversity in the population. To this end, the algorithm is driven by the biased fitness measure and discards the worse λ individuals. Note that the algorithm gives priority to discarding individuals that are considered *clones*, which are those whose distance, given by Equation (8), is zero. As pointed out by Vidal et al. (2012), the best *nbElit* individuals in terms of completion time that are not clones are guaranteed to survive.

Algorithm 6: Survivors Selection

```
1 procedure SelectSurvivors():
2     Evaluates the biased fitness for each individual;
3     while population size > μ do
4         X ← all individuals that have clones;
5         if X ≠ ∅ then
6             remove from the population the individual in X with the worst biased
                fitness value
7         else
8             remove from the population the individual with the worst biased fitness
                value
```

The initialization of the population is performed by randomly generating 2μ individuals and subsequently applying the *Split* algorithm to obtain a set of routes. In case $2\mu \geq \mu + \lambda$, the survivors selection procedure is called to reduce the population size to the minimum. Finally, in case the algorithm goes through It_{div} iterations without improvement, the diversification procedure is triggered. This procedure initially keeps only the best $\mu/3$ individuals according to the *biased fitness* measure. Then, the initialization procedure is used to generate 2μ new random individuals. Such new individuals promote diversity due to the random nature of the initialization procedure. Nonetheless, it should be noted that the diversification procedure preserves good solutions as a considerable portion of the best individuals are kept in the population.

4. Computational experiments

Our algorithm was implemented in C++ (available at https://github.com/gosoares/TSPrd) and the experiments were performed in a PC with an Intel Xeon E5-2650v4 2.2 GHz with 128 GB of RAM under the operational system Ubuntu 16.04. To attest to the efficiency of our approach, we compare the results with those from Archetti et al. (2018), and compensate for the difference in CPUs by adjusting the computational times with information acquired from PassMark (Available at https://www.cpubenchmark.net/singleThread.html. Accessed on Aug 16, 2021). According to the software, the CPU used by Archetti et al. (2018) has a *single-core* performance 24.9% better than the CPU used in our experiments. Therefore, the timing measurements reported in this paper for our algorithm were adjusted accordingly.

In this work, we used the benchmark instances proposed by Archetti et al. (2018), which comprises a total of 450 instances derived from three classical sets of TSP instances: the

Solomon instances (Solomon, 1987) and both symmetric and asymmetric TSPLIB instances (Reinelt, 1991). We also considered the 72 instances derived from the Solomon benchmark set by Montero et al. (2021), generated using the same procedure as in Archetti et al. (2018). Therefore, in total, we considered 522 benchmark instances. In this section, we refer to these three groups of instances as Solomon, TSPLIB (symmetric) and aTSPLIB (asymmetric). There are six classes of instances, classified according to a parameter $\beta \in \{0.5, 1, 1.5, 2, 2.5, 3\}$, specifying the width of the interval in which release dates are generated. In general, lower values of β lead to instances with shorter release dates and less variance. For some of these instances, the distance matrix is given, while for others, only Cartesian coordinates are provided. For the latter, as in Archetti et al. (2018), we evaluated the Euclidean distance for each pair of points and rounded the values to the nearest integer. Then, to ensure that the triangle inequality holds, we applied the Floyd-Warshall (Floyd, 1962) algorithm.

4.1. Parameter tuning

The algorithm proposed in Section 3 relies on a set of correlated parameters, which must be properly calibrated to maximize efficiency. To this end, we initially set the value of each parameter equal to the ones in Vidal et al. (2012). A summary of the parameters, their description and initial values is presented in Table 1. Note that, as in Vidal et al. (2012), the value of $itDiv$ is calculated as $0.4 \times itNi$. The value of $T_{\max}$ is set to 10 minutes so as to be consistent with the time limit imposed for the algorithm proposed by Archetti et al. (2018).

To calibrate the parameters we selected all 96 instances with number of customers ranging from 150 to 300. From preliminary experiments, we noticed that for all these instances our algorithm either found equal or better solutions than *HeuTSPrd*. Therefore, we first ran each instance 10 times with the initial parameter values and used the best solution value obtained as the reference value for the comparisons.

