A metaheuristic for the double traveling salesman problem with partial last-in-first-out loading constraints

Article *in* International Transactions in Operational Research · November 2023

DOI: 10.1111/itor.13189

3 authors, including:

Gustavo Gatica
Pontificia Universidad Javeriana - Cali
75 PUBLICATIONS **261** CITATIONS
SEE PROFILE

Carlos Contreras Bolton
University of Concepción
29 PUBLICATIONS **261** CITATIONS
SEE PROFILE

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2022) 1–26
DOI: 10.1111/itor.13189

A metaheuristic for the double traveling salesman problem with partial last-in-first-out loading constraints

Blas Mardones[a], Gustavo Gatica[b] and Carlos Contreras-Bolton[a,*]

[a] *Departamento de Ingeniería Industrial, Universidad de Concepción, Edmundo Larenas 219, Concepción 4070409, Chile*
[b] *Facultad de Ingeniería - Centro Sustentabilidad, Universidad Andres Bello, Antonio Varas 880, Santiago 7500971, Chile*
E-mail: bmardonesz@udec.cl [Mardones]; ggatica@unab.cl [Gatica]; carlos.contreras.b@udec.cl [Contreras-Bolton]

Received 22 October 2021; received in revised form 27 April 2022; accepted 17 July 2022

Abstract

This paper addresses the double traveling salesman problem with partial last-in-first-out (LIFO) loading constraints. A vehicle picks up items in the pickup area and loads them into its container, a horizontal stack. Once all the pickup operations are done, the vehicle can deliver the items to the delivery area because pickup-and-delivery areas are geographically separated. Additionally, the loading and unloading operations also follow a partial LIFO policy. The aim is to find a Hamiltonian cycle that satisfies the loading and unloading policy described above in order to minimize the distance traveled by the vehicle in both areas and the cost of the rearrangements that are performed across the routes. The state-of-the-art algorithm finds good solutions to the existing instances. However, finding good-quality solutions in large instances requires longer computing times. Therefore, we propose an iterated local search that considers two additional components to the classical components, the intensification and restart procedure. The first helps in a more thorough examination of promising neighborhoods, while the second helps in the complete diversity of the solutions in the hopes of escaping the local optima. Moreover, a new unique decoding approach for the solution representation is proposed to effectively generate feasible solutions. The proposed algorithm's performance is evaluated on an existing set of instances and in two new additional sets of larger instances that are proposed. The computational results obtained on the three test sets show that the proposed algorithm outperforms the state-of-the-art algorithm in terms of computing time required to find good-quality solutions.

Keywords: pickup and delivery; loading constraints; partial reloading; metaheuristic; iterated local search

1. Introduction

Transportation logistics is currently one of the most important fields in operations research due to its role in the world and regional economy (Magableh, 2021). According to the World Bank (2020), logistics has the potential to boost economies, help tackle climate change mitigation, and

*Corresponding author.

connect people to essential services such as health care or education. As a result of its influential role, the global freight transport market is expected to grow by 5% in the period from 2020 to 2025 as estimated (World Trade Organization, 2020).

Transportation and logistics problems have been investigated since the beginning of the operations research, one of which being the vehicle routing problem (VRP) (Dantzig and Ramser, 1959). Since then, the VRP has established itself as one of the most researched problems due to its great relevance in real-world applications and its intrinsic difficulty to be solved (Toth and Vigo, 2014).

VRP is a generalization of another well-known problem, the traveling salesman problem (TSP). TSP consists of a set of customers, for which it is required to find a minimum route that visits each customers just once, and starts and ends at the same customer. In addition, TSP is known for its computational difficulties in solving it as it belongs to the NP-complete class (Karp, 1972). However, although it has been widely studied, there is still no algorithm that solves it in polynomial time, and it remains an open problem.

There are several TSP extensions and variants, some of which are even more difficult to solve than the TSP. Some examples include the flying sidekick TSP (Dell'Amico et al., 2022), the TSP with time-dependent service times (Cacchiani et al., 2020), the clustered TSP with a prespecified order on the clusters (Hà et al., 2022), the split-demand one-commodity pickup-and-delivery TSP (Hernández-Pérez and Salazar-González, 2022), the prize-collecting TSP (Pantuza Jr. and de Souza, 2022), among others. Generally, these extensions and variants of the TSP cannot be solved efficiently through its mathematical models.

One interesting variant that arises in transport logistics is when it is necessary to optimize the routing of vehicles and the loading of goods onto them is needed. Since it is difficult to handle in reality in an integrated approach, this problem has generally been tackled separately by a routing problem and then a loading problem (Iori and Martello, 2010). Recently, these two problems have started to integrate and it has piqued the interest of many researchers in its applications and its combinatorial complexity. In fact, recently, Chagas et al. (2022) proposed a particular case of the pickup-and-delivery problem with partial last-in-first-out (LIFO) loading constraints and with a single vehicle. This new problem proposed by Chagas et al. (2022) consists of all pickup operations that must perform before any delivery operation since the pickup-and-delivery areas are geographically separated. In the pickup area, the vehicle collects items and loads them to its container. After completing all of the pickup procedures, the vehicle proceeds to the delivery area to begin distributing the items in the delivery area. The partial LIFO policy must be followed throughout the loading and unloading operations. It is a partial LIFO policy because a version of the LIFO may be violated within a given reloading depth. Thus, the authors called this problem the double TSP with partial LIFO (DTSPPL) loading constraints.

To the best of our knowledge, the DTSPPL proposed by Chagas et al. (2022) is the first work that considers the loading compartment as a single horizontal rather than numerous stacks as is used commonly. To solve the DTSPPL, this study proposes an algorithm based on an iterated local search (ILS). Our proposed algorithm contains two additional components to the initial solution procedure, perturbation strategies, local search strategies, and acceptance criteria, which are the components that normally have an ILS. The design and development of two new components, intensification and restart procedures, are thus the first of the contributions of this work. In the first component, a promising neighborhood (elite solutions) is examined more carefully to find better solutions after a given number of iterations. Meanwhile, the restart procedure helps in the complete

diversification of the solutions in the hopes of avoiding the local optimal solutions after a certain number of iterations. A new novel efficient decoding procedure for the solution representation is proposed as a second contribution, which consists of in 6-tuple that represents the solution and always provide feasible solutions. The third contribution consists of two new sets of instances that are substantially larger (25–100 customers) compared to the existing instances set (up to 20 customers). Finally, our fourth contribution is that the proposed algorithm outperforms the state-of-the-art algorithm in the three sets of instances as evidenced by computational results, which show that it is more efficient in finding better quality solutions and in shorter computing times.

The rest of this paper is laid out as follows. A literature review in Section 2 concentrates mainly on the most relevant works for the DTSPP, namely the pickup-and-delivery TSP with loading constraints. Section 3 formally defines the problem under consideration and presents an ILP formulation. Section 4 describes the proposed metaheuristic for resolving the problem. The computational experiments of the achieved results and comparison studies are presented and discussed in Section 5. Finally, Section 6 provides the concluding remarks and future works.

2. Literature review

The TSPs with loading constraints have attracted attention from researchers. In fact, Ladany and Mehrez (1984) proposed the first work that merged pickups, deliveries, and loading in the early 1980s. This problem is known as the pickup-and-delivery TSP with LIFO loading (PDTSPL). Thus, in the PDTSPL, loading and unloading operations must obey the LIFO policy since the only vehicle has a single access point, usually in the rear. PDTSPL proposed by Ladany and Mehrez (1984) consisted of a real-life problem in which a trucker must pick up similar-sized loads from a set of customers and deliver the collected items in the same region. In addition, the items are also loaded into the vehicle's loading compartment, which is a horizontal stack with only the items located on top are available for unloading and delivery. The authors propose an enumeration approach for a single vehicle to tackle a modest real-world problem.

The PDTSPL was revisited after a long time in two works by Carrabs et al. (2007a, 2007b). The authors first propose a branch-and-bound (B&B) for solving the PDTSPL, considering loading and unloading operations must follow the LIFO or first-in first-out policy. Their approach was evaluated with the instances of the literature, and it was found to be capable of solving up to 39 customers in less than 600 seconds. The second paper proposes a variable neighborhood search (VNS) by the same authors. They introduce local searches that integrate into a VNS, and it is validated on instances of sizes from 25 to 751 customers. The results show that the algorithm can efficiently solve the proposed instances.

Cordeau et al. (2010) proposed three mathematical formulations for the PDTSPL and a branch-and-cut (B&C) algorithm that uses several families of proposed valid inequalities. The computational results show that instances with up to 25 customers can be solved. Then, Li et al. (2011) developed a new VNS heuristic for the PDTSPL with tree-based search operators. The results show that their approach outperforms the state-of-the-art algorithm in terms of solution quality and computing time for large instances. Currently, the last study that addresses the PDTSPL was proposed by Veenstra et al. (2017) who worked on a variant of the PDTSPL. This variant considers that rearrangement operations are allowed only at delivery locations. Therefore, handling

operations with associated penalty costs are used without a depth limit for reloading. Additionally, they propose an ILP formulation and a large neighborhood search (LNS) that solve instances for the PDTSPL and their proposed variant with sizes ranging from 11 to 751 customers. LNS finds new best-known solutions (BKSs) for both problems.

