The double traveling salesman problem with partial last-in-first-out loading constraints

Jonatas B. C. Chagas[a,b,*], Túlio A. M. Toffolo[a], Marcone J. F. Souza[a], Manuel Iori[c]

[a]*Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, Brazil*
[b]*Departamento de Informática, Universidade Federal de Viçosa, Viçosa, Brazil*
[c]*DISMI, Università degli studi di Modena e Reggio Emilia, Reggio Emilia, Italy*

Abstract

In this paper, we introduce the Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL), a pickup-and-delivery single-vehicle routing problem where all pickup operations must be performed before any delivery one because the pickup and delivery areas are geographically separated. The vehicle collects items in the pickup area and loads them into its container, a horizontal stack. After performing all pickup operations, the vehicle begins delivering the items in the delivery area. Loading and unloading operations must obey a partial Last-In-First-Out (LIFO) policy, i.e., a version of the LIFO policy that may be violated within a given reloading depth. The objective of the DTSPPL is to minimize the total cost, which involves the total distance traveled by the vehicle and the number of reloaded items due to violations of the standard LIFO policy. We formally describe the DTSPPL by means of two Integer Linear Programming (ILP) formulations, and propose a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) to find high-quality solutions. The performance of the proposed solution approaches is assessed over a broad set of instances. Computational results have shown that both ILP formulations were able to solve only the smaller instances, whereas the BRKGA obtained better solutions for almost all instances, requiring shorter computational time.

Keywords: Pickup and delivery, Loading constraints, Partial reloading, Mathematical models, Genetic algorithm

1. Introduction

Routing problems have been widely studied due to their practical and theoretical relevance. Studies in recent years have shown that transportation logistics is not only an essential part of the distribution of goods but also a vital field in the increasingly globalized world economy (Kherbash and Mocan, 2015).

*Corresponding author.
Email address: jonatas.chagas@iceb.ufop.br (Jonatas B. C. Chagas)

Among the routing problems, stands out the classical and well-known Traveling Salesman Problem (TSP). This problem aims at finding the shortest Hamiltonian cycle in a weighted graph. The TSP is an $\mathcal{NP}$-Hard problem, nevertheless, many studies devoted to its solution and advances in the combinatorial optimization area made it possible to solve large problem instances (Applegate et al., 2006).

Many extensions and variations of the TSP have been introduced and solved by the scientific community. Among these, we mention two classes of problems that are particularly interesting for our work. In *Pickup and Delivery Problems*, the vehicle(s) transport(s) demands that have given origins and destinations (Battarra et al., 2014; Doerner and Salazar-González, 2014). In *Routing Problems with Loading Constraints*, the route(s) performed by the vehicle(s) must be compatible with some loading policy (Iori and Martello, 2010). These classes of problems have become very attractive because they are often noticed in the real world and also because of their combinatorial complexity.

A problem that combines both pickups, deliveries and loading is the Pickup and Delivery Traveling Salesman Problem with Last-In-First-Out Loading (PDTSPL), which arises in the context of a single vehicle, with a single access point (usually in the rear) for loading and unloading the transported items, which must serve a set of customers and all pickups and deliveries must obey the Last-In-First-Out (LIFO) policy.

More specifically, in the PDTSPL, each customer has a pickup location, where the items are loaded into the loading compartment (a horizontal stack) of the vehicle, and a delivery location, where the collected items must be delivered. At a pickup location, items are loaded and stored on top of the stack. At a delivery location, only the item located on the top of the stack is available for unloading and delivery. The PDTSPL aims to find the minimum cost tour to visit all pickup and delivery locations while ensuring that the LIFO policy is obeyed.

The PDTSPL is applicable in situations where items are loaded and unloaded from the rear of the vehicle and where relocations among items are forbidden, since the items may be large, heavy and/or fragile or dangerous to handle. By relocation, we mean the operation of temporarily dropping an already loaded item from the vehicle, and then reloading it back, possibly in a different position.

According to Iori and Martello (2010), the first work related to the PDTSPL was carried out by Ladany and Mehrez (1984), who addressed a real problem involving the transportation of milk containers. Posteriorly, the PDTSPL has also been studied by other researchers. Carrabs et al. (2007a) and Cordeau et al. (2010) presented exact algorithms, whilst Carrabs et al. (2007b) and Li et al. (2011) proposed heuristic methods.

Several other studies address pickup and delivery problems with LIFO loading constraints in scenarios where relocation operations are not allowed, such as the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks (PDTSPMS) (Côté et al., 2009) and the Double Traveling Salesman Problem with Multiple Stacks (DTSPMS) (Petersen and Madsen, 2009).

The PDTSPMS is a variant of the PDTSPL where the vehicle has its loading compartment divided into multiple horizontal stacks. On each stack, the loading and unloading operations must obey the LIFO principle. However, each stack is independent, so that

there are no mutual constraints among the stacks. In this way, there is an increase in the flexibility of loading/unloading operations, which can lead to a reduction in the operating costs of transport. The PDTSPMS was mainly treated by exact algorithms, which employ different and efficient branch-and-cut algorithms (Côté et al., 2009; Sampaio and Urrutia, 2016; Pereira and Urrutia, 2018).

Petersen and Madsen (2009) introduced the DTSPMS from a real-world case that a software company encountered with one of its customers. The DTSPMS is a particular case of the PDTSPMS, where the pickup and delivery operations must be completely separate because the pickup and delivery areas are geographically distant. Specifically, in the DTSPMS the vehicle starts its route in a pickup depot, performs all pickup operations in their stacks according to the LIFO principle, then returns to the pickup depot, from where the container is transferred by ship, airplane, train, or by similar transport form to a delivery depot located in the delivery area. From there, all delivery operations must be performed according to the LIFO principle. The objective of the DTSPMS is to find two routes, one in the pickup area and the other in the delivery area, so that the sum of the distances traveled in both areas is the minimum possible, while ensuring the feasibility of the vehicle loading plan. The transportation cost between the two regions is not considered as part of the optimization problem.

Since Petersen and Madsen (2009) presented it, the DTSPMS has aroused great attention in the academic community, with several exact and heuristic approaches. Felipe et al. (2009) proposed a heuristic approach based on the Variable Neighborhood Search (VNS) metaheuristic. Petersen et al. (2010) proposed different exact mathematical formulations, including a branch-and-cut algorithm. Lusby et al. (2010) presented an exact method based on matching k-best tours for each of the regions separately. Carrabs et al. (2010) developed a branch-and-bound algorithm for the Double Traveling Salesman Problem with Two Stacks (DTSP2S), a particular case of the DTSPMS in which the vehicle has exactly two stacks. Casazza et al. (2012) studied the theoretical properties of the DTSPMS, analyzing the structure of DTSPMS solutions in two separate components: routes and loading plan. Alba Martínez et al. (2013) proposed improvements to the branch-and-cut algorithm developed by Petersen et al. (2010), adding new valid inequalities that increased efficiency. Barbato et al. (2016) developed an exact algorithm which was able to solve instances involving containers of two stacks, which were not previously solved in the literature. We refer to Iori and Riera-Ledesma (2015), Silveira et al. (2015), Chagas et al. (2016) and Chagas et al. (2019) for details upon the Double Vehicle Routing Problem with Multiple Stacks (DVRPMS), a variant of the DTSPMS that considers multiple vehicles.

A real variant of the DTSPMS was suggested in Petersen's Ph.D. thesis (Petersen, 2009) (see page 75, Section 2.5.2.4), which we have named as the Double Traveling Salesman Problem with Multiple Stack and Partial Last-In-First-Out Loading Constraints (DTSPMSPL). According to Petersen (2009), the DTSPMSPL would have the same characteristics as the DTSPMS, although relocation operations are allowed as long as they obey a partial LIFO policy, which is a version of the LIFO policy that may be violated within a given reloading depth. This generalization of the DTSPMS is useful to model problems where the transported items are fragile and require secure fastening, so that their repositioning is possible,

but quite costly.

To the best of our knowledge, no study to date investigates the DTSPMSPL. Thus, in this work, we start this investigation, addressing a particular case of the DTSPMSPL where the vehicle has its loading compartment as a single horizontal stack. We have named this new transportation problem as the Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL).

The DTSPPL, like the DTSPMS, arises in transportation companies responsible for transporting large and fragile items from a pickup area to a delivery area, where these two areas are widely separated. However, in the DTSPPL, the repositioning of these items may be possible, albeit at a cost. Since replacing all items stored in the vehicle may be impractical due to the high handling cost and the limited space available during reloading, only a certain number of items may be placed outside the vehicle at any time. Hence, in the DTSPPL only the first L items from the top of the stack may be relocated at any time.

The objective of the DTSPPL is to find a route in each area, such that the total cost is the minimum possible and there is a feasible loading/unloading plan following the partial LIFO policy. The total cost involves the sum of traveled distance in both areas and the number of reloading operations performed in the loading/unloading plan.

In order to clarify the characteristics of the DTSPPL, we depict in Figure 1 a solution example of an instance with 6 customers, considering a reloading depth equal to 2, that is, $L = 2$. From the pickup depot (gray vertex on the left side of the figure), the vehicle starts its pickup route. It travels to the pickup position of customer 1, storing its item in the stack. Next, it visits the pickup position of customer 5, storing its item on the top of the horizontal stack. Then, it travels to pickup position customer 4, and at this point, a relocation is performed: the item of customer 5 is removed from the stack, then the item of customer 4 is placed in the stack and finally the item of customer 5 is replaced into the stack. The vehicle continues its pickup route as shown in the figure until all items have been collected and stored in the stack, then the vehicle returns to the pickup depot. Notice that the reloading sequence may be in any order; thus items do not need to remain in the same relative positions before their relocations, as occurs when the vehicle visits the pickup location of customer 6. Upon arrival at the pickup depot, the container is transferred to the depot (gray vertex on the right side of the figure) located in the delivery area, from where it is again transferred to a vehicle, which then starts the delivery operations. In our example, first, the vehicle travels to the delivery position of customer 2 without the requirement of any relocation. Next, it travels to the delivery position of customer 5, where items 6 and 3 need to be removed before delivering customer item 5. The delivery operations continue as shown in the figure until all customers are served. In the end, the vehicle returns to the delivery depot. Note that in this example there were 3 relocations in the pickup area (loading plan) and 3 ones in the delivery area (unloading plan), totalizing 6 relocations.

Figure 2 illustrates in a practical way the loading and unloading plan of the solution described in Figure 1. Note that each column in Figure 2 indicates the container configuration after each pickup or delivery operation. Note also that it is possible to determine which items have been relocated (they are highlighted in gray) and how they have been reloaded by analyzing adjacent pairs of container configurations.

