Lisbon School of Economics & Management
Universidade de Lisboa

MESTRADO EM

MÉTODOS QUANTITATIVOS PARA A DECISÃO ECONÓMICA E EMPRESARIAL

TRABALHO FINAL DE MESTRADO

DISSERTAÇÃO

AN ITERATED LOCAL SEARCH ALGORITHM FOR THE TRAVELING PURCHASER PROBLEM

TOMÁS SILVA KAPANCIOGLU

ORIENTAÇÃO:

PROFª DOUTORA RAQUEL MONTEIRO DE NOBRE COSTA BERNARDINO

DOCUMENTO ESPECIALMENTE ELABORADO PARA OBTENÇÃO DO GRAU DE MESTRE

MARÇO – 2024

Acknowledgements

I am very fortunate to have been able to learn so much from my supervisor, Dr. Raquel Bernardino. Thank you for your kindness and patience.

Abstract

The Traveling Purchaser Problem (TPP) is a generalization of the Traveling Salesman Problem (TSP) in which a list of items must be acquired by visiting a subset of markets. The objective is to minimize the total cost sustained along the route, including purchasing and traveling costs. Due to the NP-hard nature of the problem, solving the TPP in an exact manner is computationally challenging, implying the need for heuristic approaches in order to obtain quality solutions efficiently. This study proposes an algorithm based on the metaheuristic Iterated Local Search (ILS), complemented by a route configuration procedure that adjusts the subset of markets in the solution. The algorithm is tested in benchmark instances, providing a performance comparison with other methods. The computational experiment for the asymmetric instances reveals the effectiveness and efficiency of the algorithm, outperforming previously published results with statistical significance. Additional experiments are presented for the symmetric instances, pointing to the competitiveness and versatility of the algorithm in relation to other heuristic approaches used in the literature.

Keywords: Traveling purchaser problem, Metaheuristics, Iterated local search, Route configuration.

Resumo

O problema do comprador viajante é uma generalização do problema do caixeiro viajante, no qual uma lista de itens tem de ser adquirida ao visitar um subconjunto de mercados. O objetivo consiste em minimizar o custo total que inclui os custos de compra dos itens e os custos de deslocação. Trata-se de um problema NP-difícil, pelo que resolvê-lo de forma exata é ineficiente em termos computacionais, implicando a necessidade de recorrer a métodos heurísticos de modo a obter soluções de qualidade de forma eficiente. Este estudo propõe um algoritmo baseado no conceito metaheurístico de pesquisa local iterativa, complementado com um procedimento de configuração de rota que ajusta o subconjunto de mercados da solução. O algoritmo é testado em instâncias de referência, proporcionando uma comparação de desempenho com outros resultados na literatura. A experiência computacional para as instâncias assimétricas revela a eficácia e eficiência do algoritmo, obtendo melhores resultados com significância estatística. São apresentadas experiências computacionais adicionais para as instâncias simétricas, apontando a competitividade e versatilidade do algoritmo em relação a outros métodos heurísticos usados na literatura.

Palavras-Chave: Problema do comprador viajante, Meta-heurísticas, Pesquisa local iterativa, Configuração de rota.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Procurement activities stand as an opportunity for businesses to secure competitive advantages. Supplier selection is revealed to be a common challenge that companies are faced with, as it affects the overall business performance. Bearing this in mind, companies may resort to Operational Research tools to improve their decision-making processes.

Considering a list of items and multiple markets, a purchaser must determine which markets to visit and which items to acquire at each market. The route begins and ends at a depot, visiting a sequence of markets. Moving from one location to another incurs a specific traveling cost, which may vary between every pair of locations. Additionally, the price of an item can differ across different markets. The purchaser seeks to determine the optimal set of markets to visit, the sequence in which to visit them, and the items to purchase at each market to minimize the total costs, composed of the traveling and purchasing costs. This is the Traveling Purchaser Problem (TPP), a widely known combinatorial optimization problem (Ramesh, 1981). Tackling the TPP can be very beneficial for many companies as it generates the purchasing plan, determines the route to follow, and resolves the supplier selection intricacy.

The TPP is classified as NP-hard (Golden et al., 1981) because it generalizes the Traveling Salesman Problem (TSP). Hence, heuristic approaches are essential to obtain quality solutions efficiently. This study proposes an algorithm based on the metaheuristic Iterated Local Search (ILS) complemented with a route configuration procedure that adjusts the subset of markets in the solution. Benchmark instances are considered to test the performance of the algorithm, providing a comparison with the best results in the literature.

Chapter 2 introduces the TPP and some of its most studied variants, along with the adopted notation and mathematical formulation. Chapter 3 presents the proposed algorithm, including the ILS pseudocode, the considered neighborhoods, and the complementary procedures. Chapter 4 presents the computational experiment, including a parameter examination and an analysis to ascertain the consistency of the algorithm in relation to the incorporated randomized features. Subsequently, a comparison is presented with other methods used in the literature for asymmetric and symmetric TPP benchmark instances. Chapter 5 concludes the study and provides opportunities for future work.

Chapter 2

Traveling Purchaser Problem

This chapter begins by providing an overview of the state of the art regarding the TPP and its variants in Section 2.1. Section 2.2 presents the problem definition and the variant addressed in this study, followed by the adopted notation and an instance example. Section 2.3 presents the mathematical formulation used in this study to attain the optimal solutions for the benchmark instances.

2.1 Literature Review

The TPP was initially introduced as a routing problem by Ramesh (1981), although the corresponding combinatorial structure can be traced back to scheduling problems (Burstall, 1966). It is a generalization of the known TSP, making it NP-hard since it can be reduced to the TSP when every market offers a unique item that cannot be purchased anywhere else (Golden et al., 1981). As a result of its complexity, no exact algorithm can solve all large TPP instances efficiently, implying the necessity for heuristic approaches.

The most evident applications of the TPP revolve around procurement activities. Nonetheless, it has practical applications in other areas, including warehousing operations, job scheduling, telecommunication network design, nurse routing, and school bus routing. Due to the wide variety of applications, the TPP remains a topic of interest for researchers, as reflected by the number of papers published in recent years (Manerba et al., 2017).

A common classification of the TPP relates to quantity restrictions. Every item on the list has a specific demand that must be met. However,

limited quantities across different markets may require purchasing the same item from multiple markets. Under these circumstances, the TPP is classified as restricted (R-TPP). In contrast, the unrestricted version (U-TPP) assumes that if an item is sold at a given market, the available quantity can completely satisfy the respective demand. Another common classification revolves around graph symmetry. The symmetric version (STPP) assumes that traveling from location A to location B incurs the same cost as traveling from location B to location A. Conversely, in the asymmetric version (ATPP), traveling costs may differ depending on the traversal direction between two locations.

During the past decades, several variants have been introduced in the literature. The multi-vehicle TPP (MVTPP) considers a fleet of vehicles rather than a single one. Most studies of this variant focus on a homogeneous fleet of vehicles possessing equal carrying capacity. Collectively, the fleet attempts to acquire the item list efficiently. Resorting to multiple vehicles is crucial when specific items are unable to be carried in the same vehicle, as addressed by Manerba and Mansini (2015) and subsequently in the work of Gendreau et al. (2016) in variants of the MVTPP with pairwise incompatibility constraints (MVTPP-PIC). The transportation of incompatible dangerous goods is one of the real-world situations in which these constraints are necessary. Additional applications of the MVTPP and corresponding variants include the school bus routing problem (Riera-Ledesma and Salazar-Gonzalez, 2012) and the nurse routing problem (Manerba and Mansini, 2016). In the first application, each student (item) resides within a specific distance from multiple bus stops (markets). The objective is to collect all students while minimizing the total distance traveled by both the vehicles and the students. In the second application, the nurse routing problem links the concept of items to a range of timed services offered to geographically dispersed patients. Each patient (market) may be subject to a subset of services (items). Under predetermined priorities, the nurses must deliver various services throughout the day, maximizing the overall benefit provided while ensuring that a minimum amount of each service is delivered to the patients. More recently, Bianchessi et al. (2021) proposed a branch-price-and-cut algorithm to solve diverse variants of the MVTPP with unitary demands.

The total cost is composed by the sum of the traveling and purchasing costs. However, one of the components may hold a higher degree of priority in terms of optimization, as both components are often conflicting. This originates the bi-objective TPP (2TPP), initially introduced by Riera-

Ledesma and Salazar-González (2005). Developments of the 2TPP have been explored, namely the use of transgenetic algorithms in the work of Almeida et al. (2012). Palomo-Martínez and Salazar-Aguilar (2019) considered the traveling time spent under a variant of the 2TPP where the purchased items are meant to be delivered to a set of waiting customers. Due to the impact of road transportation on the emission of greenhouse gases, the bi-objective approach has been used to model fuel consumption and respective emissions, as addressed in the work of Cheaitou et al. (2021).

A wide variety of additional constraints have been considered, including a limit to the total traveling distance (Bianchessi et al., 2014), procurement budget restrictions (Mansini and Tocchella, 2009), and limitations to the number of visited markets, and purchased items at each market (Gouveia et al., 2011). More recently, Kucukoglu (2022) considered a time limit to complete the route, with the possibility of the fast service option. Traveling time was taken into account, as well as purchasing times. At each market, the purchaser can request the fast service option, reducing the purchasing time by paying an additional cost. This may be necessary to accelerate the process to avoid exceeding the stipulated time limit.

The introduction of uncertainty to specific elements of the problem has led to the use of stochastic methods in the literature. For instance, the available quantities may be unknown until the markets are visited. Furthermore, these quantities may decay over time according to a random process (Angelelli et al., 2009, 2016), as items may be sold to third parties. The item price has also been presumed random, following a given probabilistic distribution (Kang and Ouyang, 2011). Both of the previously mentioned characteristics were jointly addressed in the work of Beraldi et al. (2017).

Despite the degree of complexity, exact algorithms have been able to solve TPP instances of considerable size. Laporte et al. (2003) proposed a branch-and-cut method for the symmetric version of the R-TPP, able to solve instances up to 200 nodes and 200 items. Later, Riera-Ledesma and Salazar-González (2006) proposed an extension of the branch-and-cut method for the asymmetric version of the R-TPP, able to solve instances of similar size. Other methods proposed in earlier years solved smaller instances, including the lexicographic search from Ramesh (1981) and the branch-and-bound approach by Singh and van Oudheusden (1997). For an extensive survey of the TPP literature, see the work of Manerba et al. (2017). Although exact methods have been developed, the computational efforts revealed to be restrained in terms of efficiency due to the NP-hard nature of the problem, alluding to

the necessity for alternative techniques such as approximation methods.

Heuristic approaches have been applied to the TPP to provide quality solutions efficiently. The *Generalized Savings Heuristic* (Golden et al., 1981) constructs a solution for the unrestricted TPP by selecting the market that sells the most items at lower purchasing costs and then successively adds markets to the specific route position that results in the most significant improvement. The *Tour Reduction Heuristic* (Ong, 1982) modifies a feasible solution by removing the market that yields the most improvement in each iteration. The *Commodity Adding Heuristic* (Pearn and Chien, 1998) considers each item, forming a route that includes the market that minimizes the cost of acquiring that particular item. In every iteration, the cost of acquiring a new item is pondered by either choosing a visited market or adding an unvisited one to the route.