To simplify, we initially only varied the parameters μ and λ, and fixed the others to their initial values. We then ran *HGS* 10 times for all 96 instances and considered several pairs of (μ, λ). Table 2 presents a summary of the results, including an ID for the scenario, the respective values for each parameter, the average gap with respect to the reference value and the average CPU time. These results are depicted in Figure 4, where it becomes clear that some scenarios are dominated by others. Among the non-dominated ones, scenario 5 offers the best compromise. It is worth pointing out that the initial values of these parameters are represented by scenario 8, which is also dominated.

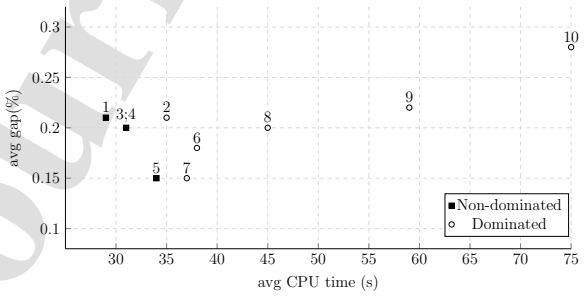
Table 1: Summary of the original HGS parameters for the TSP-rd(time).

Parameter	Description	Initial value
μ	Population size	25
λ	Number of offspring in each generation	100
el	Proportion of elite individuals, such that $nbElit = el \times \mu$	0.4
nc	Proportion of the nearest individuals that are considered in the distance evaluation, such that $n_{close} = nc \times \mu$	0.2
$itNi$	Number of iterations without improvement in the best solution in order to stop the algorithm	2000
$itDiv$	Number of iterations without improvement in the best solution in order to apply the diversification procedure	800
T_{max}	Time limit for the algorithm	10 min

Table 2: Calibration results for parameters μ and λ

ID	μ	λ	gap (%)	time (s)
1	13	50	0.21	29
2	13	100	0.21	35
3	15	40	0.20	31
4	15	60	0.20	31
5	20	40	0.15	34
6	20	80	0.18	38
7	25	50	0.15	37
8	25	100	0.20	45
9	38	150	0.22	59
10	50	200	0.28	75

Figure 4: Calibration results for parameters μ and λ

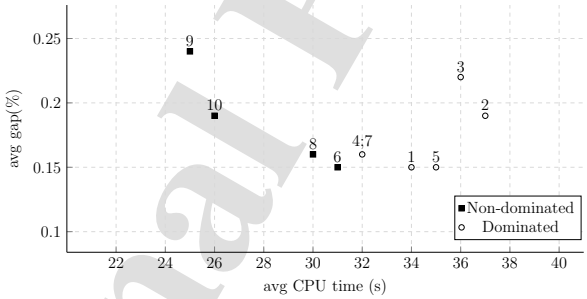


In a second step, we fixed the values of μ and λ as in scenario 5, and experimented with different values of the parameters el and nc. Table 3 shows the average results of this calibration. These results are plotted in Figure 5, in which we can observe that Scenario 6 seems to yield the best results and is therefore selected.

Table 3: Calibration results for parameters el and nc

ID	el	nc	gap (%)	time (s)
1	0.4	0.2	0.15	34
2	0.2	0.1	0.19	37
3	0.2	0.2	0.22	36
4	0.4	0.1	0.16	32
5	0.4	0.3	0.15	35
6	0.5	0.3	0.15	31
7	0.6	0.3	0.16	32
8	0.6	0.4	0.16	30
9	0.8	0.2	0.24	25
10	0.8	0.4	0.19	26

Figure 5: Calibration results for parameters el and nc



Finally, a similar experiment has been performed to calibrate the parameter $itNi$. Figure 6 presents the results of this calibration. For this parameter, we experimented with values from 500 to 20000, and the corresponding values, divided by 1000, are shown above each scenario. We chose $itNi = 10000$ given that it yielded solutions with gaps, on average, below 1% and with good computational times. Table 4 presents a summary of the parameter values chosen after the calibration.

4.2. Analysing the impact of the neighborhoods

In order to assess the impact of the neighborhood structures and choose the best setting,

Figure 6: Calibration results for $itNi$.