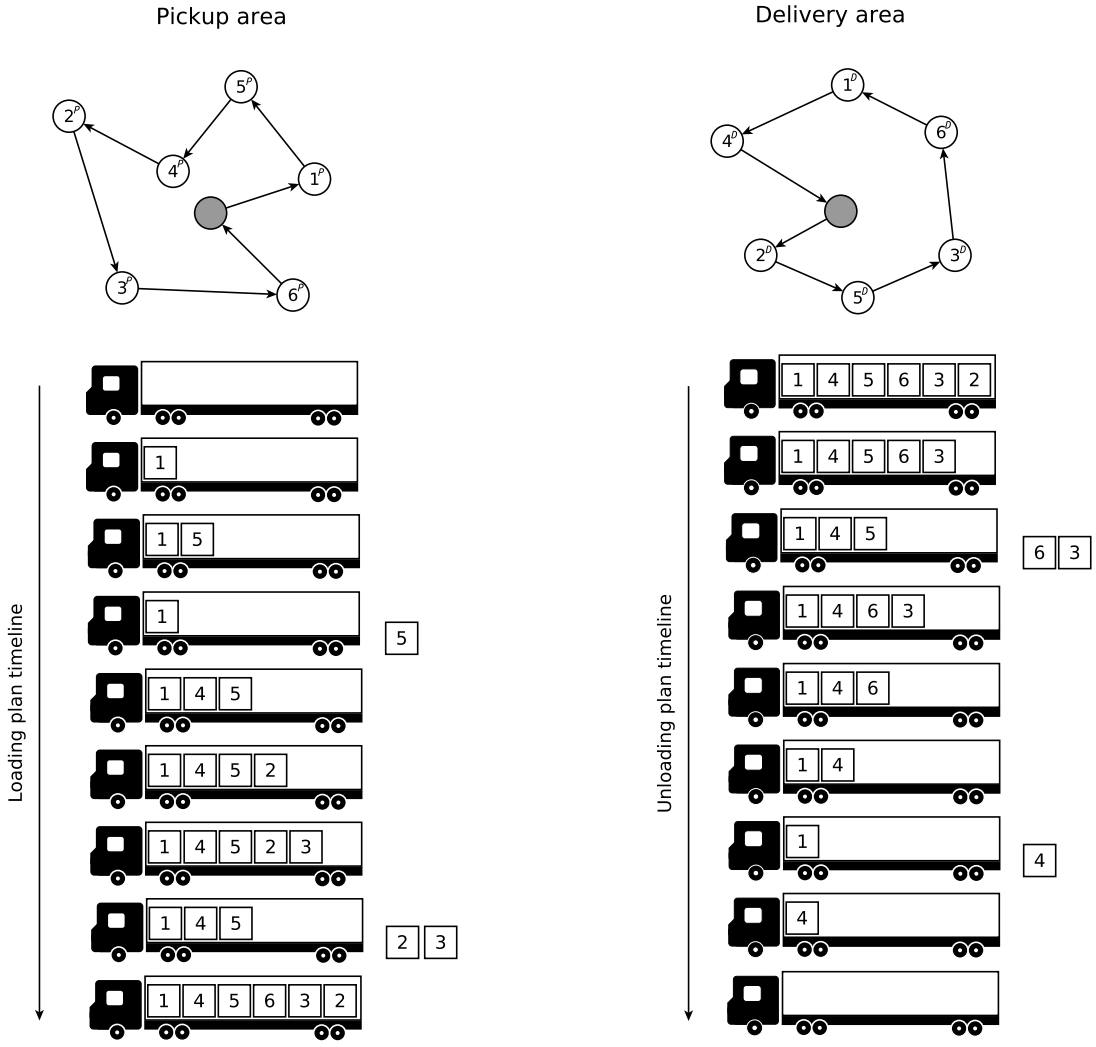

Figure 1: Solution of a DTSPPL instance involving 6 customers and reloading depth equal to 2.

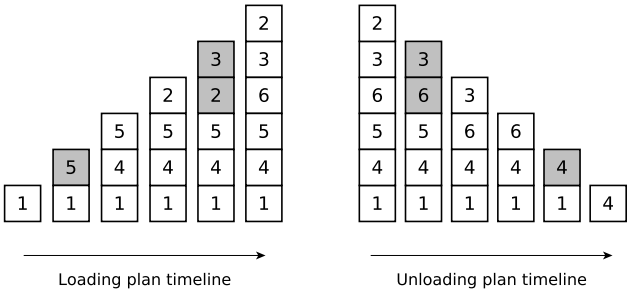

Figure 2: A practical representation of loading and unloading plan of the solution shown in Figure 1.

No previous work has approached the DTSPPL. Veenstra et al. (2017) approached a similar problem, named Pickup and Delivery Traveling Salesman Problem with Handling Costs (PDTSPH). It is a variant of the PDTSPL where relocation operations are allowed at delivery locations, and there is no maximum depth for reloading, i.e., at any delivery location, all items stored in the container may be relocated. The authors proposed a binary integer program for the PDTSPH considering that the reloading sequence is the inverse of the unloading sequence, i.e., the items remain in the same relative positions before their relocations. They have also developed a Large Neighborhood Search heuristic (LNS), which considers the reload policy adopted in the binary integer program, and another one where the reloaded items are positioned in the sequence in which they will be delivered. Their results show that this last reloading policy reduces the number of relocation operations since this strategy preventively sorts the reloaded items.

The remainder of this paper is structured as follows. In Section 2, we formally describe the DTSPPL via two Integer Linear Programming (ILP) formulations. In Section 3, we present a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) to obtain high-quality solutions. Section 4 reports the experiments and analyzes the performance of the proposed solution approaches. Finally, in Section 5 we present the conclusions and give suggestions for further investigations.

2. Problem description and mathematical formulations

In this section, we present the necessary notation to mathematically describe the DTSPPL and then propose two ILP formulations.

2.1. Problem description

The DTSPPL can be formally described as follows. Let $C = \{1, 2, ..., n\}$ be the set of n customer requests, $V_c^P = \{1^P, 2^P, ..., n^P\}$ the set of pickup locations and $V_c^D = \{1^D, 2^D, ..., n^D\}$ the set of delivery locations. For each customer request $i \in C$, an item has to be transported from the pickup location i^P to the delivery location i^D.

The DTSPPL is defined on two complete directed graphs, $G^P = (V^P, A^P)$ and $G^D = (V^D, A^D)$, which represent the pickup and delivery areas, respectively. The sets $V^P = \{0^P\} \cup V_c^P$ and $V^D = \{0^D\} \cup V_c^D$ represent the vertices in each area, with 0^P and 0^D denoting the depots of the pickup and delivery areas, respectively. The sets of arcs in pickup and delivery areas are defined by $A^P = \{(i^P, j^P, c_{ij}^P) \ \forall i^P \in V^P, \forall j^P \in V^P \mid j^P \neq i^P\}$ and $A^D = \{(i^D, j^D, c_{ij}^D) \ \forall i^D \in V^D, \forall j^D \in V^D \mid j^D \neq i^D\}$, where c_{ij}^P and c_{ij}^D correspond to the travel distances associated with arcs (i^P, j^P) and (i^D, j^D), respectively. For convenience of notation, when no confusion arises we also refer to sets V_c^P and V_c^D as the set of requests C, and we use i to denote both i^P and i^D and (i, j) to denote both (i^P, j^P) and (i^D, j^D).

A feasible solution s for the DTSPPL consists of a Hamiltonian cycle on graph G^P that starts at the pickup depot 0^P, another Hamiltonian cycle on graph G^D that begins at the delivery depot 0^D, and a loading/unloading plan. Besides, the two Hamiltonian cycles and the loading and unloading plan must obey the partial LIFO policy, which is defined by the maximum reloading depth L.

Let us denote by $\mathcal{F}$ the set of all feasible solutions for a DTSPPL instance. Each solution $s \in \mathcal{F}$ has a cost c_s that involves the travel distance on the two Hamiltonian cycles and the number of relocations performed on the loading and unloading plan, with a cost h associated with a single item relocation. The objective of the DTSPPL is to find a solution $s^* \in \mathcal{F}$, so that $c_{s^*} = \min_{s \in \mathcal{F}} c_s$.

To compare the complexity of the DTSPPL depending on the number of customers n and the reloading depth L, we calculate the total number of feasible solutions for a DTSPPL instance as described in the electronic Supplementary material (Appendix A) and we show a comparison of these numbers in Figure 3.

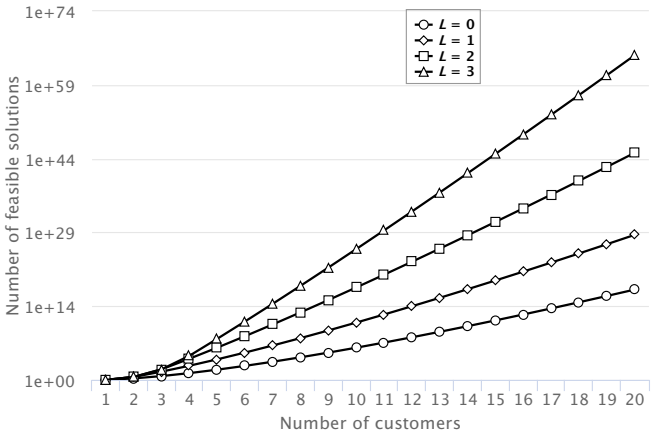

Figure 3: Number of feasible DTSPPL solutions.

Note that the number of feasible solutions is vast, even for small values of n. Moreover, it is higher for higher L values, which is easily noted since there are more options for relocations and consequently more viable routes.

2.2. Integer linear programming formulation 1

To better explain the first proposed ILP formulation (ILP1) for the DTSPPL, we categorize the constraints under three groups: (i) routes structuring constraints, (ii) loading/unloading plan and partial LIFO constraints, and (iii) reloading control constraints. After describing all constraints we present the objective function of the DTSPPL.

Throughout mathematical modeling, we also use the notation $[a, b]$ to denote the set $\{a, a+1, ..., b-1, b\}$. Note that for any $a > b$, $[a, b]$ is an empty set.