Some of the metaheuristics that have been used in TPP studies and respective variants include Ant Colony Optimization (Bontoux and Feillet, 2008), Genetic Algorithms combined with a local search (Bernardino and Paias, 2018), Adaptive Large Neighborhood Search (Kucukoglu, 2022), Simulated Annealing and Tabu Search (Voß, 1996). Most recently, Cuellar-Usaquén et al. (2023) proposed a GRASP-based methodology complemented with a Filtering and Path-Relinking strategy, providing the best results in the literature to our knowledge, in relation to the asymmetric instances addressed in this study.

2.2 Definition

The TPP consists of determining the most efficient method to purchase a list of items by visiting a subset of markets. The route begins and ends at a depot. Traveling from one location to another has an associated cost. Each item has a specific demand that must be fulfilled, and the price of each item may vary across different markets. Specific markets may not offer certain items, and even the items available for purchase might be limited in quantity. It is important to note that every item must be sold in at least one market, otherwise the problem is impossible. Visiting a greater number of markets tends to result in increased traveling costs and reduced purchasing costs, as items can be acquired where they are sold cheapest. Conversely, visiting fewer markets tends to decrease the traveling costs at the expense of increased purchasing costs. Feasible TPP solutions come in the form of a route that covers the item list, meaning the demand for every item can be fulfilled by visiting the respective subset of markets. To efficiently acquire all items is to minimize the total expenditure by balancing the traveling and purchasing costs sustained along the route.

Let M be the node set composed by the depot (0) and the markets. Let the arc set be $A = \{(i, j) : i, j \in M \wedge i \neq j\}$ and c_{ij} the traveling cost associated with arc $(i, j) \in A$. The problem can be modeled in a complete and directed graph $G = (M, A)$. Consider an item set K and a subset of markets $M(k) \subseteq M \setminus \{0\}$ where item $k \in K$ is available for purchase at the price of p_{ki}, such that $i \in M(k)$.

This study addresses the unrestricted asymmetric TPP variant (UATPP). The asymmetric property implies a potential difference in traveling costs in relation to the traversal direction between two locations. The unrestricted property implies that the demand for each item can be fulfilled by visiting any single market that sells the item. This is equivalent to assuming each item has unitary demand with the respective price adjustments. A single vehicle and depot are considered and each pair of nodes is connected by two opposite arcs, enabling the purchaser to traverse between any two locations, since G is a complete graph. Every traveling cost for each arc and every item cost for each market that sells the item are known.

The following is an example of a UATPP instance with four markets ($M = \{0, 1, 2, 3, 4\}$) and three items ($K = \{1, 2, 3\}$). The traveling and purchasing costs are represented in Figures 2.1 and 2.2, respectively. The dash symbol (-) in Figure 2.2 means the item is not available in the market.

	Depot	*Market* 1	*Market* 2	*Market* 3	*Market* 4
Depot	-	15	30	18	16
Market 1	30	-	19	24	27
Market 2	24	30	-	27	20
Market 3	24	18	15	-	24
Market 4	19	15	23	26	-

Figure 2.1: Asymmetric traveling cost matrix.

	Item 1	*Item* 2	*Item* 3
Market 1	-	-	24
Market 2	-	21	26
Market 3	23	-	20
Market 4	29	30	28

Figure 2.2: Purchasing cost matrix.

All items can be acquired by solely visiting *Market* 4, resulting in minimal traveling expenses of $c_{04} + c_{40} = 16 + 19 = 35$ and a purchasing expense of $p_{14} + p_{24} + p_{34} = 29 + 30 + 28 = 87$. The total cost associated with this route is the sum of the traveling and purchasing costs, resulting in $35 + 87 = 122$. Another option is to visit *Market* 2 and *Market* 3, as they cover the item list. Since *Item* 3 is available in both markets, the cheapest alternative is selected. The traveling cost associated with this route is $c_{02} + c_{23} + c_{30} = 30 + 27 + 24 = 81$ and the purchasing cost is $p_{13} + p_{22} + p_{33} = 23 + 21 + 20 = 64$, resulting in a total cost of $81 + 64 = 145$. Figure 2.3 portrays the feasible routing solutions mentioned previously for this instance. Conversely, Figure 2.4 illustrates an unfeasible routing solution, as *Item* 2 can not be acquired at the visited markets.

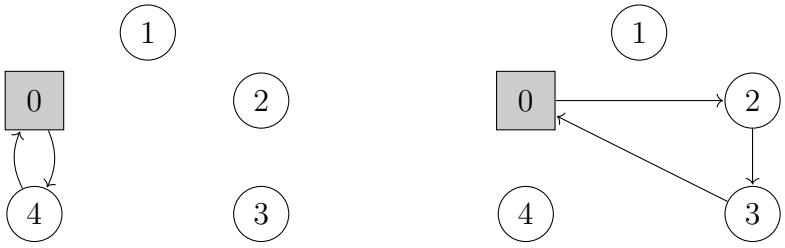

Figure 2.3: Examples of feasible routing solutions.

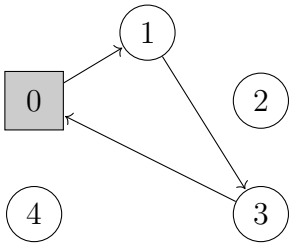

Figure 2.4: Example of an unfeasible routing solution.

2.3 Mathematical Formulation

The UATPP can be formulated in mixed integer linear programming to attain the optimal solutions for the benchmark instances under study. A single-commodity flow formulation is presented with three sets of decision variables. The single-commodity flow formulation is compact, meaning the number of constraints does not grow exponentially as instances increase in complexity. The sets of variables are:

f_{ij}**:** flow that traverses arc $(i, j) \in A$, which represents the number of items that still have to be acquired.

$$x_{ij} = \begin{cases} 1, & \text{if arc } (i,j) \in A \text{ is traversed.} \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ki} = \begin{cases} 1, & \text{if item } k \in K \text{ is purchased at market } i \in M(k). \\ 0, & \text{otherwise.} \end{cases}$$

$$\textit{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{k\in K}\sum_{i\in M(k)} p_{ki}y_{ki} \tag{1}$$

Subject to:

$$\sum_{i\in M(k)} y_{ki} = 1 \qquad \forall\, k \in K \tag{2}$$

$$\sum_{j\in M} x_{ij} \leq 1 \qquad \forall\, i \in M \tag{3}$$

$$\sum_{j\in M} x_{ji} - \sum_{j\in M} x_{ij} = 0 \qquad \forall\, i \in M \tag{4}$$

$$\sum_{j\in M\setminus\{0\}} f_{0j} = |K| \tag{5}$$

$$\sum_{j\in M} f_{ji} = \sum_{j\in M} f_{ij} + \sum_{k\in K} y_{ki} \qquad \forall\, i \in M \setminus \{0\} \tag{6}$$

$$0 \leq f_{ij} \leq |K| x_{ij} \qquad \forall\, (i,j) \in A \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{8}$$

$$y_{ki} \in \{0,1\} \qquad \forall\, k \in K,\ i \in M(k) \tag{9}$$

The objective (1) is to minimize the total cost comprised by the traveling and purchasing costs. Constraints (2) ensure each item is purchased once. Constraints (3) guarantee that each market is visited no more than once. Constraints (4) guarantee that, for each node, the number of arcs that enter the node equals the number of arcs that leave it. Constraint (5) ensures the amount of missing items at the beginning of the route is $|K|$. Constraints (6), in conjunction with (7), ensures items are only purchased at visited markets. Moreover, constraints (6) prevent the existence of sub-routes since every visited market absorbs a portion of the single commodity that stemmed from the depot in (5). In order for each flow variable f_{ij} to take a positive value, the associated arc must be traversed due to constraints (7). Although the flow variables represent integer values by concept, their integrality is ensured by the other constraints of the model. Constraints (8) and (9) define the domain of x_{ij} and y_{ki}, respectively.

Chapter 3

Methodology

This study uses the metaheuristic ILS complemented with a route configuration procedure to address the UATPP. Section 3.1 introduces the metaheuristic and the corresponding pseudocode. Section 3.2 presents a constructive heuristic that creates a feasible solution. Section 3.3 presents the neighborhoods considered for the route configuration procedure in Section 3.4 and for the local search in Section 3.5. Section 3.6 presents the perturbation operators, followed by a diversity control technique in Section 3.7, and the proposed algorithm in Section 3.8.

3.1 Iterated Local Search

The ILS pseudocode for minimization problems is presented in Algorithm 3.1. A feasible solution is initially created, followed by a local search procedure to reach a local optimum. Each iteration of the ILS encompasses two main components. The first component constitutes a perturbation involving a transformation to escape from local optima. The second component involves a local search to find the new local optimum and hopefully an overall improvement. These two steps are repeated until the termination criterion is met. Ultimately, the best-found solution x with the lowest objective function value $Z(x)$ is returned.

Algorithm 3.1 Pseudocode of ILS.

```
1: Create a feasible solution x
2: x ← LocalSearch(x)
3: incumbent ← x
4: while termination criterion is not met do
5:    x ← Perturbation(x)
6:    x ← LocalSearch(x)
7:    if Z(x) < Z(incumbent) then
8:       incumbent ← x
9:    end if
10: end while
11: return incumbent
```

3.2 Constructive Heuristic

The starting point for the metaheuristic requires a feasible solution. Therefore, a constructive heuristic is introduced in Algorithm 3.2, aiming to generate a route that covers the item list. If the list is covered, every item can be purchased in the visited markets. Hence, the feasible solution can be deduced since the items are purchased where they are sold cheapest. In each iteration, the market selection is based on the amount of items each market can cover and the respective average price. Subsequently, the markets are arranged by a nearest neighbor heuristic. The following notation is considered:

- P: set of unvisited markets.
- S: set of markets to include in the route.
- I: items unavailable for purchase in the selected markets S.
- r: route solution.
- w_i: number of items in I that can be purchased at market $i \in P$.
- a_i: average cost of the items in I that are available at market $i \in P$, that is,

$$a_i = \frac{\sum_{[k \in I \,:\, i \in M(k)]} p_{ki}}{w_i}.$$

Algorithm 3.2 ConstructiveHeuristic().