The pickup-and-delivery TSP with multiple stacks (PDTSPMS) (Côté et al., 2012) and the double TSP with multiple stacks (DTSPMS) (Petersen and Madsen, 2009) were proposed in the late 2000s. The PDTSPMS is a variant of the PDTSPL, where the vehicle loading compartment is divided into multiple horizontal stacks, each stack is independent, and their loading and unloading operations must follow the LIFO policy. The DTSPMS is a particular case of the PDTSPMS in which the pickup-and-delivery operations are completely separated due to the geographical distance between the pickup-and-delivery areas. Therefore, the DTSPPL is a particular case of the DTSPMS, where the vehicle loading compartment has a single horizontal stack, and its LIFO policy may be violated within a given reloading depth.

Meanwhile, the PDTSPMS has received less attention than the PDTSPL, despite it was first proposed by Côté et al. (2012) who presented a B&C algorithm for its resolution. The algorithm is measured on two types of instances, randomly generated and classical instances, and solved optimally with up to 28 customers. Following that, Sampaio and Urrutia (2017) propose a new ILP formulation and a B&C algorithm. Computational results show that their algorithm is competitive with the best algorithm in the literature, on instances of size between 11 and 21 customers, with a time limit of one hour. Subsequently, Pereira and Urrutia (2018) addressed new ILP formulations and a B&C algorithm. They proved new optimal solutions for several benchmark instances, and optimal solutions previously known in the literature are solved more efficiently. Finally, Pereira et al. (2022) proposes a procedure of valid inequalities and exact algorithms to solve the PDTSPMS. They also propose a new ILP formulation and new valid inequalities as the foundation for a B&C algorithm. The best performing exact algorithms in the literature are compared to the new B&C. The proposed algorithm outperforms all state-of-the-art algorithms in size instances with up to 21 customers in a maximum computing time of 18,000 seconds.

Since Petersen (2009) introduced the DTSPMS, it has got significant attention from the academic community. The first paper develops a mathematical model and four metaheuristics: ILS, tabu search (TS), simulated annealing (SA), and LNS, where the LNS generates the best results. Felipe et al. (2009a) proposed four neighborhood structures applied to a VNS. VNS was tested with instances of 33, 66, and 132 customers, finding good solutions in short computing times. Then, Felipe et al. (2009b) extended their previous work on another VNS and SA, where the SA showed the best performance. Next, Petersen et al. (2010) proposed different ILP formulations based on the original formulation proposed by Petersen and Madsen (2009) and also included a B&C algorithm that solves instances with up to 25 customers. Lusby et al. (2010) presented an exact method based on matching k-best tours for each region separately and was able to solve to optimality up to 18 customers. Lusby and Larsen (2011) then refined the above exact method through an additional preprocessing technique that uses the longest common subsequence between the respective areas. The enhanced approach was able to solve to optimality up to 24 customers.

In the years that followed, more research on exact approaches was proposed and the first work that studied the theoretical properties of the DTSPMS. Casazza et al. (2012) proposed this theoretical work by analyzing the structure of solutions and proposed polynomial algorithms for subcases of the problem. Keeping with the exact approaches, in Alba Martínez et al. (2013), a B&C algorithm

was proposed, which introduces valid inequalities and which can be separated efficiently. The new B&C approach can solve to optimality up to 28 customers. Carrabs et al. (2013) presented a B&B algorithm for the DTSPMS with only two stacks and solved instances for up to 29 customers to optimality. Barbato et al. (2016) provided a polyhedral study of the problem and an ILP formulation for the DTSPMS with only two stacks with an infinite capacity each. Furthermore, they devised a B&C algorithm that was able to find optimal solutions for instances with up to 18 customers.

Hvattum et al. (2020) analyze the effect on transportation costs in the DTSPMS, when considering the use of a container loaded from the side, instead of a container loaded from the rear. The experiment is performed with a variant of the VNS, variable neighborhood descent, and shows savings in transportation costs of up to 20%. Again the DTSPMS is studied in a theoretical way, Alfandari and Toulouse (2021) present several approximation algorithms and provide interesting approximation results, in both cases of standard and differential approximation. Recently, Ruan et al. (2021) study a particular case of DTSPMS, considering three-dimensional container loading constraints. They propose an ILP model and implement a standard genetic algorithm (GA) and an improved GA. The ILP is capable of solving instances with up to 24 customers. In contrast, standard GA and improved GA can solve efficiently up to 100 customers. Furthermore, the improved GA has the best performance than the standard GA. Finally, Chagas et al. (2022) state a variant of the DTSPMS, being a particular case that they called DTSPPL. They propose two ILP formulations, which optimally solve instances with up to eight customers within a time limit of one hour. Additionally, they propose a biased random-key GA (BRKGA), capable of solving most of the 1080 instances proposed, but with a one-hour time limit.

A summary of the main studies of the literature that relates to the considered problem and that has been reviewed in this section is presented in Table 1 in a chronological order. The first column denotes the authors and the second column indicates the type of problem addressed. The third column denotes if the work proposes a new mathematical model. While the fourth and fifth columns correspond to the type of resolution approach and the method used to solve the problem addressed, respectively. Note that ILP^c or ILP^g indicates that the ILP was solved using the CPLEX or Gurobi solver, respectively. In addition, when the type of problem has v as a superscript means that the paper considers a variant of the original problem. To summarize, there are 24 works reviewed, with 14 of them focusing on the DTSPMS, PDTSPL in 6 works, and PDTSPMS in 4 cases. Furthermore, exact approaches have been used 13 times and the heuristic approach has been used 11 times to address these works. While the most widely used exact and heuristic approaches are the B&C and the VNS, respectively.

3. Problem definition

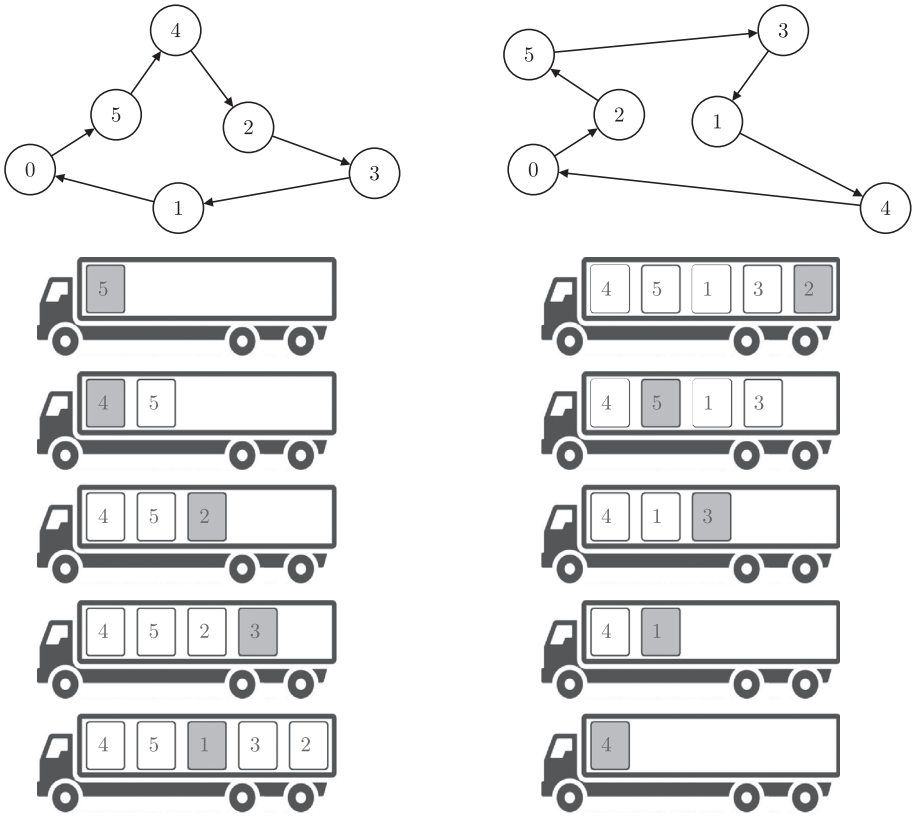
The DTSPPL can be formally defined by six sets, $C = \{1, 2, \ldots, n\}$ as the set of n customer requests, $Q = \{p, d\}$ as the set of areas with p and d represent the pickup-and-delivery areas, two sets $V_c^q = \{1^q, 2^q, \ldots, n^q\}$ correspond to the pickup-and-delivery locations, where $V^q = \{0^q\} \cup V_c^q$ represent the vertices in each area, with 0^q as the depots for each area with $q \in Q$. The last two sets, $A^q = \{(i^q, j^q) \forall i^q \in V^q, \forall j^q \in V^q | i^q \neq j^q\}$ are the sets of arcs in the pickup-and-delivery areas. Each customer's item $i \in C$ is required to be transported from pickup location i^p to delivery location i^d. Thus, it can be defined in two complete directed graphs, $G^q = (V^q, A^q)$, $q \in Q$. Every (i^q, j^q), $q \in Q$