2.2.1. Routes structuring constraints

In order to characterize the pickup and delivery routes, we define constraints (1)-(7), which use a binary decision variable $x_{ij}^{kr} \quad \forall r \in \{P, D\}$, $k \in [1, n+1]$, $(i, j) \in A^r$ that assumes value 1 if arc (i, j) is the k-th traveled arc by the vehicle in the area r, and value 0 otherwise.

$$\sum_{j\,:\,(0,\,j)\,\in\,A^r} x_{0j}^{1r} = 1 \qquad r \in \{P, D\} \tag{1}$$

$$\sum_{i\,:\,(i,\,0)\,\in\,A^r} x_{i0}^{n+1,r} = 1 \qquad r \in \{P, D\} \tag{2}$$

$$\sum_{(i,\,j)\,\in\,A^r} x_{ij}^{kr} = 1 \qquad r \in \{P, D\},\ k \in [2, n] \tag{3}$$

$$\sum_{k\,\in\,[1,\,n]} \sum_{i\,:\,(i,\,j)\,\in\,A^r} x_{ij}^{kr} = 1 \qquad r \in \{P, D\},\ j \in V_c^r \tag{4}$$

$$x_{ij}^{kr} \leq \sum_{i'\,:\,(i',\,i)\,\in\,A^r} x_{i'i}^{k-1,r} \qquad r \in \{P, D\},\ k \in [2, n+1],\ (i,j) \in A^r \tag{5}$$

$$x_{ij}^{kr} \in \{0, 1\} \qquad r \in \{P, D\},\ k \in [1, n+1],\ (i,j) \in A^r \tag{6}$$

Constraints (1) and (2) force the vehicle to leave from the depot and return to it using, respectively, the first and the last arc in each area. Constraints (3) guarantee that only a single arc may be the k-th one of each route. Constraints (4) guarantee that every customer is served. Constraints (5) establish the flow conservation for each vertex, and constraints (6) define the scope and domain of the decision variables used to represent the routes.

2.2.2. Loading/unloading plan and partial LIFO constraints

For the purpose of representing the loading plan, we define a binary decision variable y_{jl}^{kP} $\forall k \in [1, n]$, $l \in [1, k]$, $j \in C$, that assumes value 1 if the item referring to the customer request j is stored in position l on the k-th container configuration in the pickup area, and value 0 otherwise. We also define another binary decision variable y_{jl}^{kD} $\forall k \in [1, n]$, $l \in [1, n-k+1]$, $j \in C$, to represent the unloading plan, which has the same meaning as the previous variable but considers the delivery area. With these variables, we can ensure the feasibility of the loading and unloading plans throughout constraints (7)-(15), which are next explained in detail.

$$\sum_{j\,\in\,C} y_{jl}^{kP} = 1 \qquad k \in [1, n],\ l \in [1, k] \tag{7}$$

$$\sum_{j\,\in\,C} y_{jl}^{kD} = 1 \qquad k \in [1, n],\ l \in [1, n-k+1] \tag{8}$$

$$\sum_{l\,\in\,[1,\,k]} y_{jl}^{kP} = \sum_{k'\,\in\,[1,\,k]} \sum_{i\,:\,(i,\,j)\,\in\,A^P} x_{ij}^{k'P} \qquad k \in [1, n],\ j \in C \tag{9}$$

$$\sum_{l\,\in\,[1,\,n-k+1]} y_{jl}^{kD} = \sum_{k'\,\in\,[k,\,n]} \sum_{i\,:\,(i,\,j)\,\in\,A^D} x_{ij}^{k'D} \qquad k \in [1, n],\ j \in C \tag{10}$$

Constraints (7) and (8) ensure that only one item must occupy each container position in each of its configurations. Constraints (9) and (10) are graphically represented in Figures

4 and 5, respectively, considering an instance with 6 customer requests. Note that in the pickup area the k-th container configuration has to contain all items collected at the vertices $v_i^P \quad \forall i \leq k$, whilst in the delivery area the k-th container configuration has to contain all items that have not yet been delivered at the vertices $v_i^D \quad \forall i \geq k$.

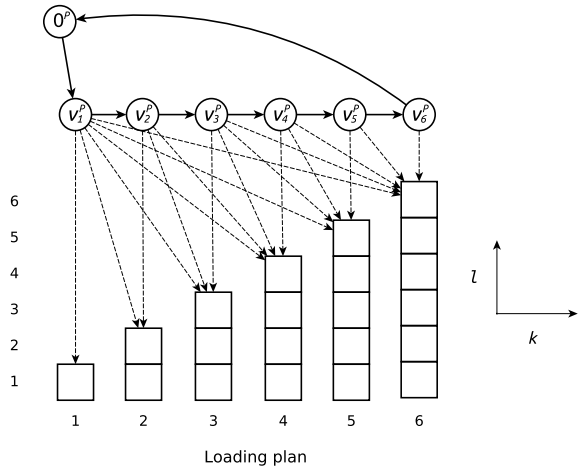

Figure 4: Graphical representation of constraints (9).

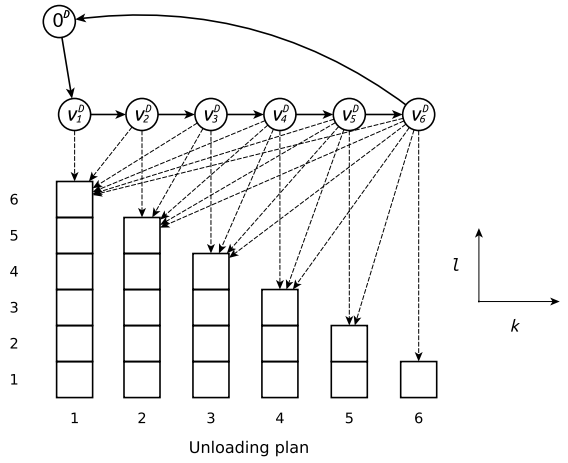

Figure 5: Graphical representation of constraints (10).

Constraints (11) certify that the first container configuration in the delivery area must be the same one as the last container configuration in the pickup area. In other words, these constraints ensure that there is no relocation of items between the transfer from the pickup depot to the delivery depot.

$$y_{jl}^{1D} = y_{jl}^{nP} \qquad l \in [1, n], j \in C \tag{11}$$

Figure 6 illustrates the behavior of constraints (11) for an instance with 6 customer requests. It also illustrates the behavior of constraints (12) and (13) that ensure the loading plan obeys the partial LIFO policy, taking as example $L = 2$. Note that constraints (12) and (13) establish which items have to remain in their previous container positions in order to not violate the partial LIFO policy.

$$y_{jl}^{kP} = y_{jl}^{nP} \qquad k \in [1, n-1],\ l \in [1, k-L], j \in C \tag{12}$$

$$y_{jl}^{kD} = y_{jl}^{1D} \qquad k \in [2, n],\ l \in [1, n-k-L+1],\ j \in C \tag{13}$$

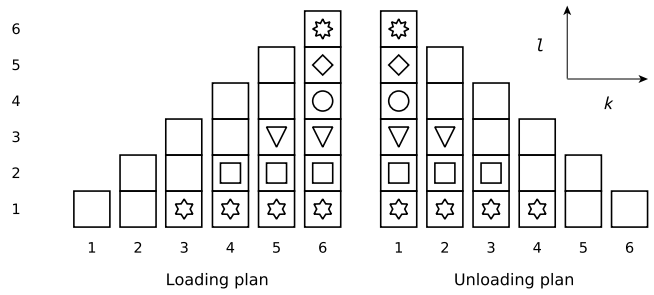

Figure 6: Graphical representation of constraints (11)-(13).

Finally, constraints (14) and (15) define the scope and domain of the decision variables used to represent the loading and unloading plan.

$$y_{jl}^{kP} \in \{0, 1\} \qquad k \in [1, n],\ l \in [1, k],\ j \in C \tag{14}$$

$$y_{jl}^{kD} \in \{0, 1\} \qquad k \in [1, n],\ l \in [1, n-k+1],\ j \in C \tag{15}$$

2.2.3. Reloading control constraints

To determine how many relocations are performed from the loading and unloading plans, we define a decision variable $z^{kr} \ \ \forall r \in \{P, D\}, k \in [1, n-1]$ that indicates the number of relocations made in the k-th container configuration in area r. We also define constraints (16) and (17), which are responsible for determining the number of relocations in the loading and unloading plans, respectively. Constraints (18) define the scope and domain of these decision variables.

$$z^{kP} \geq \left(y_{jl}^{kP} - y_{jl}^{k+1,P} \right) \cdot (\, k - l + 1\,) \qquad k \in [1, n-1],\ l \in [1, k-1],\ j \in C \tag{16}$$

$$z^{kD} \geq \left(y_{jl}^{k+1,D} - y_{jl}^{kD} \right) \cdot (\, n - k - l + 1\,) \qquad k \in [1, n-1],\ l \in [1, n-k],\ j \in C \tag{17}$$

$$z^{kr} \in \mathbb{Z}^{+} \qquad r \in \{P, D\},\ k \in [1, n-1] \tag{18}$$

2.2.4. Objective function

Constraints (1)-(18) are enough to represent all feasible solutions of the DTSPPL. Therefore, to complete the first mathematical model, we define the objective function (19), which minimizes the total cost. It involves the distance traveled in both areas, as well as the cost of all relocations performed.

$$\min \sum_{r \in \{P, D\}} \sum_{(i,j) \in A^r} c_{ij}^r \cdot \sum_{k \in [1, n+1]} x_{ij}^{kr} + h \cdot \sum_{r \in \{P,\, D\}} \sum_{k \in [1,\, n-1]} z^{kr} \tag{19}$$

2.3. Integer linear programming formulation 2

Our second ILP formulation (ILP2) uses a binary decision variable $\chi_{ij}^r \ \forall r \in \{P, D\}$, $(i, j) \in A^r$ to describe the route of the vehicle in each area. More specifically, each variable χ_{ij}^r assumes value 1 if arc (i, j) is traveled by the vehicle in area r, and value 0 otherwise. Moreover, we use an integer variable $u_j^r \ \forall r \in \{P, D\}$, $j \in [0, n]$ that gives the position of the vertex j in the route of area r.

With these new decision variables, we can use constraints (20)-(25), instead of (1)-(6) adopted for ILP1, to define the routes of the vehicle.

$$\sum_{j\,:\,(i,\,j) \in A^r} \chi_{ij}^r = 1 \qquad r \in \{P, D\},\ i \in V^r \tag{20}$$

$$\sum_{i\,:\,(i,\,j) \in A^r} \chi_{ij}^r = 1 \qquad r \in \{P, D\},\ j \in V^r \tag{21}$$

$$u_j^r \geq u_i^r + 1 - n \cdot \left(1 - \chi_{ij}^r\right) \qquad r \in \{P, D\},\ (i, j) \in A^r : j \neq 0 \tag{22}$$

$$\chi_{ij}^r \in \{0, 1\} \qquad r \in \{P, D\},\ (i, j) \in A^r \tag{23}$$

$$u_0^r = 0 \qquad r \in \{P, D\} \tag{24}$$

$$u_j^r \in \{a \in \mathbb{Z}^+ : a \leq n\} \qquad r \in \{P, D\},\ j \in C \tag{25}$$

Constraints (20), (21) and (23) ensure that each pickup and delivery location is visited exactly once, whilst constraints (22), (24) and (25) ensure the subcycle elimination.