1: $P \leftarrow M \setminus \{0\}$
2: $S \leftarrow \emptyset$
3: $I \leftarrow K$
4: **while** $I \neq \emptyset$ **do**
5: **for** i **in** P **do**
6: Calculate w_i and a_i
7: **end for**
8: $w_{\max} \leftarrow \max\{w_i : i \in P\}$
9: $a_{\min} \leftarrow \min\{a_i : i \in P \wedge w_i = w_{\max}\}$
10: Compute $i^* \in P$ such that $a_{i^*} = a_{\min}$ and $w_{i^*} = w_{\max}$
11: $S \leftarrow S \cup \{i^*\}$
12: $P \leftarrow P \setminus \{i^*\}$
13: **for** k **in** I **do**
14: **if** $i^* \in M(k)$ **then**
15: $I \leftarrow I \setminus \{k\}$
16: **end if**
17: **end for**
18: **end while**
19: Insert the depot (0) at the start of route r
20: $v \leftarrow 0$
21: **while** $S \neq \emptyset$ **do**
22: Compute market $i^* \in S$ such that $c_{vi^*} = \min\{c_{vj} : j \in S\}$
23: Insert i^* at the end of route r
24: $S \leftarrow S \setminus \{i^*\}$
25: $v \leftarrow i^*$
26: **end while**
27: Insert the depot (0) at the end of route r
28: Purchase the items where they are cheapest within the visited markets and construct feasible solution x
29: **return** x

The first *while* statement (lines 4 to 18) selects a subset of markets S to cover the item list. In each iteration, the selection of markets is primarily determined by their ability to cover the remaining items in I, in order to obtain feasibility. Amongst the markets that can cover the highest number of missing items ($w_{\max}$), the one with the lowest average purchasing cost

($a_{\min}$) is selected. Since the subset of markets S covers the item list, all that is required to obtain a feasible solution is to establish the order by which the markets are visited, which is done with the known nearest neighbor heuristic for the TSP (lines 19 to 27).

3.3 Neighborhoods

The quality of the solution may be improved by searching different neighborhoods. This section presents the neighborhoods considered by the route configuration procedure introduced in Section 3.4 and by the local search introduced in Section 3.5. Given a feasible solution x, the following neighborhoods are considered:

- $\mathcal{N}_{\text{add}}(x) = \{x' \text{ feasible} : x' \text{ can be obtained from } x \text{ by adding an unvisited market to the route}\} \cup \{x\}$.
- $\mathcal{N}_{\text{drop}}(x) = \{x' \text{ feasible} : x' \text{ can be obtained from } x \text{ by removing a visited market from the route}\} \cup \{x\}$.
- $\mathcal{N}_{\text{exchange}}(x) = \{x' \text{ feasible} : x' \text{ can be obtained from } x \text{ by exchanging a visited market in the route with an unvisited market}\} \cup \{x\}$.
- $\mathcal{N}_{\text{move}}(x) = \{x' \text{ feasible} : x' \text{ can be obtained from } x \text{ by moving a visited market to another position in the route}\} \cup \{x\}$.
- $\mathcal{N}_{\text{switch}}(x) = \{x' \text{ feasible} : x' \text{ can be obtained from } x \text{ by switching the positions of two visited markets in the route}\} \cup \{x\}$.

Items are always acquired where they are cheapest within the visited markets. Hence, if the subset of visited markets is altered by a given neighborhood search, the purchasing decisions are adjusted accordingly. Route examples of neighbor solutions are presented next. Let m be the number of markets in solution x. The size of neighborhood $\mathcal{N}_{\text{add}}(x)$ is $O([|M\backslash\{0\}|-m]\times[m+1])$, as there are $|M\backslash\{0\}|-m$ unvisited markets that can be added across $m+1$ different positions in the route. Each unvisited market may sell up to $|K|$ items, potentially improving the purchasing cost of every one of those items. Figure 3.1 displays the routes of a solution x and a neighbor solution $x' \in \mathcal{N}_{\text{add}}(x)$.

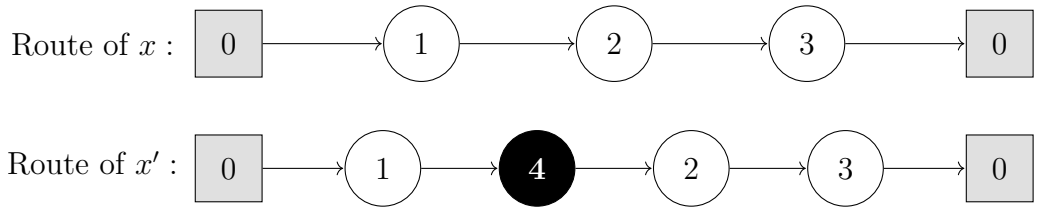

Figure 3.1: Route example of a neighbor solution x' in $\mathcal{N}_{\text{add}}(x)$.

The size of neighborhood $\mathcal{N}_{\text{drop}}(x)$ is $\mathrm{O}(m)$, as there are m markets that may be dropped and each choice may increase the purchasing cost of up to $|K|$ items at the expense of decreased traveling costs. Figure 3.2 displays the routes of a solution x and a neighbor solution $x' \in \mathcal{N}_{\text{drop}}(x)$.

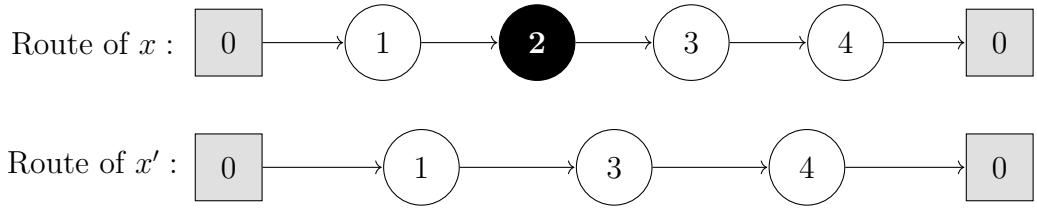

Figure 3.2: Route example of a neighbor solution x' in $\mathcal{N}_{\text{drop}}(x)$.

The size of neighborhood $\mathcal{N}_{\text{exchange}}(x)$ is $\mathrm{O}([|M \setminus \{0\}| - m] \times m)$, as there are $|M \setminus \{0\}| - m$ unvisited markets to compare with each one of the m markets in the route. Each exchange may update the purchasing cost of up to $|K|$ items. Figure 3.3 displays the routes of a solution x and a neighbor solution $x' \in \mathcal{N}_{\text{exchange}}(x)$.

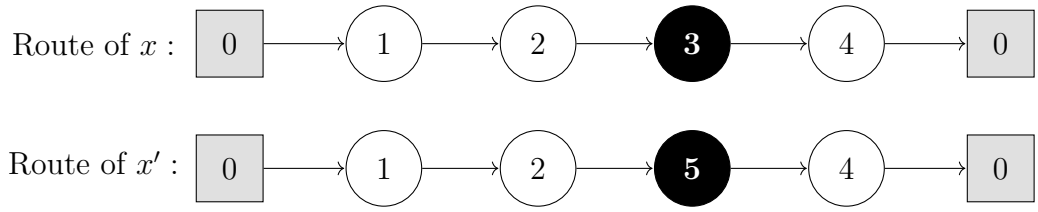

Figure 3.3: Route example of a neighbor solution x' in $\mathcal{N}_{\text{exchange}}(x)$.

The size of neighborhood $\mathcal{N}_{\text{move}}(x)$ is $\mathrm{O}(m \times [m-1])$, as each of the m markets may be moved to one of the $m-1$ remaining positions. Unlike in the previous neighborhoods, the subset of visited markets remains equal. Figure 3.4 displays the routes of a solution x and a neighbor solution $x' \in \mathcal{N}_{\text{move}}(x)$.

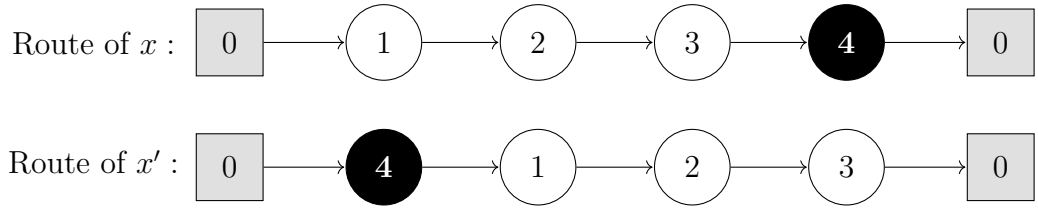

Figure 3.4: Route example of a neighbor solution x' in $\mathcal{N}_{\text{move}}(x)$.

The size of neighborhood $\mathcal{N}_{\text{switch}}(x)$ is $\mathrm{O}(\binom{m}{2})$, as there are $\binom{m}{2}$ unique pairs of markets in the route to consider switching positions. Similarly to the previous neighborhood, the subset of visited markets remains the same. Figure 3.5 displays the routes of a solution x and a neighbor solution $x' \in \mathcal{N}_{\text{switch}}(x)$.

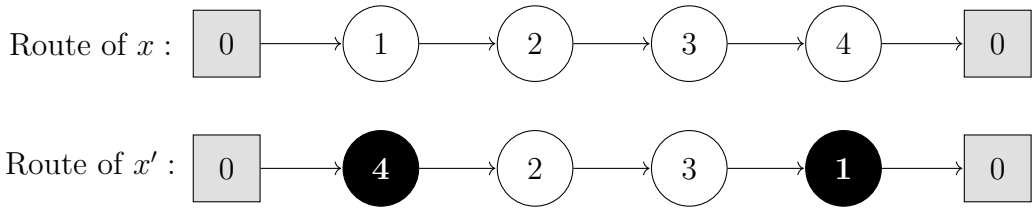

Figure 3.5: Route example of a neighbor solution x' in $\mathcal{N}_{\text{switch}}(x)$.

Algorithm 3.3 presents the *Explore* algorithm, which considers a solution x and successively searches a neighborhood $\mathcal{N}$ until either a certain maximum number of searches δ is reached or no improvement is found. Considering $\delta = +\infty$, a neighborhood is searched until the local optimum is reached. In each search of $\mathcal{N}(x)$ (line 2), the solution with the best improvement is selected. For instance, a market may be moved to different positions in the route, but the best position is always chosen as long as it yields improvement. Section A.1 provides the search algorithms for each defined neighborhood.

Algorithm 3.3 Explore(x, $\mathcal{N}$, δ).

Parameters: $\mathcal{N}$, δ

Require: initial solution x

1: **for** $i = 1$ **to** δ **do**
2: Search $\mathcal{N}(x)$ and obtain x'.
3: **if** $x = x'$ **then**
4: break
5: **end if**
6: $x \leftarrow x'$
7: **end for**
8: **return** x

3.4 Route Configuration

Following a constructive heuristic or a perturbation, the solution is subject to a route configuration procedure. The main purpose of this phase is to adjust the average route size throughout the algorithm, providing the local search with the appropriate number of markets. This is achieved by limiting the amount of searches in $\mathcal{N}_{\text{add}}(x)$ and $\mathcal{N}_{\text{drop}}(x)$. Furthermore, neighborhood $\mathcal{N}_{\text{exchange}}(x)$ is also subject to a search limit, as fully exploring this neighborhood may have negative impacts on the main algorithm in terms of solution quality and computational times, as shall be seen in Subsection 4.1.5.