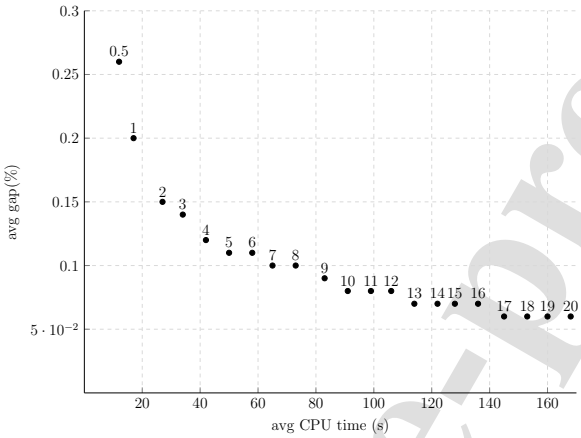


Table 4: Summary of the chosen parameter values

Parameter	value
μ	20
λ	40
el	0.5
nc	0.3
$itNi$	10000
$itDiv$	4000
T_{max}	10 min

we performed a round of experiments with different combinations. For this experiment, we only considered instances from sets Solomon, TSPLIB and aTSPLIB, which contain the largest ones. Table 5 presents the average results of this analysis considering 10 runs for each instance and setting. Column *gap* specifies the average gap to the best-known solution in the literature, whereas column *# better* presents the number of instances in which *HGS* found a better solution. Column *TT* provides the average total CPU time (in seconds), while column *TB* specifies the average time to find the best solution. Note that, in these experiments, we considered a sixth intra-route neighborhood, called $exchange(2,2)$, which swaps two disjoint sets of two adjacent customers.

According to the results depicted in Table 5, there is an almost constant improvement when adding each of the neighborhoods. The largest improvements unarguably are obtained when the first inter-route neighborhood is added. However, when $swap(1,1)$ is included

Table 5: Results obtained for nine different settings of neighborhood structures

Setting	Combination	gap	# better	TT	TB
Π_1	2-opt	-0.07	56	59.27	46.60
Π_2	Π_1 + exchange(1,1)	-0.12	58	91.50	71.38
Π_3	Π_2 + exchange(1,2)	-0.13	58	133.41	105.36
Π_4	Π_3 + exchange(2,2)	-0.11	54	143.03	112.38
Π_5	Π_4 + reinsertion	-0.29	66	149.77	117.39
Π_6	Π_5 + or-opt-2	-0.34	63	154.73	121.31
Π_7	Π_6 + shift(1)	-3.92	321	146.44	94.17
Π_8	Π_7 + swap(1,1)	-3.93	321	158.14	99.71
Π_9	Π_8 + divide&swap	-3.95	321	160.48	100.71
Π_{10}	Π_9 - exchange(2,2)	-3.95	321	154.30	96.40

and later *divide&swap*, the improvements are very small. To evaluate the relevance of keeping these two neighborhoods in the algorithm, we performed Wilcoxon signed-rank tests, which showed that there is a statistical difference between (Π_7, Π_8) and (Π_8, Π_9), with p-values equal to 0.0041 and 0.00001, respectively. Considering the results of Π_4, which are statistically worse than Π_3 (p-value equal to 0.3873), we tested another setting, $\Pi_{10} = \Pi_9 - exchange(2,2)$. The experiments with setting Π_{10} attest that the neighborhood $exchange(2,2)$ is indeed not required and by removing it we can improve the computational times. Therefore, we chose to use setting Π_{10}.

4.3. Comparison with the literature

In this section, we compare our results with those from the best-known heuristic from the literature, called *HeuTSPrd* (Archetti et al., 2018). Note that, although Archetti et al. (2018) proposes a second algorithm, called MathTSPrd, we do not include it in the comparisons because, according to the authors, this algorithm is outperformed by HeuTSPrd, only finding a better result for a single instance.

We ran our algorithm 10 times for each instance. However, it is important to mention that, in Archetti et al. (2018), the authors only ran each instance once. Because of this difference, we do not have information on average results for *HeuTSPrd*. We also analyze our results over the 72 benchmark instances proposed by Montero et al. (2021), which were not considered by Archetti et al. (2018). Detailed results for each instance can be found in the Appendix.