To complete the ILP2 model, we define constraints (26) and (27), and the objective function (28). Moreover, we also consider constraints (7), (8) and (11)-(18), which have been previously defined for ILP1.

$$\sum_{l \in [1,\, k]} y_{jl}^{kP} \geq \frac{k - u_j^P + 1}{k} \qquad k \in [1, n],\ j \in C \tag{26}$$

$$\sum_{l \in [1,\, n-k+1]} y_{jl}^{kD} \geq \frac{u_j^D - k + 1}{n - k + 1} \qquad k \in [1, n],\ j \in C \tag{27}$$

Note that constraints (26) and (27) have the same aim as constraints (9) and (10), which ensure the correct assignment of items to the loading compartment throughout the pickup

and delivery routes. Note also that equation (28), similarly to (19), describes the total DTSPPL cost to be minimized.

$$\min \sum_{r \in \{P, D\}} \sum_{(i,j) \in A^r} c_{ij}^r \cdot \chi_{ij}^r \; + \; h \cdot \sum_{r \in \{P,\, D\}} \sum_{k \in [1,\, n-1]} z^{kr} \tag{28}$$

2.4. Providing the ILP models with a feasible initial solution

We solved models ILP1 and ILP2 using Gurobi Optimizer, which is one of the best optimization solvers for ILP and other types of mathematical models. However, we noticed that even for small instances Gurobi had difficulties in solving either models within a reasonable time. Therefore, to try and help the optimization process, we decided to compute a feasible DTSPPL solution and initialize the models with it.

To generate this initial solution, we consider a particular DTSPPL case, named Double Traveling Salesman Problem (DTSP), where relocations are not allowed ($L = 0$). Thus, as every loading/unloading operations must verify the classic LIFO principle, the pickup and delivery routes must be exactly opposite each other. Therefore, we can solve a DTSP instance by solving a TSP instance on a graph $G = (V, A)$, where $V = V^P$ is the set of vertices, $A = A^P$ is the set of arcs, and with each arc $(i,\, j) \in A$ is associated a cost $c_{ij} = c_{ij}^P + c_{ji}^D$. We can then formulate the DTSP using model (29)-(33), which is the classical two-index model for the TSP (Gutin and Punnen, 2006). It uses a binary decision variable $\delta_{ij} \;\; \forall (i, j) \in A$ that assumes value 1 if arc $(i,\, j) \in A$ is traveled (in other words, if arcs $(i,\, j) \in A^P$ and $(j,\, i) \in A^D$ are traveled in the original pickup area and delivery areas, respectively), and 0 otherwise.

$$\min \sum_{(i,\, j) \in A} c_{ij} \cdot \delta_{ij} \tag{29}$$

$$\sum_{j \,:\, (i,\, j) \in A} \delta_{ij} = 1 \qquad\qquad i \in V \tag{30}$$

$$\sum_{i \,:\, (i,\, j) \in A} \delta_{ij} = 1 \qquad\qquad j \in V \tag{31}$$

$$\sum_{(i,\, j) \in A \,:\, \{i,\, j\} \in S} \delta_{ij} \leq |S| - 1 \qquad\qquad S \subset V : |S| \geq 2 \tag{32}$$

$$\delta_{ij} \in \{0, 1\} \qquad\qquad (i, j) \in A \tag{33}$$

The objective function (29) minimizes the total distance traveled. Constraints (30), (31) and (33) guarantee that each customer is visited only once, and constraints (32) ensure subtour elimination.

We solved the model (29)-(33) using a simple and efficient strategy for moderate TSP instances. In this strategy, we initially create a model with objective function (29), constraints (30), (31), (33), and only the subset of constraints (32) having $|S| = 2$. Then, we solve the model and check if its integer solution contains subtours. If there is no subtour, the

solution is an optimal TSP tour and, consequently, an optimal DTSP solution. Otherwise, we find all subtours in the integer solution, add the corresponding elimination constraints to the model and solve it again. The process is repeated until there is no subtour in the integer solution.

3. A biased random-key genetic algorithm

In this section, we describe a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) (Gonçalves and Resende, 2011), which is an evolutionary metaheuristic that has been successfully applied to several complex optimization problems (Gonçalves and Resende, 2012, 2013, 2015; Santos and Chagas, 2018).

BRKGAs, as well as classic Genetic Algorithms (GAs) (see (Mitchell, 1998) for a useful reference), are evolutionary metaheuristics that mimic the processes of Darwinian Evolution. Basically, a GA maintains a population of individuals, where each individual encodes a solution to the problem at hand. Through the use of stochastic evolutionary processes (selection, recombination, and diversification) over the population, individuals with higher fitness tend to survive, thus guiding the algorithm to explore more promising regions of the solutions space.

In the remainder of this section, we present the main components of the proposed BRKGA (Sections 3.1-3.6) and then describe how these components are combined together (Section 3.7).

3.1. Encoding structure

Each individual in BRKGAs is represented by a vector of random-keys, i.e., a vector of real numbers that assume values in the continuous interval [0, 1]. This representation is generic because it is independent of the problem addressed. Therefore, a deterministic procedure (to be presented later) is necessary to decode each individual (a vector of random-keys) to a feasible solution of the problem at hand (DTSPPL in our case).

In Figure 7, we show the structure defined to represent each BRKGA individual for the DTSPPL. We divide this structure into three partitions: pickup route, loading plan, and unloading plan and delivery route. The first one consists of n random-keys, which are responsible for determining the pickup route. The next $\sum_{k=1}^{n} \min(k,\ L+1)$ random-keys determine the loading plan carried out along the pickup route. Finally, the last $\sum_{k=1}^{n} \min(n-k+1, L+1)$ random-keys define the entire unloading plan and also the delivery route.

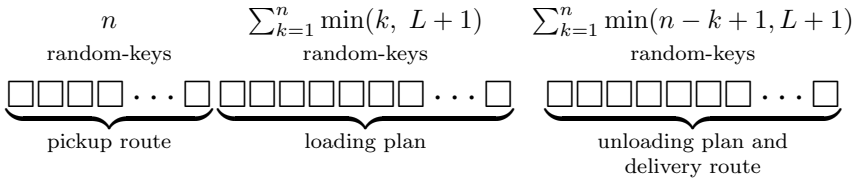

Figure 7: Chromosome structure.

3.2. Decoding procedure

As stated before, each individual $p \in \mathcal{P}$ has a generic representation in a BRKGA. Therefore, to determine the fitness of p, we developed a procedure that generates a feasible solution s from p. Thus, the fitness of p is defined proportionally to the quality of s.

Our decoding procedure consists of three stages, which must be sequentially performed due to the dependency among them. Figure 8 depicts how these stages are performed in order to decode the solution shown in Figure 1 from a vector of random-keys. Initially, the n pickup locations are mapped on the first n random-keys. Then, we sort the pickup locations according to the values of the mapped random-keys. The sorted pickup locations define the pickup route π^P performed by the vehicle. In Figure 1, the first n random-keys produce the pickup route $\pi^P = \langle 0^P, 1^P, 5^P, 4^P, 2^P, 3^P, 6^P \rangle$.

We characterize the loading plan from π^P and the next $\sum_{k=1}^{n} \min(k,\ L+1)$ random-keys. The loading plan is created iteratively in n steps, where each of them depends on the previous one. Iteratively, for each $k \in \{1, 2, ..., n\}$, we map the $\min(k,\ L+1)$ items that may be relocated without violating the partial LIFO constraints to $\min(k,\ L+1)$ random-keys. Next, we sort the items in non-decreasing order according to the values of the mapped random-keys in order to define the k-th container configuration in the pickup area. To be clearer, consider the steps shown in Figure 8. At first, the container is empty, so we just store in it the item (highlighted in gray) of the first customer visited on π^P. Notice that, though unnecessary, we kept a random-key (0.48 in this example) to decode the first operation of the loading plan. We decided to keep it for simplicity, so as to follow the same pattern used for the other loading operations. To define the second container configuration, we map item 1 (it may be relocated from the previous container configuration) to the random-key 0.59 and item 5 (the second customer visited on π^P) to the random-key 0.61. After sorting these items according to the values of the mapped random-keys, items 1 and 5 are stored in the container following the sorted order. The decoding process continues by mapping the items that may be relocated at each time to the random-keys and then sort them to define each remaining pickup container configuration.

Finally, the unloading plan and the delivery route are defined from the loading plan and the last random-keys. Similarly as before, it is created iteratively in n steps, where each of one is dependent on the previous one. For each $k \in \{1, 2, ..., n\}$ we map the $\min(n - k + 1, L + 1)$ items that may be delivered without violating the partial LIFO constraints to $\min(n - k + 1, L + 1)$ random-keys. Then, we sort the items in non-decreasing order according to the values of the mapped random-keys. The $\min(n - k + 1, L + 1) - 1$ first sorted items define the k-th container configuration in the delivery area and the last one is delivered, iteratively making up the delivery route. In Figure 8, take for example the first step ($k = 1$), where we map items 6, 3 and 2 (only these items may be relocated because in this case, the reloading depth is 2) to the random keys 0.68, 0.94 and 0.95, respectively. After sorting these random-keys, item 2 (highlighted in gray) that is mapped to the greatest random-key (0.95 in this example) is delivered. While other items are stored in the container following the order of their random-keys. This process is repeated until the whole unloading plan has been completed.