The procedure presented in Algorithm 3.4 intends to configure the route in each iteration of the ILS algorithm. Unlike in the local search, these neighborhoods are not necessarily searched until a local optimum is reached, as the route may deviate from the appropriate size and the optimum subset of markets. For instance, fully exploring neighborhood $\mathcal{N}_{\text{add}}(x)$ increases the route size on average throughout the algorithm, which may hinder the discovery of the optimal solution if it is of small size. Hence, let δ_{add}, δ_{drop}, δ_{exchange} be the maximum number of searches in $\mathcal{N}_{\text{add}}(x)$, $\mathcal{N}_{\text{drop}}(x)$, and $\mathcal{N}_{\text{exchange}}(x)$ respectively. Neighborhood $\mathcal{N}_{\text{add}}(x)$ is explored first since inserting new markets may reveal other markets in the route where the purchasing cost benefit of visiting them is overshadowed by the traveling cost increase, enabling the search in $\mathcal{N}_{\text{drop}}(x)$ to produced better results. After the markets have been removed, which implies a lower route size, the neighborhood $\mathcal{N}_{\text{exchange}}(x)$ is searched. Due to the complexity of searching $\mathcal{N}_{\text{exchange}}(x)$, shorter route sizes significantly improve the computational time of the algorithm. In the end, neighborhood $\mathcal{N}_{\text{drop}}(x)$ is searched until the local optimum is reached to remove markets not necessary to obtain a feasible solution, stabilizing the route size and subset of visited markets for the local search.

Algorithm 3.4 RouteConfiguration(x, δ_{add}, δ_{drop}, δ_{exchange}).

Parameters: δ_{add}, δ_{drop}, δ_{exchange}

Require: initial solution x

1: $x \leftarrow \text{Explore}(x,\ \mathcal{N}_{\text{add}}(x),\ \delta_{\text{add}})$
2: $x \leftarrow \text{Explore}(x,\ \mathcal{N}_{\text{drop}}(x),\ \delta_{\text{drop}})$
3: $x \leftarrow \text{Explore}(x,\ \mathcal{N}_{\text{exchange}}(x),\ \delta_{\text{exchange}})$
4: $x \leftarrow \text{Explore}(x,\ \mathcal{N}_{\text{drop}}(x),\ +\infty)$
5: **return** x

3.5 Local Search

The local search determines a local optimum. Different neighborhoods may be searched, and in distinct sequences. It is important to note that only improved feasible solutions are accepted in this component of the algorithm. These searches are implemented successively until no further improvement is found.

Algorithm 3.5 presents the proposed local search. Different search sequences of the $\mathcal{N}_{\text{move}}(x)$ and $\mathcal{N}_{\text{switch}}(x)$ neighborhoods are considered. Each one of these sequences is repeated until the local optimum is reached. First, x_1 is subject to an extensive search in the $\mathcal{N}_{\text{move}}(x)$ neighborhood, followed by an extensive search in $\mathcal{N}_{\text{switch}}(x)$. Conversely, x_2 considers exploring $\mathcal{N}_{\text{switch}}(x)$ first, which may yield a different outcome. Finally, x_3 is subject to a single search in each neighborhood, repeating this process until neither neighborhood can be further explored. Ultimately, the solution $x_k \in \{x_1, x_2, x_3\}$ with the lowest objective function value $Z(x_k)$ is chosen.

Algorithm 3.5 LocalSearch(x).

Require: initial solution x

$x_1 \leftarrow x_2 \leftarrow x_3 \leftarrow x$
$x_1 \leftarrow \text{Explore}(x_1,\ \mathcal{N}_{\text{move}}(x_1),\ +\infty)$
$x_1 \leftarrow \text{Explore}(x_1,\ \mathcal{N}_{\text{switch}}(x_1),\ +\infty)$
$x_2 \leftarrow \text{Explore}(x_2,\ \mathcal{N}_{\text{switch}}(x_2),\ +\infty)$
$x_2 \leftarrow \text{Explore}(x_2,\ \mathcal{N}_{\text{move}}(x_2),\ +\infty)$
repeat
    $x_3 \leftarrow \text{Explore}(x_3,\ \mathcal{N}_{\text{switch}}(x_3),\ 1)$
    $x_3 \leftarrow \text{Explore}(x_3,\ \mathcal{N}_{\text{move}}(x_3),\ 1)$
until no improvement is found
Select $x_k \in \{x_1, x_2, x_3\}$ such that $Z(x_k)$ is the minimum value obtained
return x_k

3.6 Perturbation

The local search eventually comes to a stop, as a local optimum was reached. However, the global optimum may remain elusive, implying the need to explore new areas in the solution space. Bearing this in mind, a perturbation is applied to escape the local optimum, promoting diversity by transforming the solution under specific criteria. In the perturbation procedure, two operators are considered. Firstly, a destroy operator, presented in Subsection 3.6.1, removes markets from the solution, likely turning it unfeasible. Subsequently, a repair operator, presented in Subsection 3.6.2, restores feasibility by adding markets to the solution in order to cover the item list.

3.6.1 Destroy Operator

The destroy operator randomly removes markets from the solution, as presented in Algorithm 3.6. All markets have the same probability of being removed following a uniform distribution. The amount of markets to remove is a percentage α of the route size m, rounded up. For instance, considering $\alpha = 20\%$ on a route with ten markets will result in the removal of two markets. Rounding up $\alpha \times m$ ensures that at least one market is removed in each perturbation. Items that become unavailable for purchase in the remaining subset of markets are referred to as missing items.

Algorithm 3.6 Destroy(x, α).

Parameters: α

Require: initial solution x

1: $k = \text{RoundUp}(\alpha \times m)$
2: Randomly remove k markets from the route in solution x
3: **return** x

3.6.2 Repair Operator

The repair operator restores feasibility by adding markets to the destroyed route. In the rare event where the item list remains covered after the destroy operator, the repair operator is not applied. Exploring new areas of the feasible region involves the addition of markets that lead to a distinct subset.

Thus, a new metric Δ_{ij} is introduced, resembling the concept implemented in the work of Bernardino and Paias (2024).

- Δ_{ij}: number of times that nodes $i, j \in M$ were in the same route together, following a perturbation or constructive heuristic. To be noted that $\Delta_{ij} = \Delta_{ji}$, with $i, j \in M : i \neq j$.
- $G_{iR} = \sum_{j \in R} \Delta_{ij}$, where R is the subset of nodes in the route.

Using this metric promotes diversity by selecting market i such that G_{iR} is minimized, which is the market that has shared the same route with the nodes in R the least amount of times. Algorithm 3.7 presents the repair operator which selects the markets based on the lowest value of G_{iR} across the unvisited markets that sell at least one missing item. This last requirement is essential to ensure feasibility is being restored. If multiple markets possess the minimum value in G_{iR}, the market that sells the most missing items is selected. Following a perturbation or constructive heuristic, the metric Δ_{ij} is incremented for each node pair (i, j) in R such that $i \neq j$. In every repair iteration, the selected market is inserted at the best position. Considering the depot in the metric is essential, as a route may have no markets after the destroy operator.

Algorithm 3.7 Repair(x).

Require: initial solution x
1: $Q \leftarrow M \setminus R$
2: **while** item list is not covered **do**
3: $L \leftarrow$ markets in Q that minimize G_{iR}
4: **if** no market in L sells missing items **then**
5: $Q \leftarrow Q \setminus L$
6: **else**
7: $i \leftarrow$ market in L with the highest amount of missing items
8: Insert market i at the best position within the route of solution x
9: $Q \leftarrow Q \setminus \{i\}$
10: **end if**
11: **end while**
12: Purchase the items where they are cheapest within the visited markets and construct feasible solution x
13: **return** x

3.7 Diversity Control

The perturbation component of the algorithm has a partial effect on the solution. Alternatively, generating an entirely new solution can reinforce diversity to a greater degree. This may be prudent to avoid extensively exploring solution regions that result in no improvement in the incumbent solution. After a given number of consecutive iterations $\lambda_{\max}$ without improving the best-known solution, a constructive heuristic is applied as presented in Algorithm 3.8, based on the previously mentioned diversity metric Δ_{ij}.

Algorithm 3.8 DiversityConstructiveHeuristic().

$x \leftarrow \emptyset$
Insert the depot (0) at the start and at the end of the route in x
$Q \leftarrow M \setminus \{0\}$
while item list is not covered **do**
  $L \leftarrow$ markets in Q that minimize G_{iR}
  if no market in L sells missing items **then**
    $Q \leftarrow Q \setminus L$
  else
    $i \leftarrow$ market in L with the highest amount of missing items
    Insert market i at the best position within the route of solution x
    $Q \leftarrow Q \setminus \{i\}$
  end if
end while
Purchase the items where they are cheapest within the visited markets and construct feasible solution x
return x

3.8 Iterated Local Search with Route Configuration

The optimal route possesses certain characteristics, including a specific dimension, a distinct subset of markets, and a sequence in which to visit them. The proposed algorithm is based on two main steps. Firstly, it aims to adjust the route size and respective subset of markets with the route configuration

procedure. Secondly, the local search arranges the remaining subset to reach the local optimum.

Algorithm 3.9 presents the Iterated Local Search with Route Configuration (ILS-RC). An initial solution x is created with the constructive heuristic presented in Algorithm 3.2. The route configuration procedure presented in Algorithm 3.4 adjusts the route size and respective subset of visited markets. Subsequently, the local search presented in Algorithm 3.5 reaches the local optimum. In each iteration of the ILS-RC, the perturbation presented in Section 3.6 is applied to escape the local optimum, using the destroy and repair operators presented in Algorithms 3.6 and 3.7, respectively. However,

Algorithm 3.9 ILS-RC($k_{\max},\ \lambda_{\max},\ \delta_{\text{add}},\ \delta_{\text{drop}},\ \delta_{\text{exchange}},\ \alpha$).

Parameters: $k_{\max},\ \lambda_{\max},\ \delta_{\text{add}},\ \delta_{\text{drop}},\ \delta_{\text{exchange}},\ \alpha$

$x \leftarrow$ ConstructiveHeuristic()
$x \leftarrow$ RouteConfiguration($x,\ \delta_{\text{add}},\ \delta_{\text{drop}},\ \delta_{\text{exchange}}$)
$x \leftarrow$ LocalSearch(x)
incumbent $\leftarrow x$
$z = Z(x)$
$\lambda = 0$
for $k = 1$ **to** $k_{\max}$ **do**
  if $\lambda = \lambda_{\max}$ **then**
    $x \leftarrow$ DiversityConstructiveHeuristic()
    $\lambda = 0$
  else
    $x \leftarrow$ Repair(Destroy($x,\ \alpha$))
  end if
  $x \leftarrow$ RouteConfiguration($x,\ \delta_{\text{add}},\ \delta_{\text{drop}},\ \delta_{\text{exchange}}$)
  $x \leftarrow$ LocalSearch(x)
  if $Z(x) < z$ **then**
    incumbent $\leftarrow x$
    $z = Z(x)$
    $\lambda = 0$
  else
    $\lambda = \lambda + 1$
  end if
end for
return incumbent

if a certain number of consecutive iterations without an overall improvement $\lambda_{\max}$ is reached, the diversity constructive heuristic presented in Algorithm 3.8 is applied instead. The route configuration and subsequent local search are applied to the perturbed solution, reaching a new local optimum and hopefully an overall improvement. This process is repeated until the termination criterion is met, given by the maximum number of iterations $k_{\max}$. Ultimately, the best-found solution with the lowest objective function value $Z(x)$ is returned, providing an approximation to the global optimum value.