Table 1
Summary of the studies reviewed

Author	Problem	Model	Approach	Method
Ladany and Mehrez (1984)	PDTSPL	–	Exact	Enumeration
Carrabs et al. (2007a)	PDTSPL	–	Exact	B&B
Carrabs et al. (2007b)	PDTSPL	–	Heuristic	VNS
Petersen and Madsen (2009)	DTSPMS	–	Heuristic	ILS, TS, SA, and LNS
Felipe et al. (2009a)	DTSPMS	–	Heuristic	VNS
Felipe et al. (2009b)	DTSPMS	–	Heuristic	SA and VNS
Cordeau et al. (2010)	PDTSPL	–	Exact	B&B
Lusby et al. (2010)	DTSPMS	–	Exact	k-best approach
Petersen et al. (2010)	DTSPMS	ILP^c	Exact	B&C
Li et al. (2011)	PDTSPL	–	Heuristic	VNS
Lusby and Larsen (2011)	DTSPMS	–	Exact	k-best approach
Casazza et al. (2012)	DTSPMS	–	Heuristic	Efficient heuristic algorithms
Côté et al. (2012)	PDTSPMS	–	Exact	B&C
Alba Martínez et al. (2013)	DTSPMS	–	Exact	B&C
Carrabs et al. (2013)	DTSPMS^v	–	Exact	B&B
Barbato et al. (2016)	DTSPMS^v	ILP^c	Exact	B&C
Veenstra et al. (2017)	PDTSPL^v	ILP^c	Heuristic	LNS
Sampaio and Urrutia (2017)	PDTSPMS	ILP^c	Exact	B&C
Pereira and Urrutia (2018)	PDTSPMS	ILP^c	Exact	B&C
Hvattum et al. (2020)	DTSPMS	–	Heuristic	VNS
Alfandari and Toulouse (2021)	DTSPMS	–	Heuristic	Approximation algorithm
Ruan et al. (2021)	DTSPMS^v	ILP^g	Heuristic	GA
Pereira et al. (2022)	PDTSPMS	ILP^c	Exact	B&C
Chagas et al. (2022)	DTSPMS^v	ILP^g	Heuristic	BRKGA

has associated a c_{ij}^q correspond to the travel distances. Additionally, for convenience of notation, the sets V_c^q is referred to as the set of requests C, i is used to denote i^q, and (i, j) to denote (i^q, j^q), $q \in Q$. The purpose of the problem is to find a Hamiltonian cycle for each graph G^q, $q \in Q$, where the loading and unloading operations must adhere to the partial LIFO policy, which is defined by a maximum reloading depth L and a cost h for each item rearranged.

Figure 1 shows an example of a feasible solution for the DTSPPL, in which a vehicle must make the pickup and delivery of five customers in two geographically separated areas, considering a reloading depth of 2. The vehicle starts its pickup route from the pickup depot (node 0). First, it travels to customer 5 and stores the customer's item in the container. It then proceeds to the pickup position of customer 4. At this point, a rearrangement is performed into the stack, which consists of temporarily removing the item of customer 5. Then, the item of customer 4 is placed in the stack (where the item of customer 5 previously used to be), and then the item of customer 5 is replaced in the stack. Subsequently, the vehicle picks up the items from customers 2 and 3, placing them into the stack, respectively. Next, the vehicle travels to customer 1, and a rearrangement is performed, where the items of customers 2 and 3, respectively, are removed. Then, item 1 is placed, and the items of customers 2 and 3 are relocated in a new order as 3 and 2. The vehicle then returns to the pickup depot, before being transferred to the depot located in the delivery area to perform the delivery operations. The distance between the pickup-and-delivery depot is not



a) Pickup area b) Delivery area

Fig. 1. Example of an instance of five customers and a reloading depth equal to 2.

considered while calculating the cost of the final trip. In the second part, the vehicle leaves from the delivery depot in the direction of customer 2, to deliver the item without any rearrangement. Then, it travels to customer 5, but to deliver the item, a rearrangement must be made, for which the items of customers 1 and 3 are removed. Once delivered item 5, the items of customers 1 and 3 are replaced back into the stack in the same order. Delivery operations continue with the items of customers 3, 1, and 4, respectively, without any rearrangement. Finally, the vehicle returns to the delivery depot. Two rearrangements in the pickup area (Fig. 1a) and one rearrangement in the delivery area (Fig. 1b), for a total of three rearrangements in this example.

The DTSPPL can be formulated using an ILP model that was introduced by Chagas et al. (2022). The formulation uses the following binary decision variables: $x_{ij}^{q}, \forall q \in Q, (i, j) \in A^{q}$ represents the route of the vehicle in each area, taking value 1 if the vehicle traverses arc (i, j) in area q, and value 0 otherwise. $y_{j\ell}^{kp}, \forall k \in [1, n], j \in C, \ell \in [1, k]$ represents the loading operations, takes value 1 if the item refers to customer request j that is stored at position ℓ in the kth container configuration in the pickup area, and 0 otherwise. While $y_{j\ell}^{kp}, \forall k \in [1, n], j \in C, \ell \in [1, n-k+1]$ represents the

Table 2
Nomenclature of the ILP model

Sets	
C	Set of customer requests $\{1, 2, \ldots, n\}$
Q	Set of pickup-and-delivery areas $\{p, d\}$
V^q	Sets of vertices in each area ($q \in Q$)
A^q	Sets of arcs in each area ($q \in Q$)
Parameters	
L	Maximum reloading depth
h	Rearrangement cost of an item
c_{ij}^q	Travel distances associated the arcs $(i, j) \in A^q$ in area $q \in Q$
Decision variables	
x_{ij}^q	Represents the route of the vehicle in each area ($\forall q \in Q, (i, j) \in A^q$)
$y_{j\ell}^{kq}$	Represents the loading and unloading operations ($\forall k \in [1, n], q \in Q, j \in C, \ell \in [1, k]$)
z^{kq}	Represents the number of rearrangements performed during the loading and unloading plans ($\forall k \in [1, n-1], q \in Q$)
u_j^q	Represents the position of a customer $j \in [0, n]$ in the route of the area $q \in Q$

unloading operations, analogously to the previous variables but considering the delivery area. In addition, z^{kq}, $\forall k \in [1, n-1]$, $q \in Q$ is a nonnegative continuous decision variable that determines how many rearrangements are performed from the loading and unloading plans. Finally, an integer variable u_j^q, $\forall q \in Q$, $j \in [0, n]$ denotes the position of a customer j in the route of the area q. Note that the notation $[a, b]$ is referred the set $\{a, a+1, \ldots, b-1, b\}$, for any $a > b$, $[a, b]$ is an empty set. To ease presentation, we summarize the sets, the parameters, and the decision variables used for the ILP model and are listed them in Table 2. Therefore, the ILP model for the DTSPPL reads as follows:

$$\min \sum_{q \in Q} \sum_{(i,j) \in A^q} c_{ij}^q x_{ij}^q + h \sum_{q \in Q} \sum_{k \in [1, n-1]} z^{kq} \tag{1}$$

subject to

$$\sum_{j:(i,j) \in A^q} x_{ij}^q = 1, \qquad q \in Q, i \in V^q \tag{2}$$

$$\sum_{i:(i,j) \in A^q} x_{ij}^q = 1, \qquad q \in Q, j \in V^q \tag{3}$$

$$u_j^q \geq u_i^q + 1 - n(1 - x_{ij}^q), \qquad q \in Q, (i, j) \in A^q : j \neq 0 \tag{4}$$

$$u_0^q = 0, \qquad q \in Q \tag{5}$$

$$1 \geq \sum_{l \in [1,k]} y_{j\ell}^{kp} \geq \frac{k - u_j^p + 1}{k}, \qquad k \in [1, n],\, j \in C \tag{6}$$

$$1 \geq \sum_{l \in [1,n-k+1]} y_{j\ell}^{kd} \geq \frac{u_j^d - k + 1}{n - k + 1}, \qquad k \in [1, n],\, j \in C \tag{7}$$

$$\sum_{j \in C} y_{j\ell}^{kp} = 1, \qquad k \in [1, n],\, \ell \in [1, k] \tag{8}$$

$$\sum_{j \in C} y_{j\ell}^{kd} = 1, \qquad k \in [1, n],\, \ell \in [1, n - k + 1] \tag{9}$$

$$y_{j\ell}^{1d} = y_{j\ell}^{np}, \qquad \ell \in [1, n],\, j \in C \tag{10}$$

$$y_{je}^{kp} = y_{j\ell}^{np}, \qquad k \in [1, n-1],\, \ell \in [1, k - L],\, j \in C \tag{11}$$

$$y_{j\ell}^{kd} = y_{j\ell}^{1d}, \qquad k \in [2, n],\, \ell \in [1, n - k - L + 1],\, j \in C \tag{12}$$

$$z^{kp} \geq (y_{j\ell}^{kp} - y_{j\ell}^{k+1,p})(k - \ell + 1), \qquad k \in [1, n-1],\, \ell \in [1, k-1],\, j \in C \tag{13}$$

$$z^{kd} \geq (y_{j\ell}^{k+1,d} - y_{j\ell}^{kd})(n - k - \ell + 1), \qquad k \in [1, n-1],\, \ell \in [1, n-k],\, j \in C \tag{14}$$

$$x_{ij}^{q} \in \{0, 1\}, \qquad q \in Q,\, (i, j) \in A^q \tag{15}$$

$$y_{j\ell}^{kp} \in \{0, 1\}, \qquad k \in [1, n],\, \ell \in [1, k],\, j \in C \tag{16}$$

$$y_{j\ell}^{kd} \in \{0, 1\}, \qquad k \in [1, n],\, \ell \in [1, n - k + 1],\, j \in C \tag{17}$$

$$z^{kq} \in \mathbb{Z}^+, \qquad q \in Q, k \in [1, n-1] \tag{18}$$

$$u_j^q \in \{a \in \mathbb{Z}^+ : a \leq n\}, \qquad q \in Q,\, j \in C. \tag{19}$$

Objective function (1) minimizes the vehicle's distance traveled, and cost of all rearrangements. Constraints (2) and (3) ensure that each pickup-and-delivery location is visited exactly once. Constraints (4) and (5) relate to the subtour elimination constraints. Constraints (6) and (7) guarantee the correct assignment of items to the vehicle loading compartment throughout the pickup-and-delivery routes. Constraints (8) and (9) enforce that only one item must occupy each container position in each of its configurations. Constraints (10) ensure that there is no rearrangement of items between the transfer from the pickup depot to the delivery depot. Constraints (11) and (12) state that items must remain in their previous container positions in order not to violate the partial LIFO policy. Constraints (13) and (14) allow calculating the number of rearrangement in the pickup-and-delivery area, respectively. Finally, constraints (15)–(19) define the domain of the decision variables.