Table 6 presents a comparison of our results with the ones obtained by Archetti et al. (2018) for the 77 instances with known optimal solutions (Montero et al., 2021). These results are aggregated by each pair (n, β), and the number of instances for each combination

is shown in column *# inst*. Columns named *# opt* and *gap* provide, respectively, the number of instances for which the upper bound found is the global optimum, and the average gap to the optimal solution. The average time (in seconds) required to find the upper bound is shown in columns *TB*, whereas the average total time is presented in columns named *TT*. It is important to point out that, in Archetti et al. (2018), the authors only provide the value of TB, and exclusively for instances with $n = \{20, 50\}$. Furthermore, note that for *HGS* we not only present average results for all 10 runs (column *average*) but also the results considering only the best solutions (column *best run*). According to these results, in the best runs our approach was always able to find the optimal solution. Even when considering all runs, for 76 out of 77 instances, *HGS* was able to find the optimal solution in every single run. In general, for these instances, our algorithm clearly outmatches *HeuTSPrd* both in terms of solution quality and computational time. For the sake of completeness, Table 7 shows the aggregated results for the remaining instances with known optimal solutions for which Archetti et al. (2018) did not report computational results. Note that *HGS* was able to find the optimal solution in all 77 instances in the best run and 76 on the average results.

Table 6: Comparison of aggregated results for instances with known optimal solutions

			HeuTSPrd			*HGS*					
						best run		average			
n	β	# inst	# opt	gap	TB	# opt	gap	# opt	gap	TT	TB
10	0.5	4	3	0.12	-	4	0.00	4	0.00	0.36	<0.01
	1	4	4	0.00	-	4	0.00	4	0.00	0.40	<0.01
	1.5	4	4	0.00	-	4	0.00	4	0.00	0.42	<0.01
	2	4	4	0.00	-	4	0.00	4	0.00	0.43	<0.01
	2.5	4	4	0.00	-	4	0.00	4	0.00	0.45	<0.01
	3	4	4	0.00	-	4	0.00	4	0.00	0.44	<0.01
15	0.5	4	3	0.17	-	4	0.00	4	0.00	0.44	<0.01
	1	4	4	0.00	-	4	0.00	4	0.00	0.49	<0.01
	1.5	4	4	0.00	-	4	0.00	4	0.00	0.53	<0.01
	2	4	4	0.00	-	4	0.00	3	0.01	0.58	0.02
	2.5	4	4	0.00	-	4	0.00	4	0.00	0.59	<0.01
	3	4	4	0.00	-	4	0.00	4	0.00	0.60	<0.01
20	0.5	4	3	1.03	90	4	0.00	4	0.00	0.56	<0.01
	1	4	2	0.55	153	4	0.00	4	0.00	0.65	0.05
	1.5	4	3	0.10	183	4	0.00	4	0.00	0.67	0.02
	2	4	3	0.13	182	4	0.00	4	0.00	0.74	0.03
	2.5	4	4	0.00	193	4	0.00	4	0.00	0.76	<0.01
	3	4	3	0.03	186	4	0.00	4	0.00	0.80	0.02
50	0.5	3	2	0.30	71	3	0.00	3	0.00	1.78	0.04
	1	2	1	0.24	24	2	0.00	2	0.00	1.98	0.08
Avg.				0.13	-		0.00		0.00	0.68	0.01
Total		77	67			77		76			