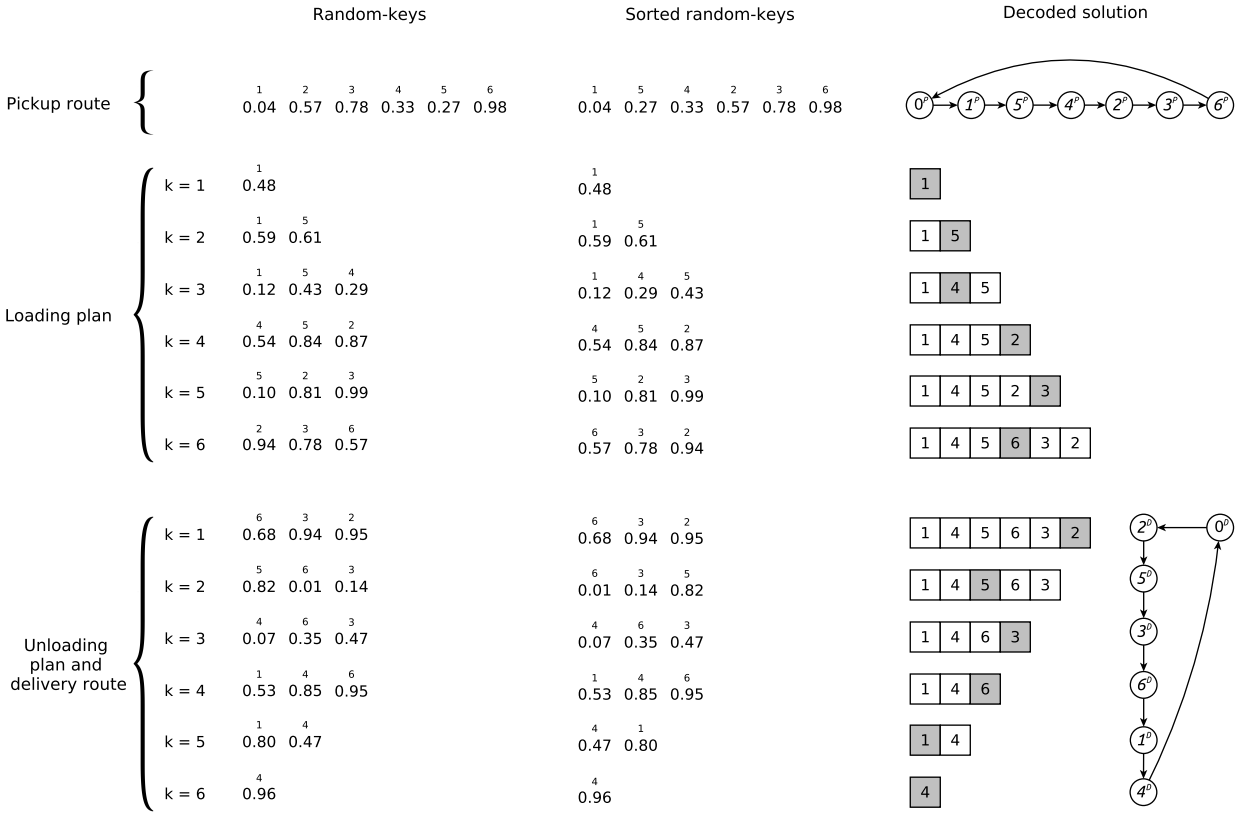

Figure 8: Decoding of the vector of random-keys $\langle$ 0.04, 0.57, 0.78, 0.33, 0.27, 0.98, 0.48, 0.59, 0.61, 0.12, 0.43, 0.29, 0.54, 0.84, 0.87, 0.10, 0.81, 0.99, 0.94, 0.78, 0.57, 0.68, 0.94, 0.78, 0.57, 0.68, 0.94, 0.78, 0.57, 0.68, 0.94, 0.95, 0.82, 0.01, 0.14, 0.07, 0.35, 0.47, 0.53, 0.85, 0.95, 0.80, 0.47, 0.96 $\rangle$ to the solution shown in Figure 1.

3.3. Initial population

Our BRKGA maintains a population $\mathcal{P}$ of N individuals throughout the evolutionary process. To make the initial population, we create $N - 1$ random individuals, where each random-key of each individual is generated independently at random in the real interval $[0, 1]$. Furthermore, in order to introduce an orientation (maybe a good one) to the search procedure of the BRKGA, we create and insert into the initial population an individual that represents the solution of model (29)-(33), i.e., the optimal DTSPPL solution in which no relocations are performed. This last individual is created following the reverse process of the decoding procedure previously described.

3.4. Biased crossover

In order to combine the genetic information of parents to generate new offsprings, a BRKGA uses a biased crossover operator. This crossover always involves two parents, where one is randomly selected from the elite group $\mathcal{P}_e$ and the other is randomly chosen from the non-elite group $\mathcal{P}_{\bar{e}}$. The groups $\mathcal{P}_e$ and $\mathcal{P}_{\bar{e}}$ are formed at each generation of the algorithm

after all individuals of population $\mathcal{P}$ have been decoded. Group $\mathcal{P}_e$ is formed by the individuals with greater fitness, while $\mathcal{P}_{\bar{e}}$ is formed by the others ones, i.e., $\mathcal{P}_{\bar{e}} = \mathcal{P} \setminus \mathcal{P}_e$. Moreover, the biased crossover operator has a parameter ρ_e which defines the probability of each random-key of the elite parent to be inherited by the offspring individual. More precisely, from an elite parent a and a non-elite parent b, we can generate an offspring c according to the biased crossover as follows:

$$c_i \leftarrow \begin{cases} a_i & \text{if } random\,(0,\,1) \;\leq\; \rho_e \\ b_i & \text{otherwise} \end{cases} \qquad \forall i \in \{1, 2, ..., M\}$$

where M is the number of random-keys of the individuals and a_i, b_i and c_i are, respectively, the i-th random-key of individuals a, b and c.

3.5. Mutant individuals

Unlike most GAs, BRKGAs do not contain mutation operators. Instead, to maintain population diversity, the BRKGAs use mutant individuals, which are merely new individuals generated by choosing for each random-key a real number between 0 and 1.

3.6. Next generation

In the BRKGA schema, from any generation k a new population is formed based on the current population $\mathcal{P}$. First, all elite individuals of generation k in $\mathcal{P}_e$ are copied into the new population (generation $k+1$) without any modification. Next, in order to maintain a high diversity of the population, some mutant individuals are added into the new population. Finally, to complete the new population, new individuals are added by using the biased crossover operator. Figure 9 illustrates the transition from generation k to generation $k+1$.

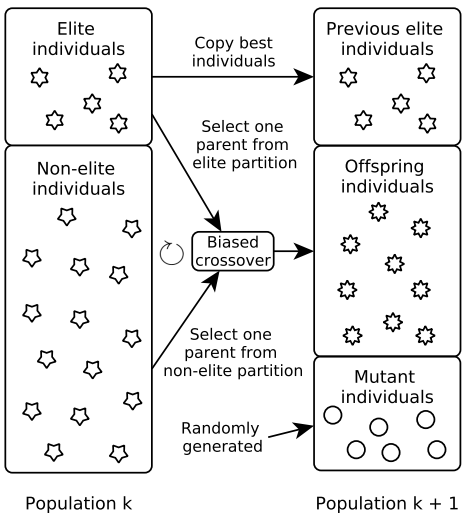

Figure 9: Transition from generation k to generation $k+1$.

3.7. Overall BRKGA

The previous components are organized as described in Algorithm 1 to build up the BRKGA. Initially (line 1), the best solution found by the algorithm is initialized as an empty solution, i.e., no solution has been found so far. At line 2, the initial population is generated. While the stopping criterion is not achieved, the algorithm performs its evolutionary cycle (lines 3 to 12). At lines 4 to 7, all individuals in the current population are decoded and the best solution found is possibly updated. After selecting the individual elites (line 8) and generating the mutant individuals (line 9) as well as the offspring individuals (line 10), the algorithm updates the population of individuals (line 11). At the end of the algorithm (line 13), the best solution found is returned.

Algorithm 1: Biased random-key genetic algorithm

```
1  s^best ← ∅
2  P ← initial population with N individuals
3  repeat
4  |  foreach p ∈ P do
5  |  |  s ← individual p decoded
6  |  |  if s is better than s^best then s^best ← s end
7  |  end
8  |  P_e ← set of the N_e best individuals (elite) from P
9  |  P_m ← set of N_m mutant individuals
10 |  P_o ← set of N − N_e − N_m offspring individuals
11 |  P ← P_e ∪ P_m ∪ P_o
12 until time limit is reached
13 return s^best
```

4. Computational experiments

We herewith present all computational experiments used to study the performance of the proposed solution approaches. As there is no previous work about the DTSPPL, we opted to compare the proposed approaches to each other.

Mathematical models ILP1 and ILP2 were coded in C/C++ language using Gurobi solver version 8.0.1. The proposed heuristic algorithm was coded in C/C++ language from the BRKGA framework developed by Toso and Resende (2015). All experiments were sequentially (nonparallel) performed on an Intel Core i7-4790K CPU @ 4.00 GHz x 8 desktop computer with 32GB RAM, running under Ubuntu 16.04 LTS 64 bits.

In the remainder of this section, we present and discuss the main results obtained from the computational experiments, whereas all detailed results can be found in the electronic Supplementary material for the paper (Appendix B). Moreover, in the electronic Supplementary material (Appendix C), we report the best known solution (routes and loading/unloading plan) obtained so far for each DTSPPL instance.

4.1. Test instances

The experiments were performed on a comprehensive set of instances built from the DTSPMS benchmark instances proposed by Petersen and Madsen (2009).

We have defined 48 types of instances, where each type is described by the number of customers n, the reloading depth L, and the cost of each relocation h. The number of customers n varies in $\{6, 8, 10, 12, 14, 16, 18, 20\}$, while the reloading depth L and the cost h vary in $\{1, 2, 3\}$ and $\{5, 10\}$, respectively.

For each type of instances, we have used the areas R05, R06, R07, R08, and R09 defined by Petersen and Madsen (2009). Each of these areas consists of two sets, where one defines the pickup locations (pickup region), and the other defines the delivery locations (delivery region). The locations of each region were generated randomly in a 100×100 square. The distance between any two points of each region is the Euclidean distance rounded to the nearest integer, following the conventions from TSPLIB. The first point of each region, which is fixed in coordinates $(50, 50)$, corresponds to the depot, while the next n points define the n customer locations.

Therefore, by combining all type of instances with the five areas, we have obtained 240 DTSPPL instances. Each of them is identified by `RAA_BB_CC_DD`, where `RAA`, `BB`, `CC` and `DD`, inform, respectively, the area name (R05 to R09), the number of customers n, the reloading depth L and the cost of each relocation h.

4.2. Parameter settings

Our mathematical models ILP1 and ILP2 were performed using Gurobi solver with all its default settings, except for the runtime that was limited to 1 hour, and for the optimization process that was limited to use only a single processor core.

Regarding the parameters of the BRKGA, we use as stopping criterion the execution time equal to M seconds, which is the number of random-keys of each individual. This value is proportional to the number of customers n and the reloading depth L, thus we give more time to the instances with larger space of solutions (see Figure 3). For the other BRKGA parameters, we use the automatic configuration method I/F-Race (Birattari et al., 2010) in order to find the most suitable configuration. We use the implementation of I/F-Race provided by the Irace package (López-Ibáñez et al., 2016), which was implemented in R language and is based on the iterated racing procedure.

Table 1 describes the BRKGA parameters as well as the tested values for each of them. Note that the population size N is given in terms of the size of each individual. Moreover, the elite population size N_e and mutant population size N_m are granted in terms of N.

After a vast experiment that used a sample of 10% of all 240 instances, Irace pointed out the parameters highlighted in bold in Table 1 as the best ones.

4.3. ILP1 vs. ILP2

Our first computational experience contrasts mathematical models ILP1 and ILP2. We begin by comparing the performance of each model regarding the number of optimal solutions found when both models were limited to 1 hour of processing time. The results that we

Table 1: BRKGA parameters.