4. Proposed metaheuristic

This section presents the proposed algorithm for solving the DTSPPL, which is based on the metaheuristic algorithm ILS (Lourenço et al., 2003). ILS is an iterative procedure that applies repeated local searches. The perturbations are then called to avoid the local optimal solutions. Therefore, this section considers the description of our algorithm, the representation, generation of the initial solution, perturbations, local searches, and two additional components to the classical components, the intensification and restart procedures.

4.1. Algorithm description

Our proposed ILS has six components: initial solution procedure, perturbation strategy, local search strategy, acceptance criterion, intensification, and restart procedure. The pseudocode is presented in Algorithm 1 and is executed till the time limit is reached. Furthermore, an elite solution set is used to preserve the best solutions throughout algorithm execution.

The initial solution (lines 1–4) is generated in the beginning, which corresponds to the current solution (s), the best solution (s^*), the first elite solution (e_0), and initialize the counter of iterations (ι). Subsequently, the main loop of the algorithm begins, and it finishes at the reaching of a given time limit. Within the loop, the perturbation strategy is applied, which calls to different perturbations according to the occurrence probabilities α_i, $i \in \{1, \ldots, 5\}$ (line 6). Then, it continues to the local search strategy, which is divided into procedures that improve the route and others that improve rearrangement (lines 7–14). The route improvement procedures apply with a probability θ, 2-opt or 3-opt algorithm (lines 7–10). The rearrangement improvement procedures apply with a probability β, two types of local search for the rearrangement (lines 11–14). The intensification procedure is performed every ρ iterations and considers a set of elite solutions (e), to which an exchange improvement (lines 15–23) is applied. Then, in the acceptance criterion, we evaluate if s is better than s^* in order to update s^*. Otherwise, according to the probability τ, s is updated by a random solution taken from the elite solutions. Finally, given the probability λ in lines 24–29, a restart procedure is applied to the current solution using the nearest neighbor heuristic or the initial solution according to the probability μ.

4.2. Solution representation

The scheme used to represent the problem solution is a 6-tuple, $s = (s.r^p, s.r^d, s.o^p, s.o^d, s.w^p, s.w^d)$, where $s.r^p$ and $s.r^d$ represent two lists corresponding to the pickup-and-delivery routes, respectively. The values of these lists correspond to the customers and have size n. The $s.o^p$ and $s.o^d$ represent two lists corresponding to the pickup rearrangement and the delivery rearrangement, respectively, and both lists have size n. Additionally, the lists can take values according to the area. Thus, in the pickup area, each item $i \in \{0, 1, \ldots, n-1\}$ can take values between 0 and $\min(i, L)$, while values between 0 and $\min(n-i, L)$ are used for the delivery area. Finally, $s.w^p$ and $s.w^d$ represent two lists of size n and correspond to the pickup-and-delivery weight, respectively. In the case that an item is

Algorithm 1. ILS

```
1  e = {∅};
2  s, s* = initial-solution();
3  e = e ∪ {s};
4  ι = 0;
5  repeat
6      perturbation(random(0,1), α);
7      if random(0,1) < θ then
8          s = 2-opt(s);
9      else
10         s = 3-opt(s);
11     if random(0,1) < β then
12         s = full-rearrangement-improvement(s);
13     else
14         s = partial-rearrangement-improvement(s);
15     if ι%ρ = 0 then
16         for e' ∈ e do
17             e' = exchange-improvement(s);
18             e = e ∪ e';
19     if f(s) < f(s*) then
20         s* = s;
21         e = e ∪ {s};
22     else if f(s) > f(s*) × (1 + τ) then
23         s = e_t,   t ∈ random(1,|e|);
24     if ι%λ = 0 then
25         if random(0,1) < μ then
26             s = initial-solution();
27         else
28             s = nearest-neighbor-heuristic();
29     ι = ι + 1;
30 until time limit is exceeded;
```

removed, this weight determines the order in which an item reenters. The values of these weights correspond to values between 0 and 1.

The decoding procedure of the 6-tuple to generate a feasible solution is detailed below.

4.2.1. Pickup route decoding

The decoding procedure begins with the item in the first position of $s.r^p$ into the container, being inserted into the container, with the associated $s.o^d$ always being 0. Then, the remaining items are

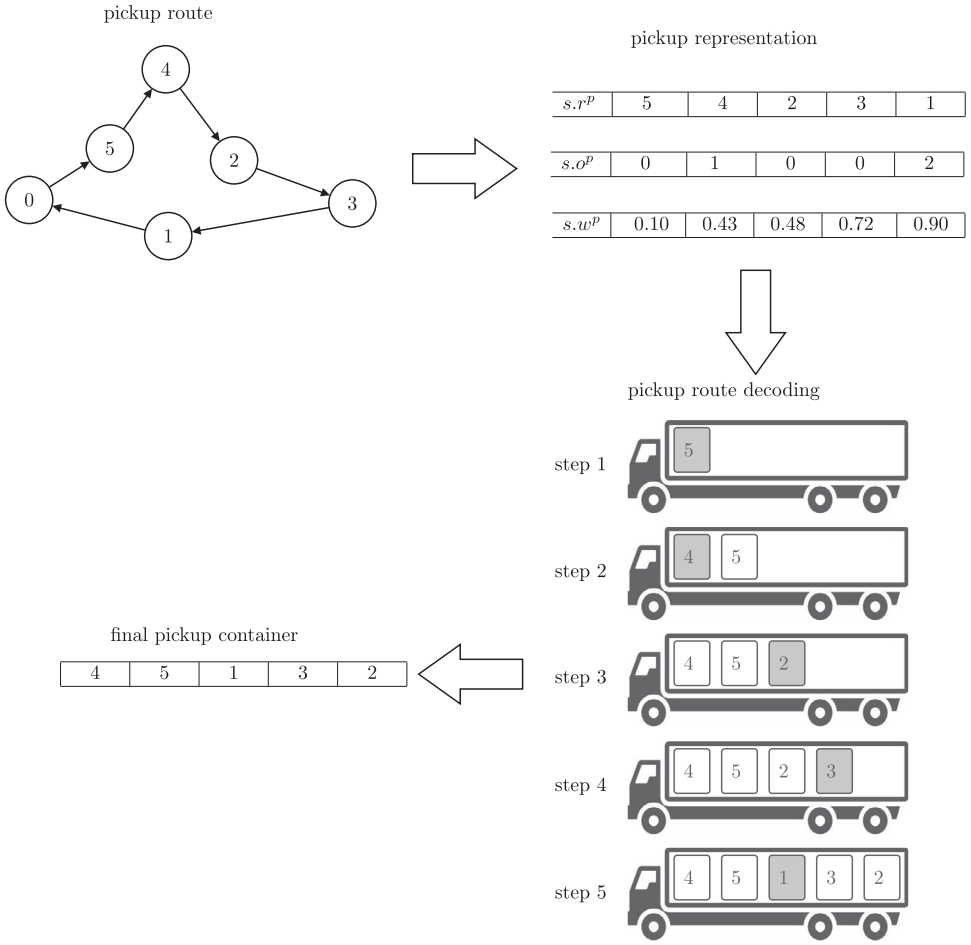


Fig. 2. Decoding procedure of the pickup route.

inserted one by one into the container, from the second to the last item, in the same order as the customers of $s.r^p$, if the corresponding $s.o^p$ is 0, that is, $s.o^p_i = 0$, $i \in \{0, 1, \ldots, n-1\}$. Otherwise, the item is inserted t positions before the last item in the container, with $t = s.o^p_i$, $i \in \{0, 1, \ldots, n-1\}$. In addition, when $t > 1$, the items that remain after the recently inserted item are sorted in nondecreasing order according to their corresponding $s.w^p$. The order of the final pickup container is obtained at the end of this procedure. The number of rearrangements at this stage corresponds to the sum of all the values in the pickup rearrangement list.

Figure 2 shows the procedure outlined above, based on the example in Fig. 1. In step 1, item 5 is first in $s.r^p$, so it inserts first in the container. Step 2, item 4, has its corresponding rearrangement equal to 1, then one position is inserted before the current item in the container. Subsequently, in steps 3 and 4, items 2 and 3 are inserted, respectively, since their corresponding rearrangement equals 0. Finally, in step 5, item 1 has a rearrangement of 2, so it is inserted two positions before the last item. Meanwhile, items 2 and 3 are sorted in nondecreasing order by their corresponding $s.w^p$,

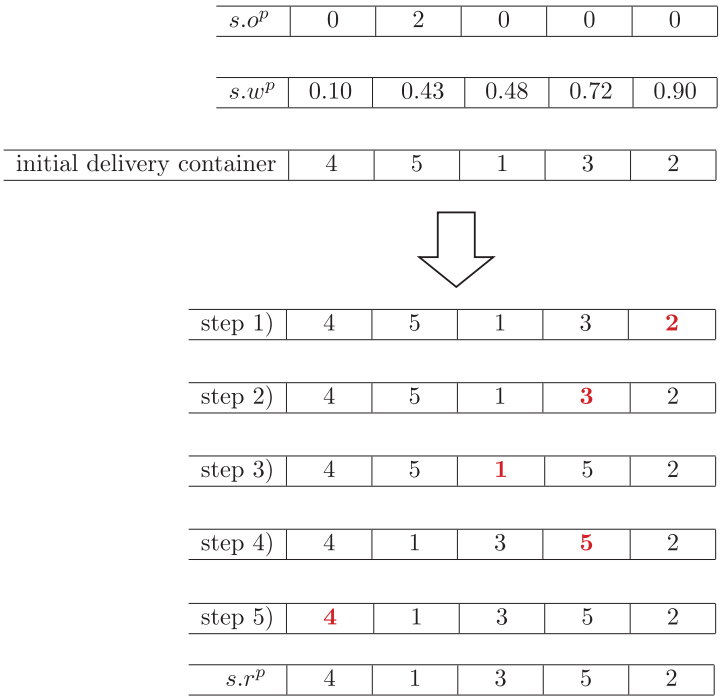


Fig. 3. Decoding procedure of the delivery route.