Table 7: Results for remaining instances with known optimal solutions

			HGS					
			best run		average			
n	β	# inst	# opt	gap	# opt	gap	TT	TB
25	0.5	4	4	0.00	4	0.00	0.69	<0.01
	1	4	4	0.00	4	0.00	0.81	0.05
	1.5	4	4	0.00	4	0.00	0.87	0.02
	2	4	4	0.00	4	0.00	0.95	<0.01
	2.5	4	4	0.00	4	0.00	1.05	0.02
	3	4	4	0.00	4	0.00	1.09	<0.01
30	0.5	4	4	0.00	4	0.00	0.85	<0.01
	1	4	4	0.00	4	0.00	0.97	0.04
	1.5	4	4	0.00	4	0.00	1.07	0.02
	2	4	4	0.00	4	0.00	1.17	0.01
	2.5	4	4	0.00	4	0.00	1.30	<0.01
	3	4	4	0.00	4	0.00	1.38	<0.01
35	0.5	4	4	0.00	4	0.00	1.08	0.01
	1	3	3	0.00	3	0.00	1.19	0.02
	1.5	3	3	0.00	3	0.00	1.31	0.04
	2	2	2	0.00	2	0.00	1.38	0.04
	2.5	2	2	0.00	2	0.00	1.55	<0.01
40	0.5	4	4	0.00	4	0.00	1.35	0.04
	1	3	3	0.00	3	0.00	1.43	0.04
	1.5	1	1	0.00	1	0.00	1.49	0.04
	2	1	1	0.00	1	0.00	1.63	0.02
45	0.5	3	3	0.00	3	0.00	1.47	0.03
	1	2	2	0.00	2	0.00	1.85	0.15
	1.5	1	1	0.00	0	0.01	2.28	0.58
Avg.				0.00		0.00	1.26	0.05
Total		77	77		76			

A comparison of the results for the instances whose optimal solution is unknown is presented in Table 8. In this case, columns named *gap* specify the gap relative to the solution of *HeuTSPrd*, and negative values refer to improvements found by our approach. Columns named *# better* refer to the number of instances for which *HGS* found a better solution than *HeuTSPrd*. Note that we omit this column for *HeuTSPrd* because their results are never better for these instances. According to these results, for 321 out of 325 instances, our approach found better solutions, even when considering the average of all 10 runs. For the remaining four instances, we found the same solutions as *HeuTSPrd*. Regarding computational times, unfortunately, for *HeuTSPrd* we only have information on the values of TB for the Solomon instances with $n = \{50, 100\}$, in which case our approach is considerably faster.

Table 9 presents aggregated results for the remaining instances, for which we only have

the upper bounds provided by Montero et al. (2021). These bounds were obtained when their approach reached the imposed time limit of one hour. As such, in this table, columns named *gap* specify the gap relative to the upper bounds found by the authors. Results show that our algorithm was able to find better solutions for 40 out of 43 instances, and found the same solution for the remaining 3 instances, even when considering all runs. Furthermore, our approach was able to find solutions that are, on average, about 3% better and in much shorter computational times.

5. Concluding remarks

In this work, we studied a variant of the traveling salesman problem with release dates in which the objective is to minimize the completion time of the routes. This problem commonly arises in last-mile applications such as fast parcel delivery, where the delivery time is a critical aspect for success. Even though the problem has direct practical applications, it has received limited attention in the literature.

We propose a hybrid genetic algorithm to efficiently solve the problem. This algorithm incorporates a series of advanced procedures to evaluate solutions and manage the population. Furthermore, the algorithm uses an auxiliary procedure called *split*, which is able to find the best set of routes from a sequence of visits, referred to as *giant-tours*. This algorithm is also used during a local search phase. Computational experiments on 522 benchmark instances from the literature attest to the efficiency of the proposed approach, which managed to find better solutions for the majority of the scenarios in shorter computational times.

A possible future work direction includes studying the dynamic version of the problem, in which not all the release dates are known at the start of the distribution process. Furthermore, in practice, release and are subjected to uncertainties, which, if not properly managed, can render solutions infeasible. The most common approaches in the literature to deal with uncertainties are Stochastic Programming and Robust Optimization. While the aforementioned variants present a greater challenge, they are also much closer to practical contexts.