Chapter 4

Computational Results

This chapter presents the computational results obtained by applying the ILS-RC in benchmark instances referenced in the literature, which are available at https://webpages.ull.es/users/jriera/TPP.htm. The main focus of this study is the Class 6 instances proposed by Singh and van Oudheusden (1997), which are asymmetric. Results are compared with those of Cuellar-Usaquén et al. (2023), the article reporting the best results to our knowledge. Additional experiments are executed for the Class 1 and Class 3 symmetric instances, available on the same website.

To measure the performance of the ILS-RC, the gap formula (10) is used for instances that have been solved to optimality by other authors. Regarding instances that have not been solved to optimality, the performance is measured by comparing the solution value obtained by the ILS-RC with the best-known upper bounds reported by Cuellar-Usaquén et al. (2023), Goldbarg et al. (2009), and Bontoux and Feillet (2008).

$$\text{Gap}(\%) = 100 \times \frac{Z(x) - Z(x^*)}{Z(x^*)} \tag{10}$$

Section 4.1 provides an analysis to each parameter considered in the ILS-RC, followed by an analysis of the randomized features in Section 4.2. The comparison details are presented in Section 4.3 regarding the TPP benchmark instances addressed by the previously mentioned works. Section 4.4 presents the Wilcoxon signed-rank tests between the ILS-RC and the best results in the literature. The detailed results of the computational experiments are available online at https://www.dropbox.com/scl/fi/n2ti0cu626g3hycv91kcf/ILS_RC_appendix.pdf?rlkey=af93s5c6m7ii4w9rod9rqybqu&dl=0.

4.1 Parameter Analysis

In order to improve performance, the ILS-RC was subject to various simulations considering different parameter values. The three classes of instances possess different characteristics. Hence, adjusting the parameters to each class separately may be prudent to provide a better fit.

This section provides a *ceteris paribus* analysis of each parameter in the ILS-RC in relation to the best combination found during the simulations. To highlight the impact of each parameter, only a subset of the largest Class 6 instances was considered for this analysis, as the ILS-RC reaches the optima in instances of smaller size regardless of the assigned parameter values. More precisely, five instances with 300 nodes and 200 items were considered, and each instance was executed five times for each parameter value to stabilize the randomized features associated with the ILS-RC.

The following subsections present the impact of the parameters, both in terms of solution quality and computational time. Subsection 4.1.1 addresses the number of iterations $k_{\max}$, followed by the analysis of the $\lambda_{\max}$ parameter in Subsection 4.1.2. Subsections 4.1.3, 4.1.4, and 4.1.5 present the analysis for the δ_{add}, δ_{drop}, and δ_{exchange}, respectively. Finally, the destroy percentage α is analyzed in Subsection 4.1.6. Each parameter is analyzed individually, while the rest remain fixed on the values summarized in Table 4.1 under Class 6, which is presented in Subsection 4.1.7. This last subsection also summarizes the parameters used for the Class 1 and Class 3 instances.

4.1.1 Number of Iterations $k_{\max}$

As the number of iterations increases, the solution quality tends to improve at the expense of increased computational time. However, after a certain point, the quality stops improving significantly while the computational time keeps increasing. Figures 4.1 and 4.2 present the average gaps and computational times, respectively, provided by running the ILS-RC with different values of $k_{\max}$, across the previously mentioned subset of instances. All other parameters remained unchanged, as presented in Table 4.1. Although running 10000 iterations results in the best average gap, the corresponding computational time restrains the ILS-RC in terms of efficiency. Conversely, considering 5000 iterations provides a noticeable time improvement at minimal solution quality expense (0.006%). Based on these results, $k_{\max}$ was set to 5000.

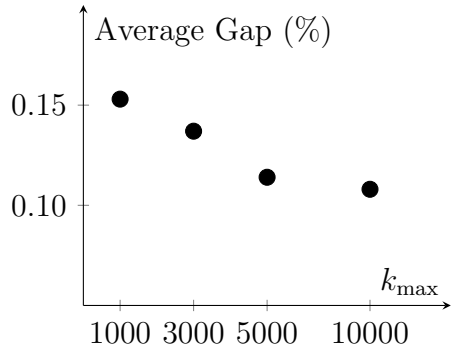

Figure 4.1: Average gap (%) for different values of $k_{\max}$.

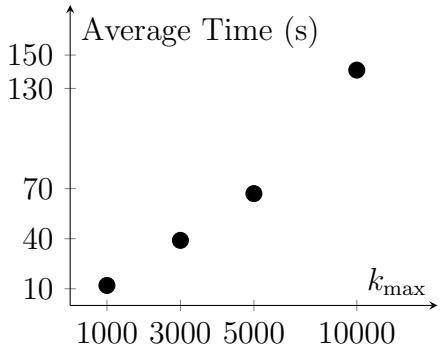

Figure 4.2: Average computational time for different values of $k_{\max}$.

4.1.2 Diversity Control $\lambda_{\max}$

Figures 4.3 and 4.4 present the average gaps and computational times, respectively, that resulted from considering different values of $\lambda_{\max}$. Although this parameter does not significantly impact the time efficiency of the ILS-RC, an improvement in the solution quality occurs when $\lambda_{\max} = 500$, implying that the diversity constructive heuristic provides benefit. Conversely, the average gaps tend to increase when $\lambda_{\max}$ is decreased below this point, as this may generate a new solution too early, not allowing the algorithm to explore a series of perturbed solutions. Based on these results, $\lambda_{\max}$ was set to 500.

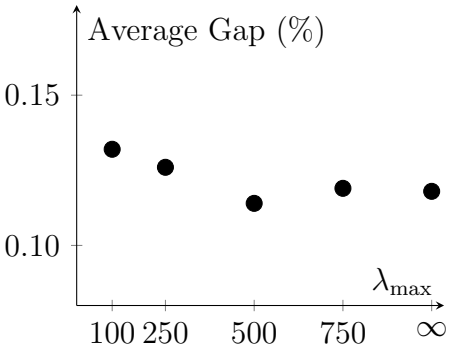

Figure 4.3: Average gap (%) for different values of $\lambda_{\max}$.

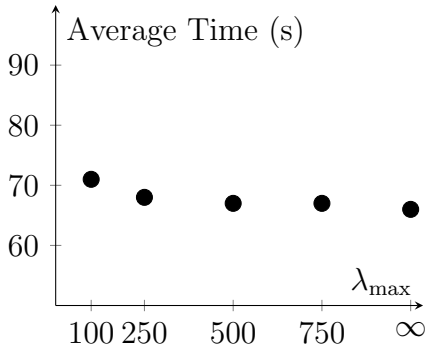

Figure 4.4: Average computational time for different values of $\lambda_{\max}$.

4.1.3 Search Limit δ_{add}

Figures 4.5 and 4.6 present the average gaps and computational times, respectively, that resulted from considering different values of δ_{add}. No significant improvement in the solution quality occurs after a maximum of two searches in this neighborhood. Hence, a limit is worth imposing, as the computational times also tend to increase with every increment of δ_{add}. This is because the average route size produced throughout the ILS-RC is larger, resulting in increased complexity in all the neighborhood searches, as mentioned in Section 3.3. Based on these results, the parameter was set to $\delta_{\text{add}} = 2$.

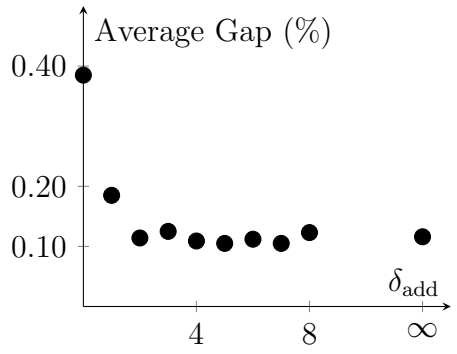

Figure 4.5: Average gap (%) for different values of δ_{add}.

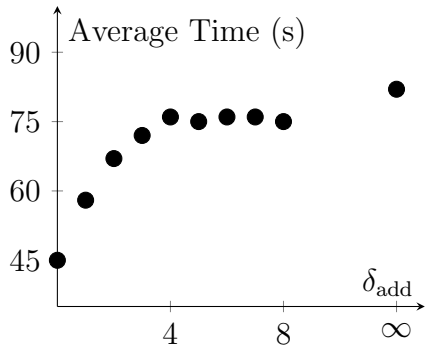

Figure 4.6: Average computational time for different values of δ_{add}.

4.1.4 Search Limit δ_{drop}

Figures 4.7 and 4.8 present the average gaps and computational times, respectively, that resulted from considering different values of δ_{drop}. The lowest average gaps occur after a maximum of two and four searches. Considering no searches has a significant impact on the computational time, as the average route size produced throughout the ILS-RC tends to grow considerably. Conversely, fully exploring this neighborhood decreases the solution quality, as the route size may be smaller on average, hindering the discovery of optimal solutions with a larger number of markets. The best gaps result from finding a balanced parameter value, enabling the ILS-RC to reach optimal routes of small and large sizes. Based on these results, the parameter was set to $\delta_{\text{drop}} = 4$, although a tie appears to happen with $\delta_{\text{drop}} = 2$.

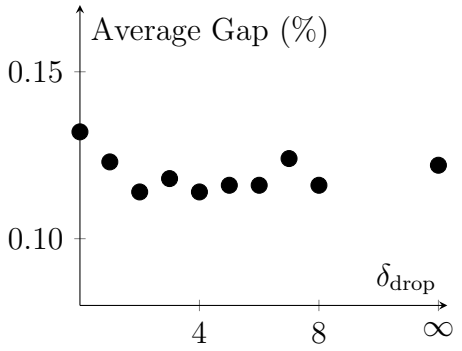

Figure 4.7: Average gap (%) for different values of δ_{drop}.

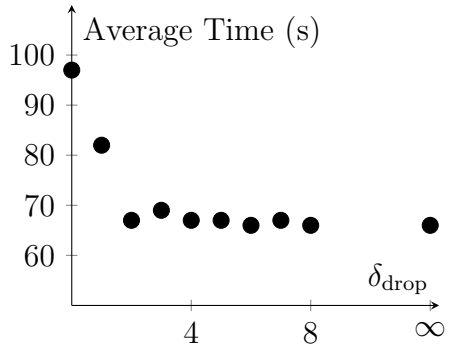

Figure 4.8: Average computational time for different values of δ_{drop}.