0.60 (3), and 0.68 (2). Therefore, they change the order to 3 and 2. Finally, at the end of the pickup decoding procedure, the last step (5), the final pickup container is generated, and the number of rearrangements is three because it is the sum of all the values in the rearrangement list.

4.2.2. Delivery route decoding

In order to perform this decoding, the pickup route decoding is necessary because the final pickup container obtained there, now in this procedure, corresponds to the initial delivery container. The delivery route decoding starts with the last item of the initial delivery container and its corresponding $s.o^d$ is always 0. From the next to last to the first item are kept in the same order, if the corresponding $s.o^d$ is 0, that is, $s.o^d_i = 0, i \in \{0, 1, \ldots, n-1\}$. Otherwise, then the item is moved t positions toward the end of the container, with $t = s.o^p_i, i \in \{0, 1, \ldots, n-1\}$. In addition, when $t > 1$, the items that remain between the positions where the recently moved item was initially, and its new position are sorted in nondecreasing order according to their corresponding $s.w^d$. The sum of all the values in the delivery rearrangement list is used to calculate the number of rearrangements at this stage.

Based on the example in Fig. 1, the procedure described above can be seen in Fig. 3, and the current item is highlighted in red. Step 1, since item 2 is the last in the initial delivery container, so it is kept as the last. Then, in steps 2 and 3, items 3 and 1 are kept in the same position since their corresponding $s.o^d$ is 0. In step 4, item 5 has its corresponding rearrangement in 2, so it moves in 2 positions toward the end of the container. Meanwhile, items 1 and 3 that remain in between the

previous and new positions of item 5 are sorted in nondecreasing order by their corresponding $s.w^d$, 0.48 (1), and 0.72 (3), remaining in the same order, 1 and 3. Finally, in step 5, item 4 is kept in the same position in the container since its corresponding reorder is 0. Once the procedure is finished, the delivery route is obtained. Therefore, the delivery plan in practice is carried out as seen in Fig. 1. The number of rearrangements is two since it is the sum of all the values in the rearrangement list.

4.3. Initial solution

Our initial solution is the same used by Chagas et al. (2022) and Felipe et al. (2009a) for a generating feasible pickup route. The procedure considers the case without allowing rearrangements, that is, $L = 0$. For this case, the pickup-and-delivery routes are opposite exactly since each loading/unloading operation must be verified by the LIFO policy, considering that the vehicle has a loading compartment with a single stack. Therefore, this case can solve as a TSP, where each arc (i, j) has an associated cost $c_{ij} = c^p_{ij} + c^d_{ji}$. Then, the TSP is solved through the classical model of two indices (Applegate et al., 2007) by adding subtour elimination constraints iteratively until the pickup route ($s.r^p$) does not contain subtours (Gutin and Punnen, 2007). Subsequently, in order to complete the remaining tuples of s, the first time, the remaining tuples take only values 0 in their lists in such a way that the decoding generates pickup-and-delivery routes opposite exactly. Then, the next calls are generated the following way: $s.o^q_i, \forall i \in \{0, \ldots, n-1\}, q \in Q$ is generated with a random value, which if is less than 0.25, each value is set between 1 and $\min(i, L)$ for the pickup area and between 1 and $\min(n - i, L)$ for the delivery area. Otherwise, a 0 is set. Finally, $s.w^q_i, \forall i \in \{1, \ldots, n\}, q \in Q$ set as a random value between 0 and 1.

4.4. Perturbation

Each time the perturbation strategy is called in Algorithm 1 in line 3, only one of five perturbation procedures is used. The perturbation procedure is chosen according to a randomly generated value (ω) between 0 and 1 that matches some given probabilities of the perturbation procedures. Thus, the ranges of the probabilities for Perturbation 1 ($0 \leq \omega \leq \alpha_1$), Perturbation 2 ($\alpha_1 < \omega \leq \alpha_1 + \alpha_2$), Perturbation 3 ($\alpha_1 + \alpha_2 < \omega \leq \alpha_1 + \alpha_2 + \alpha_3$), Perturbation 4 ($\alpha_1 + \alpha_2 + \alpha_3 < \omega \leq \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$), and Perturbation 5 ($1 - \alpha_5 < \omega \leq 1.0$). Next, the description of the perturbation procedures.

- Perturbation 1: given $s.r^p$, exchange a randomly chosen customer from the route with the next customer in route of the previously chosen customer.
- Perturbation 2: given $s.r^p$, this route is randomly shuffled.
- Perturbation 3: Given $s.o^p$ and $s.o^d$, select randomly a position and change its value by a random value between 0 and L, which satisfies with the rules of the value of each area.
- Perturbation 4: Given $s.w^p$ and $s.w^d$, change each element of $s.w^p$ and $s.w^p$ by a random number between 0 and 1, a probability of 0.5, otherwise, element value is kept.
- Perturbation 5: Given $s.w^p$ and $s.w^d$, change each element of $s.w^p$ and $s.w^p$ by a random number between 0 and 1.

4.5. Local search

Our local search strategy is divided into route and rearrangement improvement procedures. The route improvement procedures are 2-opt (Croes, 1958) and 3-opt (Lin, 1965) with a probability θ. The rearrangement improvement procedures are a partial rearrangement local search and a full rearrangement local search, both with a probability β. The local search of partial rearrangement consists of an iterative process on the current solution. It iteratively checks that $s.o_i^q = l$, $\forall i \in \{0, \ldots, n-1\}$, $\forall l \in \{0, \ldots, L\}$, $q \in Q$ and if its new cost is less than the current cost. Then, the procedure finishes with the first improvement in the partial version. Meanwhile, the full version attempts to incorporate as many improvements as possible. Finally, we defined a fifth local search of route improvement, which is applied inside the intensification procedure. In this local search, two customers are randomly chosen and exchanged on the route. If an improvement is obtained, the exchange is applied. Otherwise, the original tour is not changed. This procedure is repeated n times.

5. Computational results

In this section, we present the computational experiments performed to show the effectiveness of our algorithm. Therefore, we compare ILS with the BRKGA algorithm by Chagas et al. (2022), the state-of-the-art algorithm for the DTSPPL. Source code of BRKGA is available in Chagas et al. (2022) and is written in C++ programming language, working with a given time limit. Our metaheuristic is implemented in the C++ programming language using the GNU Compiler Collection. In addition, Gurobi Optimizer 9.1.1 is used to solve the two-index model for the TSP in the initial solution. The computational experiments of both algorithms were performed on the supercomputing infrastructure of the National Laboratory for High Performance Computing in the node with a Dell PowerEdge R640 and 2 processors Intel Xeon Gold 6152 CPU 2.10 GHz, with 22 cores and 768 GB of RAM (all experiments were running with a single thread), using the CentOS Linux 7 for 64 bits.

In order to guarantee a fair comparison with the BRKGA, our experiments were conducted by executing both algorithms in ten independent runs with different random seeds for each instance, considering three different time limits for each set of instances. Accordingly, we defined these sets of time limits based on the sizes of the sets of instances. Hence, we set the time limits of 1, 5, and 10 seconds for the small-sized set (Set 1). Next, the medium-sized set (Set 2) was set with time limits of 10, 30, and 60 seconds. Finally, we set the time limits of 60, 180, and 300 seconds for the large-sized set (Set 1).

5.1. Benchmark instances

We consider 1080 instances (Set 1) proposed by Chagas et al. (2022) to evaluate the quality of the proposed algorithm. They define a number of customers $n = \{6, 8, 10, 12, 15, 20\}$, a reloading depth $L = \{1, 2, 3, 4, 5, n\}$, and a cost h varying between $\{0, 1, 2, 5, 10, 20\}$. In addition, for each area, the instances proposed in Petersen and Madsen (2009) are used, which define areas R05, R06, R07, R08, and R09. Each area has pickup-and-delivery locations. These areas were randomly created in a 100×100 square, the depot is placed in the center of the square, that is,

at coordinates (50, 50), while the next n points define the n customer locations. Besides, all distances are Euclidean and rounded to the nearest integer, according to conventions used by TSPLIB (Reinelt, 1991).