Acknowledgments

This research was partially supported by: (i) CNPq, grants 314088/2021-0 and 406245/2021-5; (ii) CAPES, grant 88887.671346/2022-00; (iii) Paraíba State Research Foundation (FAPESQ), grants 261/2020 and 041/2023; (v) Universidade Federal da Paraíba through the Public Call n. 03/2020 “Produtividade em Pesquisa

Table 8: Comparison of aggregated results for instances with unknown optimal solutions

				HeuTSPrd	*HGS*					
					best run		average			
		β	# inst	TB	# better	gap	# better	gap	TT	TB
Solomon	50	0.5	1	59	0	-4.08	0	-4.08	2.27	0.27
		1	2	233	2	-3.87	2	-3.87	2.39	0.19
		1.5	4	142	4	-4.65	4	-4.59	2.69	0.32
		2	4	294	4	-6.09	4	-6.06	3.32	0.63
		2.5	4	107	4	-4.80	4	-4.79	3.36	0.40
		3	4	317	3	-5.05	3	-5.05	3.26	0.04
	100	0.5	4	263	4	-1.54	4	-1.46	9.87	2.43
		1	4	307	4	-3.02	4	-2.77	13.02	4.26
		1.5	4	366	4	-3.66	4	-3.50	16.31	5.66
		2	4	236	4	-2.91	4	-2.75	17.91	5.30
		2.5	4	222	4	-2.54	4	-2.50	17.20	2.94
		3	4	272	4	-1.59	4	-1.57	19.69	3.93
TSPLIB	50 ~ 100	0.5	12	-	11	-0.72	11	-0.68	8.06	2.13
		1	12	-	11	-1.58	11	-1.51	10.13	3.13
		1.5	12	-	12	-2.23	12	-2.13	13.10	4.59
		2	12	-	12	-1.82	12	-1.77	14.71	4.79
		2.5	12	-	12	-1.17	12	-1.13	16.60	5.52
		3	12	-	12	-0.70	12	-0.70	15.94	3.91
	101 ~ 150	0.5	11	-	11	-1.80	11	-1.70	21.09	6.55
		1	11	-	11	-2.95	11	-2.71	30.31	12.65
		1.5	11	-	11	-2.94	11	-2.77	41.76	19.95
		2	11	-	11	-2.35	11	-2.22	45.14	19.57
		2.5	11	-	11	-1.72	11	-1.66	43.67	15.25
		3	11	-	11	-1.17	11	-1.11	42.99	12.57
	151 ~ 250	0.5	9	-	9	-1.93	9	-1.71	76.26	35.27
		1	9	-	9	-3.70	9	-3.44	99.27	48.89
		1.5	9	-	9	-3.24	9	-3.02	114.50	53.71
		2	9	-	9	-3.00	9	-2.82	126.77	56.14
		2.5	9	-	9	-2.19	9	-2.15	127.00	49.44
		3	9	-	9	-1.53	9	-1.48	131.95	49.09
	251 ~ 500	0.5	10	-	10	-3.93	10	-3.65	420.60	287.51
		1	10	-	10	-5.34	10	-4.80	470.44	350.13
		1.5	10	-	10	-5.48	10	-4.98	496.25	378.89
		2	10	-	10	-5.14	10	-4.85	506.83	382.28
		2.5	10	-	10	-3.97	10	-3.83	515.56	374.43
		3	10	-	10	-3.42	10	-3.23	526.72	368.25
aTSPLIB	33 ~ 403	0.5	5	-	5	-28.86	5	-28.74	50.89	2.46
		1	5	-	5	-23.81	5	-23.69	54.31	2.70
		1.5	5	-	5	-18.38	5	-18.26	54.47	5.08
		2	5	-	5	-15.29	5	-15.22	49.10	1.64
		2.5	5	-	5	-12.07	5	-12.04	49.05	1.74
		3	5	-	5	-9.55	5	-9.51	47.52	1.31
Avg.				-		-5.14		-5.01	103.15	61.57
Total			325		321		321			

PROPESQ/PRPG/UFPB", proposal code PVL13394-2020. We also thank four anonymous referees for their valuable comments and suggestions which helped improve the paper.