4.1.5 Search Limit δ_{exchange}

Figure 4.9 displays the average gaps that resulted by considering different values of δ_{exchange}. No significant improvement in the solution quality occurs after a maximum of two searches. Moreover, the corresponding average times presented in Figure 4.10 tend to grow considerably. Hence, limiting the number of searches in $\mathcal{N}_{\text{exchange}}(x)$ is prudent in order to promote solution quality at minimal time expense. The average gaps may stop improving past a certain point since the best-found subset of markets is reached in every run of the ILS-RC, despite different values of δ_{exchange}. Based on these results, the parameter was set to $\delta_{\text{exchange}} = 2$.

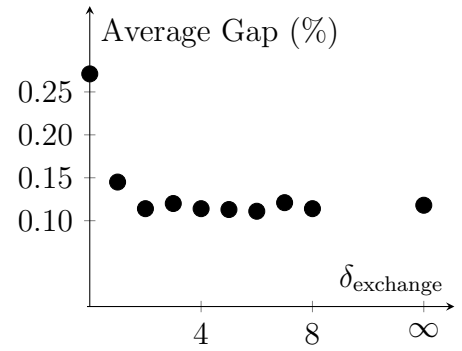

Figure 4.9: Average gap (%) for different values of δ_{exchange}.

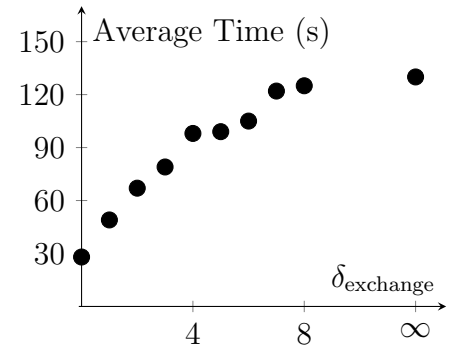

Figure 4.10: Average computational time for different values of δ_{exchange}.

4.1.6 Destroy Percentage α

Figures 4.11 and 4.12 present the average gaps and computational times, respectively, resulting from considering different values of α. No significant improvement in the solution quality occurs after $\alpha = 10\%$, since considering large percentages may hinder the exploration of routes with a similar subset of markets, preventing minor adjustments to come to fruition. The computational times tend to grow as resorting to the diversity metric Δ_{ij} usually increases the route size, since markets are inserted based on the number of times they have been joined together rather than on their ability to cover the item list. Based on these results, the parameter was set to $\alpha = 10\%$.

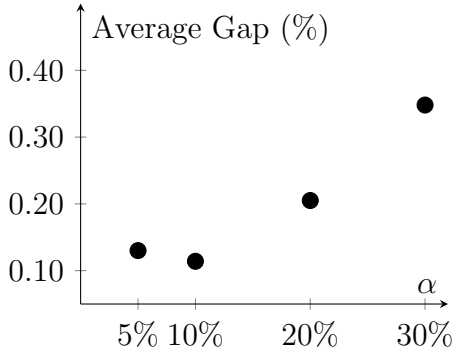

Figure 4.11: Average gap (%) for different values of α.

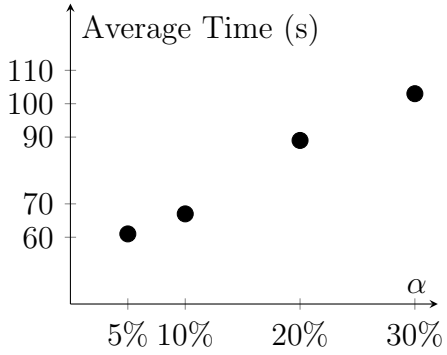

Figure 4.12: Average computational time for different values of α.

4.1.7 Summary

Table 4.1 summarizes the parameters used by the current study for each class of instances, adjusting the ILS-RC to the features of each class. The parameters were chosen based on the previous simulations. As the additional

Table 4.1: Parameters used for each set of benchmark instances.

	$k_{\max}$	$\lambda_{\max}$	δ_{add}	δ_{drop}	δ_{exchange}	α
Class 6	5000	500	2	4	2	10%
Class 1	15000	250	8	0	4	30%
Class 3	8000	500	5	3	4	5%

analyses executed for Class 1 and Class 3 instances were similar, they are omitted. Nevertheless, each parameter followed a similar behaviour in relation to the solution quality and computational time of the Class 6 analysis, merely changing the value in which the best result occurred.

4.2 Randomization Analysis

The ILS-RC contains randomized features, as the destroy operator randomly removes markets from the solution. Hence, the solution quality produced from the ILS-RC may differ in accordance with the seed used. To ascertain if the ILS-RC is consistent regardless of the seed used, a simulation was conducted on every Class 6 instance addressed in this study. A total of 30 runs were conducted, each considering a different seed to test if the ILS-RC is robust. Figure 4.13 presents the boxplots of the average gaps and computational times that resulted from this simulation. The ILS-RC is able to achieve consistent results regardless of the seed used, as the average gaps range between 0.016% and 0.024%. The differences are only noticeable when observed to three decimal places. Similarly, the average computational times are also very stable.

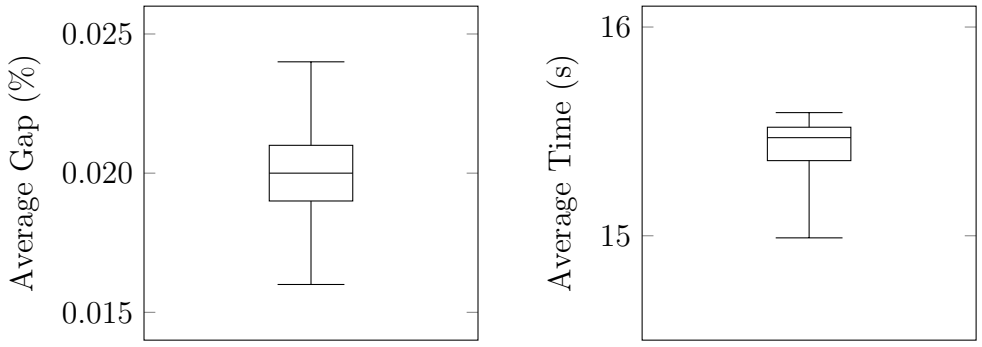

Figure 4.13: Boxplots of the average gaps and computational times for 30 different seeds.

In a more detailed perspective, Table 4.2 presents a gap summary for each considered instance. The columns display the number of nodes $|M|$, the number of items $|K|$, the instance *Id*, the minimum, maximum, range, and average gaps for each instance across the 30 runs of the ILS-RC with different seeds. The average computational time is displayed in the last column. The ILS-RC is able to reach the optimum solution at least once

in the different seeds used, except for six instances with the largest number of items ($|K| = 200$). It is important to note that the optimum value for the instance with $|M| = 300$, $|K| = 200$, and $Id = 1$, remains unknown. The gap associated with this instance was calculated using a lower bound for optimal value. Hence, the values portrayed for this instance represent an upper bound for the respective gap statistics. Beyond this specific instance, the next largest average gap is 0.15% that corresponds to the instance with $|M| = 300$, $|K| = 200$, and $Id = 5$, which also holds the largest maximum gap of 0.20%. These values are relatively small, demonstrating the effectiveness of the ILS-RC in obtaining quality solutions. All instances report a range lower than 0.18%, implying that the ILS-RC is consistent despite incorporating randomized features.

Table 4.2: Summary per instance for the 30 different seeds.

			Gap (%)				Time (s)
$\|M\|$	$\|K\|$	Id	Min	Max	Range	Average	Average
50	50	1	0.00	0.00	0.00	0.00	1
50	50	2	0.00	0.00	0.00	0.00	2
50	50	3	0.00	0.00	0.00	0.00	2
50	50	4	0.00	0.00	0.00	0.00	1
50	50	5	0.00	0.00	0.00	0.00	2
50	100	1	0.00	0.09	0.09	0.00	3
50	100	2	0.00	0.00	0.00	0.00	4
50	100	3	0.00	0.00	0.00	0.00	3
50	100	4	0.00	0.00	0.00	0.00	3
50	100	5	0.00	0.00	0.00	0.00	3
50	200	1	0.00	0.05	0.05	0.01	6
50	200	2	0.00	0.02	0.02	0.00	6
50	200	3	0.00	0.04	0.04	0.01	6
50	200	4	0.00	0.07	0.07	0.04	6
50	200	5	0.00	0.07	0.07	0.01	6
75	50	1	0.00	0.00	0.00	0.00	2
75	50	2	0.00	0.00	0.00	0.00	3

Continued on next page

Table 4.2 – continued from previous page

			Gap (%)				Time (s)
$\|M\|$	$\|K\|$	Id	Min	Max	Range	Average	Average
75	50	3	0.00	0.00	0.00	0.00	3
75	50	4	0.00	0.00	0.00	0.00	3
75	50	5	0.00	0.00	0.00	0.00	3
75	100	1	0.00	0.00	0.00	0.00	5
75	100	2	0.00	0.00	0.00	0.00	5
75	100	3	0.00	0.00	0.00	0.00	5
75	100	4	0.00	0.00	0.00	0.00	5
75	100	5	0.00	0.00	0.00	0.00	5
75	200	1	0.00	0.02	0.02	0.01	11
75	200	2	0.00	0.04	0.04	0.04	11
75	200	3	0.00	0.09	0.09	0.02	11
75	200	4	0.02	0.13	0.11	0.07	11
75	200	5	0.00	0.05	0.05	0.02	12
100	50	1	0.00	0.00	0.00	0.00	2
100	50	2	0.00	0.00	0.00	0.00	2
100	50	3	0.00	0.00	0.00	0.00	2
100	50	4	0.00	0.00	0.00	0.00	2
100	50	5	0.00	0.09	0.09	0.06	2
100	100	1	0.00	0.00	0.00	0.00	8
100	100	2	0.00	0.00	0.00	0.00	8
100	100	3	0.00	0.00	0.00	0.00	9
100	100	4	0.00	0.00	0.00	0.00	7
100	100	5	0.00	0.04	0.04	0.02	8
100	200	1	0.00	0.07	0.07	0.04	16
100	200	2	0.00	0.07	0.07	0.01	17
100	200	3	0.00	0.05	0.05	0.01	15
100	200	4	0.00	0.00	0.00	0.00	16
100	200	5	0.00	0.11	0.11	0.06	15
200	50	1	0.00	0.00	0.00	0.00	8