On the other hand, we generate two new sets of instances, a medium sized (Set 2) and a larger sized (Set 3). For the construction, the same procedure to generate the instances from Chagas et al. (2022) is used, except for the areas R20, R21, R22, R23, and R24 that we created follow the above procedure by Petersen and Madsen (2009). Thus, Set 2 contains 1080 instances from 25 to 50 customers, and we consider the areas R10, R11, R12, R13, and R14, $n = \{25, 30, 35, 40, 45, 50\}$, $L = \{1, 2, 5, 10, 15, n\}$, and $h = \{0, 1, 2, 5, 10, 20\}$. While, Set 3 contains 360 instances from 75 to 100 customers and we consider the areas R20, R21, R22, R23, and R24, $n = \{75, 100\}$, $L = \{1, 10, 20, 40, 50, n\}$, and $h = \{0, 1, 2, 5, 10, 20\}$. Additionally, we define the BKS value for each instance. Thus, the BKSs for Set 1 correspond to the best solution found by any considered methods: ILP1 (Chagas et al., 2022), ILP2 (Chagas et al., 2022), BRKGA, and ILS, whereas for Sets 2 and 3, only BRKGA and ILS were considered.

5.2. Parameter tuning

Our ILS considers the following 12 parameters: $\alpha_i, i \in \{1, \ldots, 5\}$ respond to the probabilities of each perturbation procedure, and the sum of all these parameters must be 1. The parameter θ is the probability that determines the choice between 2-opt or 3-opt. The parameter β is the probability of choosing between the local search of partial or full rearrangement. The parameter ρ is the number of iterations that indicate that the intensification procedure is applied once for every ρ iterations. τ is the coefficient used in the acceptance criterion. $|e|$ indicates the maximum size of the elite solutions. The parameter λ is the number of iterations that indicates for once every λ iterations is applied to the restart procedure. Finally, μ is the probability to decide with which algorithm the solution is restarted.

A good choice of parameter values is critical in metaheuristic algorithms. For this reason, we adopted an effective automated method for parameter tuning, called irace (López-Ibáñez et al., 2016), which is implemented in the R programming language and is based on the iterated racing procedure. We applied irace on a subset of training instances since the process of determining good settings is very time-consuming. The selection of the instances was carried out experimentally by running ILS with a set of preliminary parameters, giving 20 instances. The 20 selected instances were those in which ILS had the worst performance. Thus, according to this format {area, L, h, n}, we consider the following selected instances: {R07, 5, 0, 20}, {R07, 20, 0, 20}, {R09, 20, 0, 20}, {R08, 20, 0, 20}, {R08, 5, 0, 20}, {R05, 20, 0, 20}, {R06, 5, 0, 20}, {R07, 3, 0, 15}, {R07, 5, 0, 15}, {R08, 15, 2, 15}, {R11, 15, 10, 50}, {R11, 15, 10, 45}, {R10, 15, 10, 50}, {R14, 15, 10, 45}, {R14, 15, 20, 50}, {R13, 15, 20, 45}, {R10, 10, 10, 50}, {R10, 15, 20, 50}, {R11, 10, 10, 50}, {R12, 15, 10, 50}. Additionally, we define that irace runs ILS 10 times with different seeds and with a time limit of five seconds.

The parameter tuning process required approximately 10 hours of computing time, using 40 threads running on another node of the aforementioned supercomputing infrastructure. Table 3 shows the parameter name, the set of tested values or the parameter range, and the final value obtained through the irace tuning. We selected the configuration that provided, on the 20 training

Table 3
Set of parameters for tuning of ILS

Name	Different tested value	Final value
α_1	[0.0, 1.00]	0.0654
α_2	[0.0, 1.00]	0.0135
α_3	[0.0, 1.00]	0.7102
α_4	[0.0, 1.00]	0.1781
α_5	[0.0, 1.00]	0.0328
θ	[0.0, 1.00]	0.2442
β	[0.0, 1.00]	0.4427
ρ	{500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000}	3500
τ	[0.0, 0.30]	0.1067
$\|e\|$	{5, 10, 15, 20, 25, 30, 35, 40, 45, 50}	5
λ	{50,000, 100,000, 150,000, 200,000, 250,000, 300,000, 350,000, 400,000, 450,000, 500,000}	100,000
μ	[0.0, 1.00]	0.0513

Table 4
Results with 1 second for Set 1

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
6	BRKGA	2.11	2.57	1.03	102
	ILS	0.33	0.92	1.00	135
8	BRKGA	5.40	6.60	1.07	50
	ILS	1.38	2.70	1.00	70
10	BRKGA	9.67	10.80	1.14	14
	ILS	2.16	4.30	1.00	53
12	BRKGA	12.60	12.95	1.25	17
	ILS	2.94	5.02	1.00	57
15	BRKGA	12.30	12.34	1.65	9
	ILS	3.20	5.97	1.00	65
20	BRKGA	9.69	9.69	3.29	6
	ILS	2.35	5.05	1.00	65
Overall	BRKGA	8.63	9.16	1.57	198
	ILS	2.06	3.99	1.00	445

instances, the best comparison in terms of the gap to the BKS, and the corresponding values are reported in the last column.

5.3. Experiments on instances of Set 1

This computational experiment has been performed on the instances of Set 1. Tables 4–6 present the summary of results of the 1080 instances of Set 1 with the time limits of 1, 5, and 10 seconds, respectively. The first column corresponds to the number of customers (n) in the corresponding set

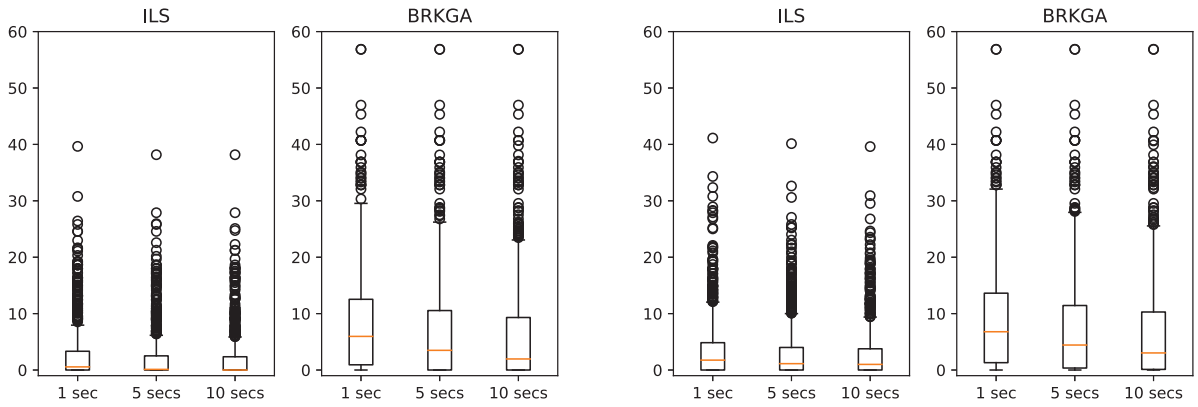
Table 5
Results with five seconds for Set 1

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
6	BRKGA	0.60	1.00	5.03	129
	ILS	0.38	0.66	5.00	133
8	BRKGA	3.47	4.12	5.06	83
	ILS	1.53	2.26	5.00	73
10	BRKGA	7.01	8.18	5.14	38
	ILS	2.59	3.88	5.00	53
12	BRKGA	10.34	11.30	11.58	27
	ILS	3.49	4.61	5.00	62
15	BRKGA	10.88	11.16	5.22	18
	ILS	3.16	4.27	5.00	87
20	BRKGA	8.95	8.96	6.28	6
	ILS	1.66	2.53	5.00	122
Overall	BRKGA	6.88	7.45	6.38	301
	ILS	2.14	3.04	5.00	530

Table 6
Results with 10 seconds for Set 1

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
6	BRKGA	0.40	0.67	10.03	143
	ILS	0.36	0.62	10.00	136
8	BRKGA	2.47	3.09	16.51	93
	ILS	1.47	2.16	10.00	76
10	BRKGA	5.62	6.81	10.11	52
	ILS	2.30	3.69	10.00	54
12	BRKGA	8.74	10.11	10.19	41
	ILS	3.23	4.46	10.00	65
15	BRKGA	10.05	10.55	10.31	33
	ILS	2.86	4.03	10.00	106
20	BRKGA	8.36	8.50	10.97	10
	ILS	1.52	2.34	10.00	155
Overall	BRKGA	5.94	6.62	11.35	372
	ILS	1.96	2.88	10.00	592

and the average over the 180 instances in the set. The second and third columns report the best (Best GAP) and the average (Avg GAP) solution values obtained in 10 runs, respectively, the average over 180 instances of the average of the percentage gaps between the values of the best-found solution by an algorithm and the corresponding BKSs. While in the fourth and fifth columns are average computing times in seconds over the 10 runs and HIT, respectively, where HIT corresponds to the number of times that the algorithm found the Best GAP equal to 0. Finally, the last row reports, for each column, the average value over all the instance sets. Note that each row of n corresponds



(a) Best GAP values.

(b) Avg GAP values.

Fig. 4. Boxplots of gaps values of the algorithms for each time limit in Set 1.

to 180 instances of size n with the different values of L, h, and areas. The details of the results for each of the instances are available at the following URL.[1]

Table 4 shows that the Best GAP, Avg GAP, and HIT of the proposed algorithm outperform values obtained by BRKGA considering one second for instances of Set 1. ILS is run exactly within the given time limit. Instead, in some cases, BRKGA exceeds the given time limit, even the time is tripled in $n = 20$. BRKGA likely exceeds the time limit because it is a population-based algorithm. Therefore, computing the fitness function is computationally expensive since it must encode and decode each solution.