Table 9: Comparison of aggregated results for instances with unknown optimal solutions

			HGS					
			best run		average			
n	β	# inst	# better	gap	# better	gap	TT	TB
35	1	1	1	-1.49	1	-1.49	1.82	0.48
	1.5	1	1	-4.63	1	-4.63	2.01	0.35
	2	2	2	-3.49	2	-3.44	1.67	0.14
	2.5	2	2	-2.21	2	-2.21	1.94	0.08
	3	4	3	-2.06	3	-2.06	1.80	0.05
40	1	1	1	-1.65	1	-1.65	1.74	0.04
	1.5	3	3	-2.81	3	-2.81	1.87	0.17
	2	3	3	-3.29	3	-3.29	2.00	0.05
	2.5	4	4	-3.25	4	-3.25	2.19	0.16
	3	4	4	-1.82	4	-1.82	2.30	0.05
45	0.5	1	0	0.00	0	0.00	1.74	0.06
	1	2	1	-5.11	1	-5.11	2.26	0.47
	1.5	3	3	-5.74	3	-5.65	2.16	0.15
	2	4	4	-5.77	4	-5.75	2.60	0.34
	2.5	4	4	-4.24	4	-4.24	2.53	0.06
	3	4	4	-2.87	4	-2.87	2.68	0.05
Avg.				-3.15		-3.14	2.08	0.17
Total		43	40		40			

References

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

Archetti, C., Feillet, D., Mor, A., & Speranza, M. G. (2020). Dynamic traveling salesman problem with stochastic release dates. *European Journal of Operational Research*, *280*, 832–844.

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

Beasley, J. (1983). Route first—cluster second methods for vehicle routing. *Omega*, *11*, 403–408. URL: https://www.sciencedirect.com/science/article/pii/0305048383900336. doi:https://doi.org/10.1016/0305-0483(83)90033-6.

Bulhões, T., Ha, M. H., Martinelli, R., & Vidal, T. (2018). The vehicle routing problem with service level constraints. *European Journal of Operational Research*, *265*, 544–558.

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

Dantzig, G. B., & Hamser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*, 80–91.

Davis, L. (1991). *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Commun. ACM*, *5*, 345.

Klapp, M. A., Erera, A. L., & Toriello, A. (2018a). The dynamic dispatch waves problem for same-day delivery. *European Journal of Operational Research*, *271*, 519–534.

Klapp, M. A., Erera, A. L., & Toriello, A. (2018b). The one-dimensional dynamic dispatch waves problem. *Transportation Science*, *52*, 229–496.

Li, W., Wu, Y., Kumar, P. N. R., & Li, K. (2020). Multi-trip vehicle routing problem with order release time. *Engineering Optimization*, *52*, 1279–1294.

Liu, L., Li, K., & Liu, Z. (2017). A capacitated vehicle routing problem with order available time in e-commerce industry. *Engineering Optimization*, *49*, 449–465.

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

Montero, A., Méndez-Díaz, I., & Miranda-Bront, J. J. (2021). *Solving the Traveling Salesman Problem with release dates via branch-and-cut*. Technical Report Universidad de Buenos Aires. URL: http://www.optimization-online.org/DB_HTML/2021/04/8350.html.

Mor, A., & Speranza, M. G. (2022). Vehicle routing problems over time: a survey. *Annals of Operations Research*, *314*, 255–275.

Pina-Pardo, J. C., Silva, D. F., & Smith, A. E. (2021). The traveling salesman problem with release dates and drone resupply. *Computers & Operations Research*, *129*, 105170.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, *31*, 1985–2002.

Reinelt, G. (1991). Tsplib—a traveling salesman problem library. *ORSA Journal on Computing*, *3*, 376–384.

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

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

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

Subramanian, A. (2012). *Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems*. Ph.D. thesis Programa de Pós-Graduação em Computação, Universidade Federal Fluminense.

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., & Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, *37*, 1899–1911.

Toth, P., & Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications* volume 2. Bologna: Siam.

Vidal, T. (2016). Technical note: Split algorithm in o(n) for the capacitated vehicle routing problem. *Computers & Operations Research*, *69*, 40–47.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*, 611–624.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, *234*, 658–673.

Yang, W., Ke, L., Wang, D. Z., & Lam, J. S. L. (2021). A branch-price-and-cut algorithm for the vehicle routing problem with release and due dates. *Transportation Research Part E: Logistics and Transportation Review*, *145*, 102167.

Highlights

- We develop a hybrid genetic algorithm that takes into account population diversity
- We propose a novel dynamic programming procedure for splitting giant tours
- We present a new neighborhood structure tailored for the TSPrd(time)
- Improved upper bounds were found for most of the benchmark instances