Parameter	Description	Tested values
N	population size	$1M, 2M, 5M, 10M, 20M, 50M, 100M, \mathbf{200M}, 500M$
N_e	elite population size	$0.05N, 0.10N, \mathbf{0.15N}, 0.20N, 0.25N, 0.30N$
N_m	mutant population size	$0.05N, 0.10N, 0.15N, \mathbf{0.20N}, 0.25N, 0.30N$
ρ_e	elite allele inheritance probability	$0.55, 0.60, 0.65, \mathbf{0.70}, 0.75, 0.80, 0.85, 0.90$

M is the number of random-keys of each individual.

obtained are shown in Table 2, where each cell reports the number of instances solved to proven optimality by ILP1 (left) and ILP2 (right), considering each specific type of instance.

Table 2: Comparison of the number of instances solved to proven optimality by each model (ILP1 *vs.* ILP2).

n	$L = 1$		$L = 2$		$L = 3$	
	$h = 5$	$h = 10$	$h = 5$	$h = 10$	$h = 5$	$h = 10$
6	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5
8	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5	5 *vs.* 5	5 *vs.* 4
10	5 *vs.* 2	5 *vs.* 0	1 *vs.* 1	1 *vs.* 0	0 *vs.* 0	0 *vs.* 0
12 - 20	0 *vs.* 0	0 *vs.* 0	0 *vs.* 0	0 *vs.* 0	0 *vs.* 0	0 *vs.* 0

It can be noticed from Table 2 that the proposed mathematical models were able to solve all instances with up to 8 customers, except for one case ($n = 8; L = 3; h = 10$ for ILP2). Both models have presented difficulties in solving instances with 10 or more customers, especially with the increase of the reloading depth and the cost of each relocation. However, we observe that model ILP1 was able to solve more instances than ILP2: 72 vs. 62 in total.

In order to better investigate the behavior of the models, we examined the performance of their lower bounds. For each model and for each instance, we consider the lower bound obtained at the root node of the search tree (linear relaxation) and also the one reached at the end of the computation. These values are referenced as $\texttt{ILP1\_LB}_0$ and ILP1_LB, respectively, for model ILP1, and as $\texttt{ILP2\_LB}_0$ and ILP2_LB, respectively, for model ILP2.

We apply the Relative Percentage Difference (RPD) metric to measure the percentage variation of a specific lower bound ($\texttt{LB}^{\texttt{s}}$) to the largest one ($\texttt{LB}^{\texttt{b}}$) found. This metric, named RPD_LB, is calculated as $|(\texttt{LB}^{\texttt{s}} - \texttt{LB}^{\texttt{b}})| \ / \ \texttt{LB}^{\texttt{b}} \times 100\%$. Figure 10 shows the RPD_LB of the lower bounds $\texttt{ILP1\_LB}_0$, ILP1_LB, $\texttt{ILP2\_LB}_0$, and ILP2_LB for all 240 test instances. We divided and plotted these results into six charts, wherein each of them all instances with the same reloading depth and relocation cost are grouped.

We observe from Figure 10 that, when considering the lower bounds achieved, model ILP2 is more effective than ILP1 for most of the instances. Note indeed that both $\texttt{ILP2\_LB}_0$ and ILP2_LB are higher than $\texttt{ILP1\_LB}_0$ and ILP1_LB, respectively, for almost all instances. Note also that for many instances even the linear relaxation of model ILP2 ($\texttt{ILP2\_LB}_0$) is tighter than the lower bound of model ILP1 at the end of one hour of processing time (ILP1_LB). In addition, we observe that the most instances in which model ILP1 has found the better ILP1_LB values are those that involve 10 customers. These instances are the

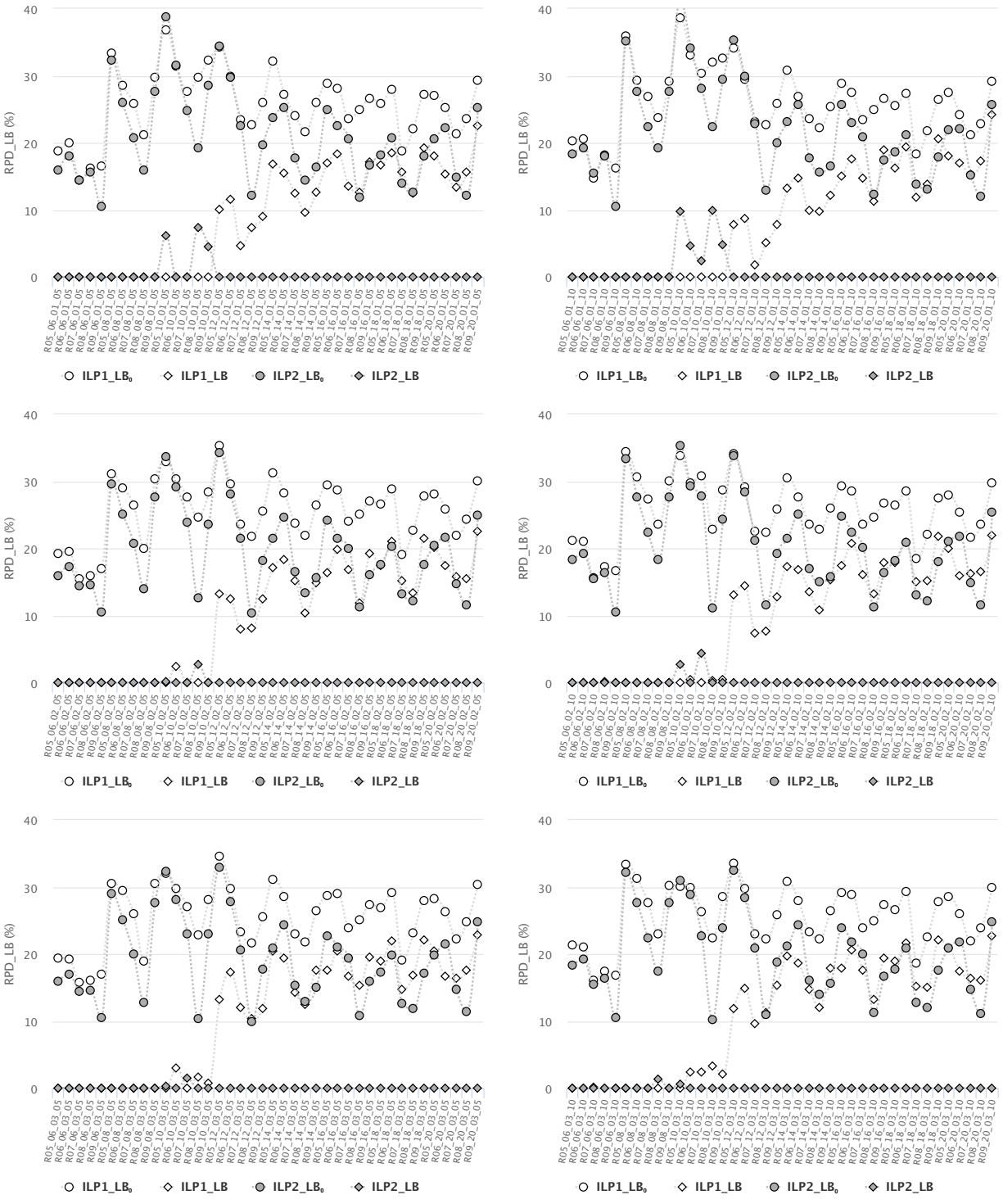

Figure 10: Relative Percentage Difference (RPD) values of the lower bounds of the mathematical models.

smaller ones for which both models have presented difficulties. However, also in this case model ILP2 had a better performance in terms of lower bounds achieved.

We conclude our analysis that confronts models ILP1 and ILP2 by examining the performance of their upper bounds (`ILP1_UB` and `ILP2_UB`, respectively), which have been obtained within the time limit established. In Table 3, we compare these upper bounds by reporting for each instance a value ω, which is computed as `(ILP1_UB - ILP2_UB) / min(ILP1_UB,`

Table 3: Comparison between upper bounds achieved by models ILP1 and ILP2.

n	L	h	R05	R06	R07	R08	R09
6	1	5	0.00	0.00	0.00	0.00	0.00
8			0.00	0.00	0.00	0.00	0.00
10			0.00	0.00	0.00	0.00	0.00
12			0.00	1.82	-2.03	-1.43	0.00
14			-7.66	0.00	-3.78	-8.19	0.00
16			-1.89	1.34	-7.12	1.14	0.22
18			-2.79	0.46	0.61	-4.21	-6.53
20			1.65	1.08	0.96	1.81	0.44
6	2		0.00	0.00	0.00	0.00	0.00
8			0.00	0.00	0.00	0.00	0.00
10			0.00	-0.26	0.00	-0.42	-0.15
12			-0.16	0.48	0.00	0.00	-0.54
14			-0.58	-7.19	-8.24	1.24	1.10
16			0.00	3.93	-6.86	-0.10	-0.87
18			-8.61	**0.00**	4.85	-1.93	**0.00**
20			2.43	0.18	2.76	3.36	0.44
6	3		0.00	0.00	0.00	0.00	0.00
8			0.00	0.00	0.00	0.00	0.00
10			0.00	-0.13	0.00	0.00	0.00
12			-7.16	2.22	0.00	0.50	-1.92
14			-0.12	0.00	-7.14	0.21	1.10
16			-0.11	1.34	0.00	4.20	0.11
18			1.72	**0.00**	7.01	0.09	0.10
20			4.31	3.07	3.40	**0.00**	0.44
6	1	10	0.00	0.00	0.00	0.00	**0.00**
8			0.00	0.00	0.00	0.00	**0.00**
10			0.00	-1.09	0.00	-0.53	**0.00**
12			0.00	-4.00	-3.83	-0.68	**0.00**
14			-0.68	**0.00**	-2.64	-0.10	**0.00**
16			-0.20	0.86	-2.93	-0.09	**0.00**
18			-0.63	0.00	0.00	-2.56	**0.00**
20			0.60	**0.00**	-0.95	-2.48	**0.00**
6	2		0.00	0.00	0.00	0.00	**0.00**
8			0.00	0.00	0.00	0.00	**0.00**
10			-0.82	0.00	0.00	0.67	**0.00**
12			-2.73	-1.73	-3.52	-0.71	**0.00**
14			-2.41	**0.00**	-2.03	-0.20	**0.00**
16			2.45	0.86	-1.82	0.00	**0.00**
18			0.89	0.55	-0.39	1.13	**0.00**
20			2.24	**0.00**	3.73	**0.00**	**0.00**
6	3		0.00	0.00	0.00	0.00	**0.00**
8			0.00	0.00	0.00	0.00	**0.00**
10			-3.15	0.00	-1.35	-1.47	**0.00**
12			-3.60	-1.18	0.75	-0.72	**0.00**
14			-1.94	**0.00**	-2.97	0.00	**0.00**
16			-1.03	**0.00**	1.50	1.23	**0.00**
18			2.33	**0.00**	2.06	1.04	**0.00**
20			6.08	**0.00**	**0.00**	**0.00**	**0.00**

ILP2_UB) × 100%. Thus, each instance with a negative value ω (highlighted in silver color) indicates that model ILP1 has found a solution $|\omega|\%$ better than the one found by model ILP2; while positive values (highlighted in gray color) indicate the opposite behavior.