Table 5 shows that the results have similar behavior to Table 4. Both algorithms improve the quality of the solutions by having a fivefold increase in time, while ILS continues to outperform BRKGA in Best GAP, Avg GAP, and the HIT, except $n = 8$ in HIT. In this case, the BRKGA's computing time is appreciated closer to the time limit.

Finally, Table 6 shows that the results of ILS outperform BRKGA for the Best GAP, Avg GAP, and HIT, except $n = \{6, 8\}$ in HIT. Also, the computing time of BRKGA does not substantially exceed the time limit. This behavior may be because it already has enough computing time to generate the initial population and a few iterations.

Based on the preceding results and Fig. 4, algorithms gaps in boxplots are displayed. The increase in time limit favors both algorithms because, as the time limit increases, the GAP values of both algorithms gradually decrease. Therefore, we can infer that the ILS is more efficient because it needs a short computing time to find good solutions, despite both algorithms starting from the same initial solution. In contrast, the BRKGA needs more computing time to reach solutions of similar quality as it is reported in Chagas et al. (2022) that needs at least one hour of computing time. It is worth noting that the overall averages of the BRKGA for the Best GAP, Avg GAP, and HIT with a time limit of 10 seconds cannot reach the results obtained by the ILS with only a time limit of 1 second.

[1]See https://sites.google.com/view/ccontrerasbolton/instances-codes.

Table 7
Results with 10 seconds for Set 2

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
25	BRKGA	8.60	8.65	13.86	13
	ILS	0.71	1.86	10.00	84
30	BRKGA	9.49	9.49	20.26	13
	ILS	1.00	2.78	10.00	61
35	BRKGA	9.20	9.20	29.15	12
	ILS	0.86	2.48	10.00	64
40	BRKGA	7.10	7.10	40.07	16
	ILS	0.75	2.05	10.00	50
45	BRKGA	7.59	7.59	55.15	13
	ILS	0.71	1.93	10.00	46
50	BRKGA	7.61	7.61	79.06	12
	ILS	0.68	1.81	10.00	44
Overall	BRKGA	8.27	8.27	39.59	79
	ILS	0.79	2.15	10.00	349

Table 8
Results with 30 seconds for Set 2

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
25	BRKGA	7.62	7.84	32.47	19
	ILS	0.31	1.49	30.00	124
30	BRKGA	8.88	8.97	33.19	13
	ILS	0.31	2.06	30.00	111
35	BRKGA	8.85	9.00	42.00	12
	ILS	0.32	1.91	30.00	102
40	BRKGA	7.10	7.10	54.22	16
	ILS	0.24	1.52	30.00	102
45	BRKGA	7.59	7.59	78.42	13
	ILS	0.27	1.44	30.00	98
50	BRKGA	7.61	7.61	87.79	12
	ILS	0.23	1.29	30.00	90
Overall	BRKGA	7.94	8.02	54.68	85
	ILS	0.28	1.62	30.00	627

5.4. Experiments on instances of Set 2

This computational experiment has been performed on the instances of Set 2. Tables 7–9 depicted the summary of results of the 1080 instances of Set 2 with the time limits of 10, 30, and 60 seconds, respectively. The structure of the tables is identical to the tables of Set 1.

Table 7 shows that the Best GAP, Avg GAP, and HIT of the ILS outperform the BRKGA. Particularly, ILS obtained a value of the Best GAP (0.79%) more than 10-fold smaller than BRKGA (8.27%) and almost quadruple when considered Avg GAP, 2.15% versus 8.27%. Moreover, the ILS achieved precisely the given time limit in the computing times. Instead, the BRKGA exceeded the time limit of 30 seconds in all the cases. Indeed, the computing time is double for

Table 9
Results with 60 seconds for Set 2

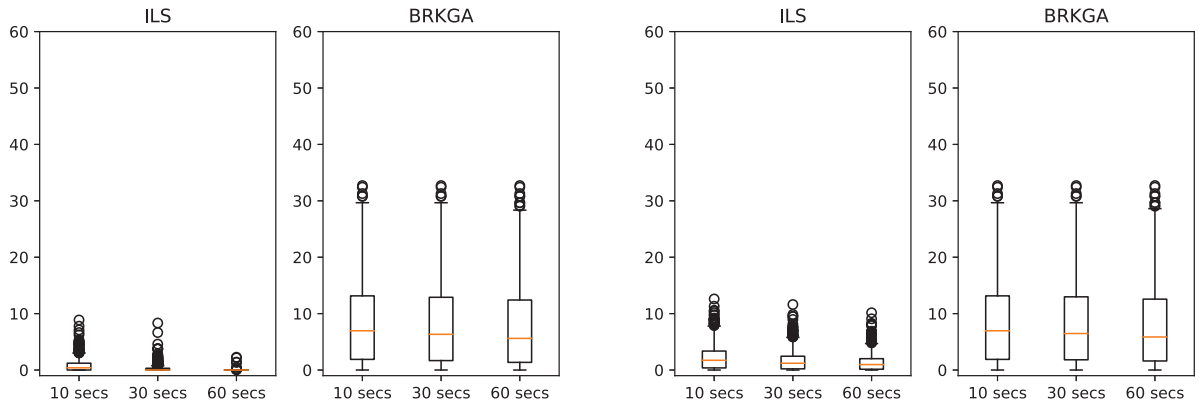
n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
25	BRKGA	7.06	7.41	77.42	27
	ILS	0.03	1.27	60.00	174
30	BRKGA	8.63	8.84	78.54	16
	ILS	0.00	1.73	60.00	178
35	BRKGA	8.42	8.50	65.66	15
	ILS	0.00	1.58	60.00	179
40	BRKGA	6.56	6.65	75.23	21
	ILS	0.01	1.27	60.00	177
45	BRKGA	7.10	7.21	86.94	15
	ILS	0.00	1.16	60.00	180
50	BRKGA	7.44	7.53	114.20	13
	ILS	0.00	1.03	60.00	180
Overall	BRKGA	7.53	7.69	83.00	107
	ILS	0.01	1.34	60.00	1068

30 customers, almost triple for 35 customers, quadruple for 40 customers, fivefold for 45 customers, almost eightfold for 50 customers, and is almost quadruple for all the instances (overall). Additionally, we can note that the Best GAP and Avg GAP are the same for the BRKGA, except for $n = 25$. This behavior is due to the BRKGA only reaching the initial solution procedure, which has two steps. The first step consists of obtaining an initial value by the same initial procedure as our algorithm, and in the second step, BRKGA must initialize each individual of its population with these initial values. This last step requires high time-consuming, so the given time limit is widely surpassed.

Table 8 shows that the results have similar behavior to Table 7 although the distance between the algorithms extends even more in favor of the ILS in the Best GAP, Avg GAP, and HIT. The BRKGA's computing time is not as long as the previous computing periods. However, it is still double the given time limit in some cases. Indeed, in $n = \{25, 30, 35\}$ the Best GAP and Avg GAP are not equal, so BRKGA achieved run some iterations.

Finally, Table 9 shows that ILS outperforms BRKGA in all the metrics. Note that ILS obtained a 0.01% in the Best GAP and 1.34% in the Avg GAP. Moreover, ILS almost achieved the total number of hits, missing only 12 instances. In all previous experiments, BRKGA reached computing times significantly longer than the time limit of 60 seconds, although no case exceeded the computing times in the double like in the previous runnings of the time limits.

As we can observe in all the results for the instances of Set 2, both algorithms improve their performance as time grows time limit, but pretty slight the BRKGA's improvement, as we can depict graphically in Fig. 5. However, ILS is again more efficient since it needs a shorter time to find good-quality solutions, despite both algorithms starting from the same initial solution. In addition, we can observe that the overall BRKGA averages for the Best GAP, Avg GAP, and HIT, when considering the time limit of 60 seconds, cannot reach the results obtained by ILS with the time limit of 10 seconds.



(a) Best GAP values.

(b) Avg GAP values.

Fig. 5. Boxplots of gaps values of the algorithms for each time limit in Set 2.

Table 10
Results with 60 seconds for Set 3

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
75	BRKGA	5.28	5.28	652.85	14
	ILS	0.74	1.57	60.00	32
100	BRKGA	5.51	5.51	1505.30	9
	ILS	1.22	2.09	60.00	31
Overall	BRKGA	5.39	5.39	1079.07	23
	ILS	0.98	1.83	60.00	63

Table 11
Results with 180 seconds for Set 3

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
75	BRKGA	4.96	5.03	689.18	17
	ILS	0.22	1.03	180.00	80
100	BRKGA	5.51	5.51	1525.29	9
	ILS	0.36	1.25	180.00	64
Overall	BRKGA	5.24	5.27	1107.24	26
	ILS	0.29	1.14	180.00	144

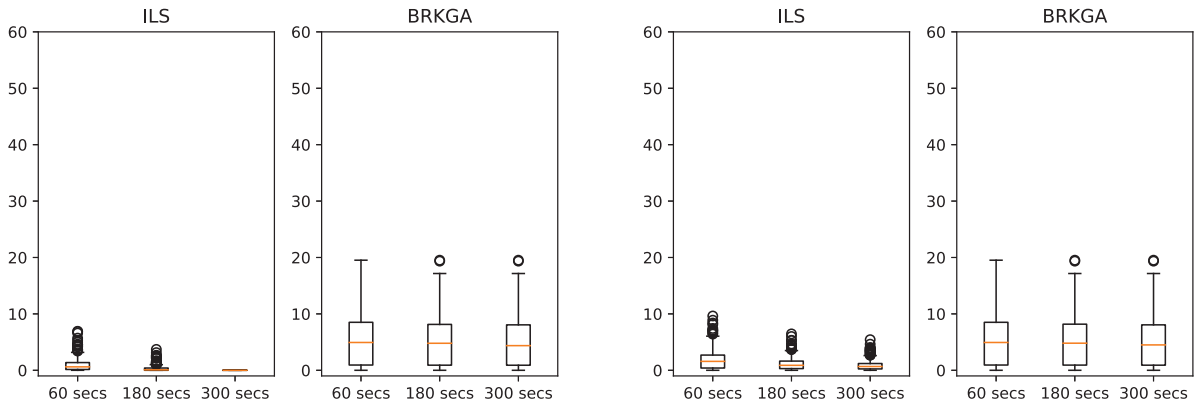
5.5. Experiments on instances of Set 3

This computational experiment has been performed on the instances of Set 3. Tables 10–12 show the summary of results of the 360 instances of Set 3 with the time limits of 60, 180, and 300 seconds, respectively. The structure of the tables is identical to the tables of Sets 1 and 2.