Continued on next page

Table 4.2 – continued from previous page

			Gap (%)				Time (s)
$\|M\|$	$\|K\|$	Id	Min	Max	Range	Average	Average
200	50	2	0.00	0.00	0.00	0.00	9
200	50	3	0.00	0.00	0.00	0.00	9
200	50	4	0.00	0.00	0.00	0.00	10
200	50	5	0.00	0.00	0.00	0.00	8
200	100	1	0.00	0.04	0.04	0.00	23
200	100	2	0.00	0.04	0.04	0.01	21
200	100	3	0.00	0.00	0.00	0.00	14
200	100	4	0.00	0.00	0.00	0.00	12
200	100	5	0.00	0.05	0.05	0.00	11
200	200	1	0.00	0.14	0.14	0.07	45
200	200	2	0.09	0.18	0.09	0.13	40
200	200	3	0.05	0.09	0.04	0.06	44
200	200	4	0.05	0.16	0.11	0.10	45
200	200	5	0.00	0.09	0.09	0.06	43
300	50	1	0.00	0.00	0.00	0.00	11
300	50	2	0.00	0.17	0.17	0.01	14
300	50	3	0.00	0.00	0.00	0.00	15
300	50	4	0.00	0.00	0.00	0.00	17
300	50	5	0.00	0.00	0.00	0.00	13
300	100	1	0.00	0.00	0.00	0.00	20
300	100	2	0.00	0.00	0.00	0.00	16
300	100	3	0.00	0.04	0.04	0.02	21
300	100	4	0.00	0.13	0.13	0.02	35
300	100	5	0.00	0.13	0.13	0.01	40
300	200	1	0.30	0.39	0.09	0.34	78
300	200	2	0.00	0.02	0.02	0.00	53
300	200	3	0.00	0.07	0.07	0.04	60
300	200	4	0.00	0.16	0.16	0.07	75
300	200	5	0.09	0.20	0.11	0.15	75

4.3 Comparison Details

Cuellar-Usaquén et al. (2023) addressed the Class 6 asymmetric instances, providing the best results in the literature to our knowledge. A GRASP-based methodology was proposed, complemented with a Filtering and Path-Relinking strategy. Two algorithms were considered, each with a different constructive procedure, namely the purchase-focused (P) and route-focused (R) procedures. Additional experiments were conducted by the same article, addressing the Class 1 and Class 3 symmetric instances. Other works that reported results for the Class 3 instances include the Transgenetic Algorithm proposed by Goldbarg et al. (2009) and the Ant Colony Algorithm introduced by Bontoux and Feillet (2008). Table 4.3 summarizes the implementation details of the articles included in the comparison. Due to differences in computing architectures, the reported times are not directly comparable. Cuellar-Usaquén et al. (2023) used the most efficient computing architecture,

Table 4.3: Summary of characteristics of the compared approaches.

Author	**Solution approach**	**Implementation details**
This work	ILS with Route Configuration	C++. 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz, Windows 10 with 8 GB RAM
Cuellar-Usaquén et al. (2023)	GRASP + Filtering\PR	C++. AMD Ryzen 7-3800X @ 3.90GHz, Windows 10 with 32 GB RAM
Goldbarg et al. (2009)	Transgenetic Algorithm	Pentium4 @ 2.80GHz, Ubuntu Linux with 512 MB of RAM
Bontoux and Feillet (2008)	Ant Colony Algorithm	C++. Pentium4 @ 2.00GHz, Linux / Debian

while the remaining authors used less efficient architectures in relation to the one used in this work.

Cuellar-Usaquén et al. (2023) implemented a mixed integer programming (MIP) formulation to verify the optima of the instances that had not been solved to optimality. Regarding Class 1, the approach resulted in new best-known values for instances with unknown optima, having reached 19 out of the 25 optimal solutions. The authors also reported the best performance in instances with known optima. However, in Class 3, the Transgenetic Algorithm proposed by Goldbarg et al. (2009) and the Ant Colony Algorithm introduced by Bontoux and Feillet (2008) still hold the best results in the literature to our knowledge, although one new best-known value was achieved by Cuellar-Usaquén et al. (2023).

Subsection 4.3.1 presents the comparison for the Class 6 instances. Subsections 4.3.2 and 4.3.3 present the comparison for the Class 1 and Class 3 instances, respectively.

4.3.1 Asymmetric Instances: Class 6

Table 4.4 presents a comparison of gaps and computational times for the Class 6 instances considered by Cuellar-Usaquén et al. (2023). The results correspond to 75 instances of Class 6 with $|M| \in \{50, 75, 100, 200, 300\}$ and $|K| \in \{50, 100, 200\}$, including five instances per unique combination of $|M|$ and $|K|$. The ILS-RC was executed five times for each instance, providing the same number of replicates used by Cuellar-Usaquén et al. (2023), as the algorithm incorporates randomized features. Instances are grouped by the number of nodes $|M|$ and the number of items $|K|$. The reported values correspond to the averages of all instances of the indicated sizes, including the five replicates considered for each instance. As previously mentioned, the results from the two procedures considered by the GRASP-based methodology are presented, including the purchase-focused (P) and route-focused (R) procedures. The ILS-RC obtained better results on average in less computational time, although a less efficient computing architecture was used. The results outperformed for every set of instances grouped by the number of nodes $|M|$ and number of items $|K|$, with the greatest improvement for $|K| = 200$.

Table 4.4: Gaps and computational times for the Class 6 instances.

		Gap %			**Time (s)**		
		GRASP + Filtering/PR		ILS - RC	GRASP + Filtering/PR		ILS - RC
		P	R	-	P	R	-
$\|M\|$	50	0.03	0.04	0.01	33	39	3
	75	0.04	0.04	0.01	48	58	6
	100	0.03	0.04	0.02	50	62	9
	200	0.06	0.06	0.03	71	71	23
	300	0.11	0.09	0.05	122	113	36
$\|K\|$	50	0.01	0.01	0.00	19	21	6
	100	0.03	0.03	0.00	42	44	12
	200	0.12	0.12	0.06	133	141	29
Average		0.05	0.05	0.02	65	69	15

4.3.2 Symmetric Instances: Class 1

Class 1 instances have the number of nodes fixed at $|M| = 33$, and the number of items ranges in $|K| \in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$. These instances are symmetric, in contrast to the instances in Class 6. There are five instances for each value of $|K|$, resulting in a total of 50 instances. The ILS-RC was executed five times for each instance, providing the same number of replicates used in Cuellar-Usaquén et al. (2023). Instances up to $|K| = 250$ were solved to optimality by Laporte et al. (2003) and are referred to as Class 1 closed instances, in opposition to the open instances with $|K| \geqslant 300$ for which no optimal values were reported in the literature.

Table 4.5 presents a comparison of gaps and computational times for the Class 1 closed instances considered by Cuellar-Usaquén et al. (2023). The ILS-RC obtained better results for $|K| = 250$ and matched the results for $|K| < 250$. The number of iterations was able to be increased to $k_{\max} = 15000$ since the number of nodes is fixed at a relatively low value $|M| = 33$. This significantly influences the ILS-RC in terms of efficiency, as the search complexity of the neighborhoods is connected to $|M|$.

Table 4.5: Gaps and computational times for the Class 1 closed instances.

		Gap %			**Time (s)**		
		GRASP + Filtering/PR		ILS - RC	GRASP + Filtering/PR		ILS - RC
$\|M\|$	$\|K\|$	P	R	-	P	R	-
33	50	0.00	0.00	0.00	20	20	3
33	100	0.00	0.00	0.00	39	42	5
33	150	0.00	0.00	0.00	68	70	7
33	200	0.00	0.00	0.00	92	97	9
33	250	0.21	0.22	0.00	128	131	11
Average		0.04	0.04	0.00	69	72	7

Regarding the Class 1 open instances, 19 out of 25 optimal values were reported by Cuellar-Usaquén et al. (2023). The remaining six optimal values were verified by implementing the MIP formulation presented in Section 2.3, in order to confirm if the optimum was reached in each instance. Table 4.6 presents the results of this implementation for the six instances. Table 4.7 presents a comparison of the objective function values for the Class 1 open instances between the ILS-RC and the work of Cuellar-Usaquén et al. (2023). The values correspond to the best-found solution across the five replicates that were run for each instance, and values with asterisks correspond to optimal solutions. As previously mentioned, Cuellar-Usaquén et al. (2023) reached 19 optimal solutions in the open instances, while the ILS-RC reached the optimal solution in every instance.

Table 4.6: Class 1 optima obtained with the implemented MIP formulation.

$\|M\|$	$\|K\|$	Id	$Z(x^*)$	Time (s)
33	300	3	13992	61
33	300	5	14181	40
33	350	5	15789	51
33	400	3	16838	182
33	450	1	17847	42
33	450	4	17522	198

Table 4.7: Objective function values for the Class 1 open instances.

			Objective Function Value			**Time(s)**
			GRASP + Filtering/PR		ILS-RC	ILS-RC
$\|M\|$	$\|K\|$	Id	P	R	-	-
33	300	1	13,883*	13,883*	13,883*	13
33	300	2	13,954*	13,954*	13,954*	13
33	300	3	13,995	14,020	13,992*	14
33	300	4	13,765*	13,765*	13,765*	13
33	300	5	14,281	14,281	14,181*	14
33	350	1	15,306*	15,306*	15,306*	15
33	350	2	14,324*	14,324*	14,324*	15
33	350	3	15,858*	15,858*	15,858*	17
33	350	4	15,550*	15,550*	15,550*	15
33	350	5	15,816	15,810	15,789*	15
33	400	1	17,097	17,084*	17,084*	18
33	400	2	16,158	16,116*	16,116*	17
33	400	3	16,903	16,903	16,838*	17
33	400	4	17,149*	17,149*	17,149*	18
33	400	5	17,162*	17,162*	17,162*	16
33	450	1	17,860	17,860	17,847*	19
33	450	2	17,329*	17,329*	17,329*	17
33	450	3	18,203*	18,203*	18,203*	20
33	450	4	17,530	17,524	17,522*	19
33	450	5	18,424*	18,428	18,424*	19
33	500	1	19,273	19,270*	19,270*	20
33	500	2	19,310*	19,310*	19,310*	20
33	500	3	19,399	19,376*	19,376*	22
33	500	4	19,300*	19,300*	19,300*	21
33	500	5	18,956*	18,956*	18,956*	20

4.3.3 Symmetric Instances: Class 3

Class 3 contains the Euclidean instances, which are also symmetric, with $|M| \in \{50, 100, 150, 200, 250, 300, 350\}$ and $|K| \in \{50, 100, 150, 200\}$, with five instances per combination of $|M|$ and $|K|$. The traveling costs are determined by Euclidean distances, which are generated using the procedure presented in Figure A.1. Class 3 instances are also distinguished as closed and open instances, and the ILS-RC was executed five times for each instance.

Table 4.8 provides a comparison of gaps and computational times between the proposed approach and the best reported results regarding the Class 3 closed instances. The ILS-RC was able to obtain better results for every set of instances grouped by $|M|$ and $|K|$ in relation to the GRASP/Path-Relinking algorithm proposed by Cuellar-Usaquén et al. (2023). However, the ILS-RC underperformed in comparison with the Transgenetic Algorithm proposed by Goldbarg et al. (2009) for $|M| > 100$ and for every set of instances grouped by $|K|$. Class 3 instances tend to be restraining in terms of computational time due to an increased average route size in relation to other classes.

Table 4.8: Gaps and computational times for the Class 3 closed instances.