From the results shown in Table 3, we observe that both models have obtained solutions with same the quality ($\omega = 0.00\%$) for several instances. This fact has occurred especially for smaller instances since both models have found optimal solutions. Moreover, some instances (ω values equal to 0.00% and highlighted in bold) that have not been solved have obtained equivalent solutions because the models have not even been able to improve their initial incumbent solution, which we recall is the optimal solution obtained when reloading operations are not allowed. Note that this fact has befallen on all instances that involve area R09 and relocation cost 10, indicating that on these instances the customers are located in such a way that no relocation is attractive or that both models have difficulty finding a way to accomplish them.

We also observe in Table 3 that both mathematical models are similar concerning the number of better solutions found. Moreover, the difference between their solutions is not significative for most instances. On average, this difference is only equal to 0.268%. Therefore, we can consider that both models are equally efficient regarding their upper bounds. However, according the results shown in Figure 10, model ILP2 has reached tighter lower bounds for almost all instances. These results are plausible since the model ILP2 was able to explore more nodes in the enumerated tree search (see Appendix B in the electronic Supplementary material for further detail), which can lead to tighter lower bounds. Moreover, the mathematical solver Gurobi performs general heuristic algorithms, general cutting planes, and branching rules in order to solve the models. In this way, a mathematical model can reach similar solutions to another model, despite having worse lower bounds.

4.4. ILPs vs. BRKGA

We focus now on the computational analysis of the proposed heuristic algorithm. For this purpose, we compare the average and best results reached in 10 independent runs of the BRKGA with upper bounds found by the mathematical models.

In Table 4, we report a summary of the results obtained by our models and heuristic approach. Each row of this table informs the average of results for each type of instance, i.e., each row contains the average results of five instances. The first three columns of Table 1 describe the type of instances. The results of the two mathematical models, ILP1 and ILP2, are described in the columns UB, R, and $T(s)$, which inform the objective function value UB and the total number of reloading operations R that were performed in the solution found after $T(s)$ seconds of processing. The results obtained by the proposed BRKGA are presented in the last 5 columns of the table. Columns Avg, σ, $Best$, and $T(s)$ show the average solution value, the standard deviation, the best solution value, and the average processing time consumed by the 10 independent runs of the BRKGA, respectively. The last column R of the table reports the total number of reloading operations that were performed in the best solution obtained by the BRKGA.

In order to emphasize the solution quality of each solution approach, we highlight in bold the best results achieved for each type of instance.

Table 4: Comparison between solutions found by models ILP1 and ILP2, and BRKGA.

Instances			ILP1			ILP2			BRKGA				
n	L	h	UB	R	T(s)	UB	R	T(s)	Avg	σ	Best	R	T(s)
6	1	5	**498.4**	2.8	0.8	**498.4**	2.8	0.3	**498.4**	0.0	**498.4**	2.8	28.0
8			**602.0**	3.4	45.4	**602.0**	3.4	70.6	602.0	0.1	**602.0**	3.4	38.1
10			**668.6**	6.6	1437.4	**668.6**	6.6	2614.8	674.8	3.4	**668.6**	6.6	48.1
12			747.8	6.4	1h	749.8	9.6	1h	745.1	1.5	**741.6**	5.8	58.1
14			860.8	5.8	1h	894.4	2.0	1h	855.6	9.5	**841.8**	7.2	68.2
16			976.2	4.2	1h	986.8	1.4	1h	963.1	10.3	**949.8**	7.0	78.2
18			1044.4	6.4	1h	1070.0	1.2	1h	1039.6	8.5	**1028.8**	6.2	88.3
20			1153.4	1.4	1h	1139.8	1.4	1h	1114.6	6.1	**1101.2**	8.2	98.4
6	2		**496.2**	3.0	1.2	**496.2**	3.0	0.7	497.3	1.3	**496.2**	3.0	36.1
8			**593.2**	6.2	98.4	**593.2**	5.0	99.0	594.7	0.6	**593.2**	5.0	50.1
10			**656.8**	8.6	3017.7	658.0	8.4	3055.5	665.0	3.9	658.6	7.0	64.2
12			727.6	8.4	1h	727.8	7.4	1h	733.5	7.5	**726.2**	8.2	78.2
14			868.6	6.0	1h	891.2	2.0	1h	849.8	10.6	**835.8**	9.6	92.3
16			970.0	6.4	1h	976.0	3.4	1h	952.9	18.4	**930.0**	11.8	106.4
18			1060.2	3.8	1h	1072.6	1.4	1h	1036.8	17.7	**1004.6**	12.4	120.6
20			1161.8	1.6	1h	1140.8	2.6	1h	1114.3	12.5	**1096.0**	8.2	134.6
6	3		**495.8**	2.8	1.7	**495.8**	2.8	1.8	497.8	0.7	496.6	3.0	42.1
8			**589.2**	5.4	317.8	**589.2**	5.4	157.3	594.9	2.8	591.8	4.2	60.1
10			**650.6**	8.2	1h	650.8	8.8	1h	659.3	5.1	651.0	7.6	78.2
12			723.4	9.8	1h	731.2	11.0	1h	731.6	14.6	**714.2**	10.6	96.3
14			870.6	6.6	1h	879.4	4.0	1h	847.8	11.1	**834.8**	8.8	114.4
16			970.2	4.2	1h	959.0	7.0	1h	944.8	13.9	**925.0**	8.6	132.6
18			1084.4	1.0	1h	1066.6	2.4	1h	1035.8	17.3	**1010.6**	9.4	150.8
20			1162.8	0.4	1h	1137.6	3.2	1h	1107.6	15.1	**1085.4**	10.2	168.9
6	1	10	**506.8**	1.0	0.9	**506.8**	1.0	0.8	**506.8**	0.0	**506.8**	1.0	28.0
8			**617.8**	2.4	49.2	**617.8**	2.4	304.8	**617.8**	0.0	**617.8**	2.4	38.1
10			**692.4**	1.8	1841.3	695.0	3.8	1h	**692.4**	0.0	**692.4**	1.8	48.1
12			767.6	1.8	1h	780.6	1.0	1h	764.1	2.5	**763.0**	2.6	58.1
14			892.8	2.2	1h	898.6	1.0	1h	878.3	5.4	**871.8**	3.0	68.2
16			989.0	2.0	1h	993.2	1.0	1h	984.4	1.8	**981.4**	2.6	78.2
18			1070.8	2.0	1h	1078.0	1.2	1h	1064.5	3.3	**1059.6**	4.0	88.3
20			1153.0	2.0	1h	1159.6	0.4	1h	1139.7	4.3	**1134.6**	4.4	98.4
6	2		**504.8**	0.8	1.4	**504.8**	0.8	2.4	505.0	0.5	**504.8**	0.8	36.1
8			**613.2**	3.2	343.0	**613.2**	3.2	585.7	613.8	1.1	**613.2**	3.2	50.1
10			689.0	4.0	3435.7	689.0	5.2	1h	691.4	1.3	**688.6**	3.6	64.1
12			762.0	4.8	1h	774.6	3.8	1h	762.3	2.5	**759.0**	2.8	78.2
14			892.6	3.4	1h	900.6	0.6	1h	875.7	4.6	**870.6**	3.4	92.3
16			995.4	3.0	1h	992.2	1.0	1h	983.3	1.4	**982.0**	2.4	106.5
18			1082.4	1.2	1h	1077.4	1.4	1h	1068.0	3.6	**1059.6**	3.8	120.6
20			1164.8	0.0	1h	1151.2	1.4	1h	1142.2	5.1	**1138.0**	3.4	134.7
6	3		**504.8**	0.8	2.2	**504.8**	0.8	2.6	505.5	0.3	**504.8**	0.8	42.1
8			**609.6**	3.0	744.4	**609.6**	3.0	1085.0	615.8	2.7	612.8	2.8	60.1
10			686.0	5.2	1h	693.6	4.4	1h	691.8	4.2	**684.2**	4.4	78.2
12			759.4	4.4	1h	766.6	5.0	1h	760.2	2.7	**757.2**	2.6	96.3
14			895.2	2.4	1h	903.6	0.6	1h	887.6	7.8	**877.8**	3.0	114.4
16			995.6	0.6	1h	992.2	1.4	1h	983.9	1.4	**981.8**	2.6	132.6
18			1088.4	1.2	1h	1076.6	1.2	1h	1069.7	8.9	**1052.0**	5.0	150.7
20			1164.8	0.0	1h	1151.2	1.2	1h	1144.2	6.1	**1137.2**	3.2	168.9

It can be clearly noticed from Table 4 that the BRKGA was able to find better solutions than the mathematical models for almost all types of instances. The best performance of the BRKGA with respect to the mathematical models was obtained for the larger instances, especially for those involving lower relocation cost. This behavior is completely understand-

able because a lower relocation cost may allow a higher number of reloading operations (column R) in order to find good routes, which can generate better solutions. We can also note that, according to the standard deviation of the solution values obtained by BRKGA (column σ), our heuristic approach presented a good convergence among its independent runs.

Regarding the computational time spent by each solution approach, in general, the BRKGA has better efficiency than the mathematical models. Only for the smaller instances, the models complete their executions before the time limit of 1 hour, whereas the BRKGA has taken less than 3 minutes to execute any instance.

In order to better analyze the relative difference between the solutions found, we use again the RPD metric. In this analysis, we compare the average (`BRKGA_Avg`) and best upper bound (`BRKGA_Best`) reached in 10 independent runs of the BRKGA with the best upper bound (`ILPs_UB`) found by the mathematical models. Now, we name the RPD metric as `RPD_BKS` and calculate it as `|(X - BKS)| / BKS` $\times$ `100%`, where `X` is the specific value that we want to measure its percentage variation according to the `BKS`, which is the objective value of the best known solution, i.e., `BKS = min(ILPs_UB, BRKGA_Best)`.

In Figure 11, we report for each instance the `RPD_BKS` evaluations for `ILPs_UB`, `BRKGA_Avg` and `BRKGA_Best`. These evaluations are divided into six charts, wherein each one groups all instances with the same reloading depth and relocation cost. From these charts, we can reaffirm the efficiency of BRKGA over mathematical models for each individual instance.

Notice that the BRKGA found better solutions for almost all instances even considering the average results found in its independent runs. Moreover, for some instances, the best upper bound reached by models is more than 10% worse than the best solution found by the BRKGA.