Tables 10–12 show that the Best GAP, Avg GAP, and HIT of the ILS outperform the BRKGA in this set of larger instances. In Table 12 that corresponds to the time limit with greater time, 300 seconds, we can see that the ILS obtained 0.00% against 5.11% of BRKGA in the Best GAP, 0.88%

Table 12
Results with 300 seconds for Set 3

n	Algorithm	Best GAP (%)	Avg GAP (%)	Time (seconds)	HIT
75	BRKGA	4.76	4.82	749.24	17
	ILS	0.00	0.85	300.00	180
100	BRKGA	5.47	5.49	1565.93	9
	ILS	0.00	0.92	300.00	180
Overall	BRKGA	5.11	5.15	1157.58	26
	ILS	0.00	0.88	300.00	360



(a) Best GAP values.

(b) Avg GAP values.

Fig. 6. Boxplots of gaps values of the algorithms for each time limit in Set 3.

against 5.15% in Avg GAP, for ILS and BRKGA, respectively. In terms of hits, ILS obtained all the hits against 26 hits of BRKGA. Again, the BRKGA achieved computing times significantly longer than the given time limits. Thus, we observe that the BRKGA in the set of instances needs more than 1000 seconds to be able to generate initial solutions, as shown in Table 10 since the Best GAP and Avg GAP are the same. Then, with time limits greater, BRKGA achieved to run some iterations and slightly improve its results.

Generally, ILS improves its performance as the time limit increases, which we can observe in all the results for the instances of Set 3. In contrast, BRKGA improves performance slightly when the time limit increases, as shown in Fig. 6. Therefore, ILS is again more efficient since it needs a shorter time to find good-quality solutions. Furthermore, another issue with the BRKGA is its high memory consumption while solving larger instances. Indeed, some instances needed approximately 400 GB of RAM. In contrast, ILS needed at most 21 MB of RAM for the larger instances.

6. Conclusions

In this paper, a new algorithm based on ILS was proposed for solving the DTSPPL. The proposed metaheuristic contains six components: initial solution procedure, perturbation strategies, local

search strategies, acceptance criterion, intensification, and restart procedures. The results obtained in the computational experiments show that the ILS outperforms the state-of-the-art algorithm on a set of 2520 instances by generating good-quality solutions in a set of given time limits. Additionally, we proposed two new benchmarks of instances: the first set consists of 1080 instances with several customers ranging between 25 and 50, and the second set has 360 instances with 75 and 100 customers.

A potential future research direction is to extend the proposed algorithm to tackle the DTSPMS or the DTSPPL. Another area of research is the development of exact algorithms based on a new formulation model for efficiently addressing the small (that are still challenging to solve) and the proposed new instances. Additionally, the exact algorithms could also benefit from a warm start obtained by applying our proposed approach. Finally, it could be interesting to extend the proposed algorithm to solve some of the variants of the TSP recently published (Bernardino and Paias, 2021; Cacchiani et al., 2021; Cuellar-Usaquén et al., 2022).

Acknowledgments

This research was partially supported by the supercomputing infrastructure of the NLHPC (ECM-02) and also partially funded by VRID – INICIACIÓN 220.097.016-INI, Vicerrectoría de Investigación y Desarrollo (VRID), Universidad de Concepción and the Research Management of Universidad Andres Bello with the grant number DI12-20/REG. We would like to express our gratitude to the anonymous reviewers for their insightful suggestions, which helped us enhance our work.

References

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

Alfandari, L., Toulouse, S., 2021. Approximation of the double traveling salesman problem with multiple stacks. *Theoretical Computer Science* 877, 74–89.

Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J., 2007. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ.

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

Bernardino, R., Paias, A., 2021. Heuristic approaches for the family traveling salesman problem. *International Transactions in Operational Research* 28, 1, 262–295.

Cacchiani, V., Contreras-Bolton, C., Escobar-Falcón, L.M., Toth, P., 2021. A matheuristic algorithm for the pollution and energy minimization traveling salesman problems. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.12991.

Cacchiani, V., Contreras-Bolton, C., Toth, P., 2020. Models and algorithms for the traveling salesman problem with time-dependent service times. *European Journal of Operational Research* 283, 3, 825–843.

Carrabs, F., Cerulli, R., Cordeau, J.F., 2007a. 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, 223–238.

Carrabs, F., Cerulli, R., Speranza, M.G., 2013. A branch-and-bound algorithm for the double travelling salesman problem with two stacks. *Networks* 61, 1, 58–75.

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

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

Chagas, J.B.C., Toffolo, T.A.M., Souza, M.J.F., Iori, M., 2022. The double traveling salesman problem with partial last-in-first-out loading constraints. *International Transactions in Operational Research* 29, 4, 2346–2373.

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

Côté, J.F., Archetti, C., Speranza, M.G., Gendreau, M., Potvin, J.Y., 2012. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks* 60, 4, 212–226.

Croes, G.A., 1958. A method for solving traveling-salesman problems. *Operations Research* 6, 6, 791–812.

Cuellar-Usaquén, D., Gomez, C., Álvarez Martínez, D., 2022. A grasp/path-relinking algorithm for the traveling purchaser problem. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.12985.

Dantzig, G.B., Ramser, J.H., 1959. The truck dispatching problem. *Management Science* 6, 1, 80–91.

Dell'Amico, M., Montemanni, R., Novellani, S., 2022. Exact models for the flying sidekick traveling salesman problem. *International Transactions in Operational Research* 29, 3, 1360–1393.

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

Felipe, A., Ortuño, M., Tirado, G., 2009b. New neighborhood structures for the double traveling salesman problem with multiple stacks. *TOP* 17, 190–213.

Gutin, G., Punnen, A., 2007. *The Traveling Salesman Problem and Its Variations*, *Combinatorial Optimization*, Vol. 12. Springer US, Boston, MA.

Hà, M.H., Nguyen Phuong, H., Tran Ngoc Nhat, H., Langevin, A., Trépanier, M., 2022. Solving the clustered traveling salesman problem with *d*-relaxed priority rule. *International Transactions in Operational Research* 29, 2, 837–853.

Hernández-Pérez, H., Salazar-González, J.J., 2022. A branch-and-cut algorithm for the split-demand one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research* 297, 2, 467–483.

Hvattum, L.M., Tirado, G., Felipe, A., 2020. The double traveling salesman problem with multiple stacks and a choice of container types. *Mathematics* 8, 6.

Iori, M., Martello, S., 2010. Routing problems with loading constraints. *TOP* 18, 1, 4–27.

Karp, R.M., 1972. Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds) *Complexity of Computer Computations*. Springer US, Boston, MA, pp. 85–103.

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

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

Lin, S., 1965. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal* 44, 10, 2245–2269.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search. In Glover, F. and Kochenberger, G.A. (eds) *Handbook of Metaheuristics*. Springer US, Boston, MA, pp. 320–353.

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

Lusby, R.M., Larsen, J., 2011. Improved exact method for the double tsp with multiple stacks. *Networks* 58, 4, 290–300.

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

Magableh, G.M., 2021. Supply chains and the COVID-19 pandemic: a comprehensive framework. *European Management Review* 18, 3, 363–382.

Pantuza Jr, G., de Souza, M.C., 2022. Formulations and a lagrangian relaxation approach for the prize collecting traveling salesman problem. *International Transactions in Operational Research* 29, 2, 729–759.

Pereira, A.H., Mateus, G.R., Urrutia, S.A., 2022. Valid inequalities and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *European Journal of Operational Research* 300, 1, 207–220.

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

Petersen, H., 2009. Decision Support for Planning of Multimodal Transportation with Multiple Objectives. Ph.D. thesis, Technical University of Denmark.

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

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

Reinelt, G., 1991. Tsplib—a traveling salesman problem library. *ORSA Journal on Computing* 3, 4, 376–384.

Ruan, M., Shen, C., Tang, J., Qi, C., Qiu, S., 2021. A double traveling salesman problem with three-dimensional loading constraints for bulky item delivery. *IEEE Access* 9, 13052–13063.

Sampaio, A.H., Urrutia, S., 2017. New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research* 24, 1–2, 77–98.

Toth, P., Vigo, D., 2014. *Vehicle Routing: Problems, Methods, and Applications* (2nd edn). Society for Industrial and Applied Mathematics, Philadelphia, PA.

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

World Bank, 2020. The World Bank Annual Report 2020: supporting countries in unprecedented times. Technical Report, World Bank, Washington, DC.

World Trade Organization, 2020. World Trade Report 2020. Technical Report, World Trade Organization, Ginebra.