5. Conclusions and open perspectives

We have addressed the Double Traveling Salesman Problem with Multiple Stacks and Partial Last-In-First-Out Loading Constraints (DTSPMSPL), a generalization of the Double Traveling Salesman Problem with Multiple Stacks (DTSPMS) that allows performing reloading operations on each stack within a given reloading depth. In this paper, we have approached a variant of the DTSPMSPL, named Double Traveling Salesman Problem with Partial Last-In-First-Out Loading Constraints (DTSPPL), where the vehicle has its loading compartment as a single stack. To mathematically model the DTSPPL, we have presented two Integer Linear Programming (ILP) formulations. Moreover, we have developed a heuristic algorithm based on the Biased Random-Key Genetic Algorithm (BRKGA) metaheuristic.

The performance of the ILP formulations and of the BRKGA was studied on a comprehensive set of instances built from the DTSPMS benchmark instances. Both ILP formulations were able to solve to proven optimality only the smaller instances within one hour of processing time. One of them had tighter lower bounds for almost all instances, whereas the other could optimally solve more instances. The BRKGA has outperformed both ILP formulations in terms of better solution values achieved and smaller computating effort required, especially for the larger instances.

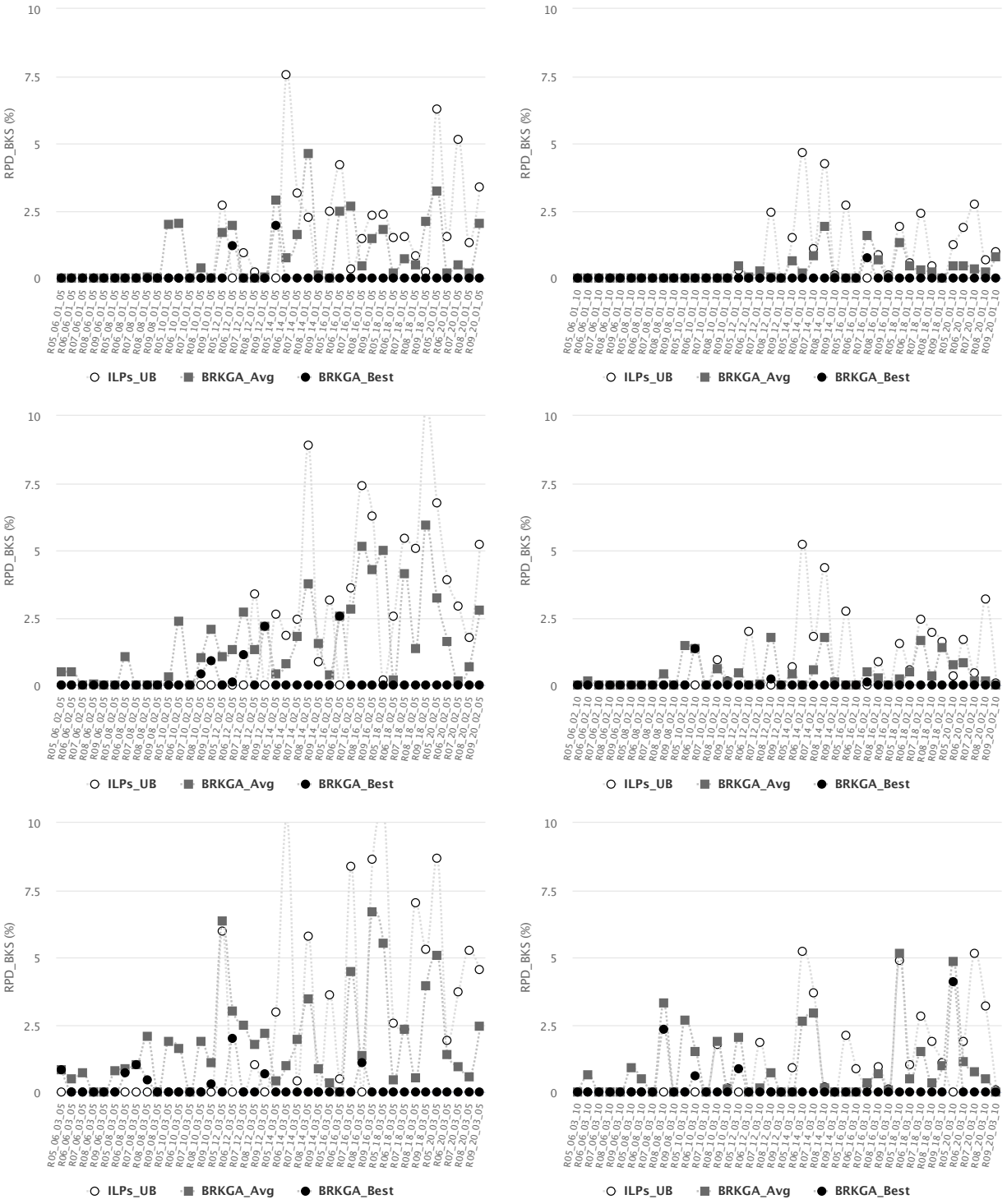

Figure 11: Relative Percentage Difference (RPD) values of the best solution found by each approach.

There are many possibilities for extending this work. Maybe the most relevant one would

be to model and solve the DTSPMSPL, the version of the DTSPPL where the loading compartment of the vehicle is divided into multiple stacks instead of a single one. Because of the increased computational complexity, we expect metaheuristic techniques to be the best option to find good solutions for this problem generalization. It would also be interesting to study the impact of forcing the LIFO policy to be fully respected, but allowing at the same time multiple visits to the customers. In such type of problems, iterative aggregate/disaggregate formulations, as in (Bruck and Iori, 2017), proved to be quite effective. Finally, another interesting extension of the DTSPPL/DTSPMSPL would be to consider a fleet of vehicles to serve the customers. More vehicles would cause an increase in the flexibility of the loading/unloading operations, which could lead to a reduction in operating costs.

Acknowledgments

The authors thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), and Universidade Federal de Ouro Preto (UFOP) for supporting this research.

References

M. A. Alba Martínez, J.-F. Cordeau, M. Dell'Amico, and M. Iori. A branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *INFORMS Journal on Computing*, 25(1):41–55, 2013.

D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2006.

M. Barbato, R. Grappe, M. Lacroix, and R. W. Calvo. Polyhedral results and a branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *Discrete Optimization*, 21:25–41, 2016.

M. Battarra, J.-F. Cordeau, and M. Iori. Pickup and delivery problems for goods transportation. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, MOS-SIAM Series on Optimization, pages 161–192. SIAM, 2nd edition, 2014.

M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

B. Bruck and M. Iori. Non-elementary formulations for single vehicle routing problems with pickups and deliveries. *Operations Research*, 65:1597–1614, 2017.

F. Carrabs, R. Cerulli, and J.-F. Cordeau. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. *INFOR: Information Systems and Operational Research*, 45(4):223, 2007a.

F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007b.

F. Carrabs, R. Cerulli, and M. G. Speranza. A branch-and-bound algorithm for the double TSP with two stacks. *Networks*, 61(1):58–75, 2010.

M. Casazza, A. Ceselli, and M. Nunkesser. Efficient algorithms for the double traveling salesman problem with multiple stacks. *Computers & Operations Research*, 39(5):1044–1053, 2012.

J. B. C. Chagas, U. E. F. Silveira, M. P. L. Benedito, and A. G. Santos. Simulated annealing metaheuristic for the double vehicle routing problem with multiple stacks. In *19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1311–1316, Rio de Janeiro, Brasil, 2016. IEEE.

J. B. C. Chagas, U. E. F. Silveira, A. G. Santos, and M. J. F. Souza. A variable neighborhood search heuristic algorithm for the double vehicle routing problem with multiple stacks. *International Transactions in Operational Research*, 2019. . Available at `https://doi.org/10.1111/itor.12623`.

J.-F. Cordeau, M. Iori, G. Laporte, and J. J. Salazar González. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks*, 55(1):46–59, 2010.

J.-F. Côté, C. Archetti, M. G. Speranza, M. Gendreau, and J.-Y. Potvin. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(4):212–226, 2009.

K. Doerner and J.-J. Salazar-González. Pickup and delivery routing problems for people transportation. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, MOS-SIAM Series on Optimization, pages 193–212. SIAM, 2nd edition, 2014.

Á. Felipe, M. T. Ortuño, and G. Tirado. The double traveling salesman problem with multiple stacks: A variable neighborhood search approach. *Computers & Operations Research*, 36(11):2983–2993, 2009.

J. F. Gonçalves and M. G. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

J. F. Gonçalves and M. G. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39(2):179–190, 2012.

J. F. Gonçalves and M. G. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510, 2013.

J. F. Gonçalves and M. G. Resende. A biased random-key genetic algorithm for the unequal area facility layout problem. *European Journal of Operational Research*, 246(1):86–107, 2015.

G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

M. Iori and S. Martello. Routing problems with loading constraints. *Top*, 18(1):4–27, 2010.

M. Iori and J. Riera-Ledesma. Exact algorithms for the double vehicle routing problem with multiple stacks. *Computers & Operations Research*, 63:83–101, 2015.

O. Kherbash and M. L. Mocan. A review of logistics and transport sector as a factor of globalization. *Procedia Economics and Finance*, 27:42–47, 2015.

S. P. Ladany and A. Mehrez. Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301–306, 1984.

Y. Li, A. Lim, W.-C. Oon, H. Qin, and D. Tu. The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. *European Journal of Operational Research*, 212(3):482–496, 2011.

M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

R. M. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. An exact method for the double TSP with multiple stacks. *International Transactions in Operational Research*, 17(5):637–652, 2010.

M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

A. H. Pereira and S. Urrutia. Formulations and algorithms for the pickup and delivery traveling salesman problem with multiple stacks. *Computers & Operations Research*, 93:1–14, 2018.

H. L. Petersen. *Decision Support for Planning of Multimodal Transportation with Multiple Objectives*. PhD thesis, Technical University of Denmark (DTU), 2009.

H. L. Petersen and O. B. Madsen. The double travelling salesman problem with multiple stacks–formulation and heuristic solution approaches. *European Journal of Operational Research*, 198(1):139–147, 2009.

H. L. Petersen, C. Archetti, and M. G. Speranza. Exact solutions to the double travelling salesman problem with multiple stacks. *Networks*, 56(4):229–243, 2010.

A. H. Sampaio and S. Urrutia. New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research*, 24: 77–98, 2016.

A. G. Santos and J. B. C. Chagas. The thief orienteering problem: Formulation and heuristic approaches. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1191–1199, Rio de Janeiro, Brasil, 2018. IEEE.

U. E. F. Silveira, M. P. L. Benedito, and A. G. Santos. Heuristic approaches to double vehicle routing problem

with multiple stacks. In *15th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 231–236, Marrakesh, Marocco, 2015. IEEE.

R. F. Toso and M. G. Resende. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93, 2015.

M. Veenstra, K. J. Roodbergen, I. F. Vis, and L. C. Coelho. The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research*, 257(1):118–132, 2017.