		Gap %				**Time (s)**			
		Trans-genetic	GRASP + Filtering/PR		ILS - RC	Trans-genetic	GRASP + Filtering/PR		ILS - RC
		-	P	R	-	-	P	R	-
$\|M\|$	50	0.00	0.27	0.68	0.00	1	35	32	8
	100	0.00	1.49	1.16	0.00	1	56	51	24
	150	0.03	2.83	3.04	0.11	2	67	73	55
	200	0.00	2.26	2.49	0.29	3	83	75	81
	250	0.00	2.72	5.03	0.12	3	106	109	88
$\|K\|$	50	0.00	1.07	2.06	0.13	1	27	28	18
	100	0.00	0.64	0.55	0.06	2	54	45	36
	150	0.00	3.68	4.09	0.06	2	74	80	61
	200	0.03	2.26	3.22	0.14	3	121	120	78
Average		0.01	1.91	2.48	0.10	2	69	68	47

Table 4.9 presents a comparison of the objective function values for the Class 3 open instances. The values correspond to the best-found solution across the five replicates that were run for each instance. The ILS-RC could not reach new best-known values, merely matching best-known values reported by other works. One possible explanation for this is that the reported values might already be close to the optimal values, leaving less opportunity for improvement.

Table 4.9: Objective function values for Class 3 open instances.

			Objective Function Value			
$\|M\|$	$\|K\|$	Id	Bontoux et al. (2008)	Goldbarg et al. (2009)	Cuellar et al. (2023)	ILS - RC
200	150	4	**2419**	**2419**	**2419**	**2419**
200	200	4	**2344**	**2344**	2403	**2344**
250	100	1	**1301**	**1301**	**1301**	**1301**
250	100	4	**1673**	**1673**	**1673**	**1673**
250	100	5	**1641**	**1641**	**1641**	**1641**
250	150	4	**1836**	**1836**	**1836**	**1836**
250	150	5	**1531**	**1531**	**1531**	**1531**
250	200	2	**2785**	2786	2838	2789
250	200	3	**1924**	**1924**	1947	**1924**
250	200	4	**2116**	**2116**	**2116**	**2116**
250	200	5	**1797**	**1797**	1851	1992
300	50	1	**1477**	**1477**	1485	**1477**
300	50	2	**813**	**813**	879	**813**
300	50	3	**1117**	**1117**	**1117**	**1117**
300	50	4	**1176**	**1176**	**1176**	**1176**
300	50	5	1257	**1256**	**1256**	1257
300	100	1	**1035**	**1035**	**1035**	**1035**
300	100	2	**1179**	1180	**1179**	**1179**
300	100	3	**1498**	**1498**	**1498**	**1498**
300	100	4	**1749**	**1749**	1770	**1749**

Continued on next page

Table 4.9 – continued from previous page

			Objective Function Value			
$\|M\|$	$\|K\|$	Id	Bontoux et al. (2008)	Goldbarg et al. (2009)	Cuellar et al. (2023)	ILS - RC
300	100	5	**1774**	**1774**	**1774**	**1774**
300	150	1	**1457**	**1457**	**1457**	**1457**
300	150	2	**1656**	**1656**	**1656**	1685
300	150	3	2485	**2484**	**2484**	**2484**
300	150	4	**1801**	**1801**	**1801**	**1801**
300	150	5	**1816**	**1816**	**1816**	**1816**
300	200	1	1815	**1803**	1814	1812
300	200	2	1791	**1790**	1839	1836
300	200	3	2442	**2437**	2443	2441
300	200	4	**1815**	**1815**	1960	**1815**
300	200	5	2022	**2014**	2022	2015
350	50	1	**723**	**723**	**723**	**723**
350	50	2	**736**	**736**	**736**	**736**
350	50	3	**942**	**942**	**942**	**942**
350	50	4	**805**	**805**	917	**805**
350	50	5	**1125**	1225	1225	1225
350	100	1	**1317**	**1317**	**1317**	**1317**
350	100	2	**962**	**962**	**962**	**962**
350	100	3	**796**	**796**	**796**	**796**
350	100	4	**1059**	**1059**	**1059**	**1059**
350	100	5	**1566**	**1566**	**1566**	1567
350	150	1	1457	1459	**1455**	1457
350	150	2	**1315**	**1315**	**1315**	**1315**
350	150	3	**2553**	2558	**2553**	**2553**
350	150	4	**1239**	**1239**	**1239**	**1239**
350	150	5	**2288**	**2288**	**2288**	**2288**
350	200	1	1503	**1498**	1511	1503
350	200	2	1374	**1369**	**1369**	**1369**

Continued on next page

Table 4.9 – continued from previous page

			Objective Function Value			
$\|M\|$	$\|K\|$	Id	Bontoux et al. (2008)	Goldbarg et al. (2009)	Cuellar et al. (2023)	ILS - RC
350	200	3	**1873**	**1873**	**1873**	**1873**
350	200	4	1385	**1356**	1359	1359
350	200	5	**2336**	**2336**	**2336**	2341

4.4 Wilcoxon Signed-Rank Tests

Table 4.10 presents the Wilcoxon signed-rank one-tailed test statistics and corresponding p-values computed between the ILS-RC and the work of Cuellar-Usaquén et al. (2023). Open instances are not included since only the best-found solutions were reported between the five replicates conducted per instance. The best values from both variations of the GRASP methodology (P and R) were selected for the tests. Two tests are computed at a significance level of 5%, one considering the values grouped by $|M|$ and the other by $|K|$. Let d_i represent the differences between the GRASP and the ILS-RC paired values presented in Tables 4.4, 4.5, and 4.8. The null hypothesis states that the median for d_i is equal or inferior to zero. The alternative hypothesis states that the median for d_i is greater than zero, that is,

$$H_0 : median_{d_i} \leqslant 0 \quad \text{vs} \quad H_1 : median_{d_i} > 0$$

Table 4.10: Wilcoxon signed-rank one-tailed tests between the work of Cuellar-Usaquén et al. (2023) and the ILS-RC.

Group	Test Statistic	p-value	Decision
$\|M\|$	66	0.002	Reject H_0
$\|K\|$	36	0.007	Reject H_0

The p-value is 0.002 for the test that considers the instances grouped by $|M|$ and it is 0.007 for the test with the instances grouped by $|K|$. The null hypothesis is rejected at a significance level of 5% for each test. There is statistical evidence to claim that the average gaps of the ILS-RC are inferior

in relation to the average gaps of the GRASP-based methodology proposed by Cuellar-Usaquén et al. (2023), including at a significance level of 1%.

Table 4.11 presents additional tests that were computed between the values obtained by the ILS-RC and the work of Goldbarg et al. (2009). Let d_i represent the differences between the ILS-RC and the Transgenetic Algorithm paired values presented in Table 4.8. Two tests were computed using the values grouped by $|M|$ and $|K|$, respectively.

Table 4.11: Wilcoxon signed-rank one-tailed tests between the ILS-RC and the work of Goldbarg et al. (2009).

Group	Test Statistic	p-value	Decision
$\|M\|$	6	0.091	Fail to Reject H_0
$\|K\|$	10	0.049	Reject H_0

The p-value is 0.091 for the test that considers the instances grouped by $|M|$. The null hypothesis is not rejected at a significance level of 5%, meaning there is not enough statistical evidence to claim that the average gaps of the Transgenetic Algorithm proposed by Goldbarg et al. (2009) are inferior in relation to the average gaps produced by the ILS-RC. However, it is important to note that the number of unequal observations is small, which affects the power of the test. The p-value is 0.049 for the test with the instances grouped by $|K|$. The null hypothesis is rejected at a significance level of 5%, meaning there is statistical evidence to claim that the average gaps of the Transgenetic Algorithm proposed by Goldbarg et al. (2009) are inferior in relation to the average gaps produced by the ILS-RC for the Class 3 closed instances.

Chapter 5

Conclusion

This study proposed the ILS-RC, which is an ILS algorithm complemented with a route configuration procedure that adjusts the route size and respective subset of markets. Neighborhoods are not necessarily searched extensively in this complementary procedure, unlike in the local search component of the ILS-RC. This is prudent in order to produce, on average, routes of a specific size, aiming at the number of markets in the optimal solutions. Following this procedure, a local search is applied to reach the local optimum by arranging the sequence in which the markets are visited. The computational experiment revealed the effectiveness and efficiency of the ILS-RC, able to consistently obtain quality solutions in reasonable time despite the incorporated randomized features.

Benchmark instances were considered, providing a comparison with the best results in the literature to our knowledge. For the Class 6 asymmetric instances proposed by Singh and van Oudheusden (1997), the results were compared with the work of Cuellar-Usaquén et al. (2023), achieving better results in less computational time. Additional experiments were presented for the Class 1 and Class 3 symmetric instances, also addressed by Cuellar-Usaquén et al. (2023). The ILS-RC parameters were adjusted for each class, providing a better fit to the features of each group. Class 1 instances that have been solved optimally by Laporte et al. (2003) are referred to as closed instances, in contrast to open instances for which no optimal values were reported. For the closed instances, the ILS-RC was able to reach the optimal solution in every instance, outperforming the previously published results. For the open instances, the ILS-RC was able to reach every optimum value across the five replicates executed for each instance. In contrast, the methodology

proposed by Cuellar-Usaquén et al. (2023) was only able to reach 19 out of the 25 optimum solutions. In the Class 3 instances, commonly known as Euclidean instances, the ILS-RC was able to provide better results in relation to Cuellar-Usaquén et al. (2023) for the closed instances but was outperformed by the results achieved by Goldbarg et al. (2009). In the open instances, the ILS-RC displayed competitive performance with the other articles, including the work of Bontoux and Feillet (2008). Ultimately, the improvements found in relation to the methodology proposed by Cuellar-Usaquén et al. (2023) demonstrated to be statistically significant. The ILS-RC proved to be effective and versatile, able to adjust to the asymmetric and symmetric TPP instances with reliable performance, both in terms of solution quality and computational time.

Future works may address the TPP with different methods, namely the adaptive large neighborhood search (ALNS) metaheuristic. This relatively new concept has gained increased interest, as reflected by the growing number of publications in recent years (Mara et al., 2022).

References

Almeida, C. P., Gonçalves, R. A., Goldbarg, E. F., Goldbarg, M. C., and Delgado, M. R. (2012). An experimental analysis of evolutionary heuristics for the biobjective traveling purchaser problem. *Annals of Operations Research*, 199:305–341.

Angelelli, E., Mansini, R., and Vindigni, M. (2009). Exploring greedy criteria for the dynamic traveling purchaser problem. *Central European Journal of Operations Research*, 17:141–158.

Angelelli, E., Mansini, R., and Vindigni, M. (2016). The stochastic and dynamic traveling purchaser problem. *Transportation Science*, 50(2):642–658.

Beraldi, P., Bruni, M. E., Manerba, D., and Mansini, R. (2017). A stochastic programming approach for the traveling purchaser problem. *IMA Journal of Management Mathematics*, 28(1):41–63.

Bernardino, R. and Paias, A. (2018). Metaheuristics based on decision hierarchies for the traveling purchaser problem. *International Transactions in Operational Research*, 25(4):1269–1295.

Bernardino, R. and Paias, A. (2024). The family capacitated vehicle routing problem. *European Journal of Operational Research*, 314(3):836–853.

Bianchessi, N., Irnich, S., and Tilk, C. (2021). A branch-price-and-cut algorithm for the capacitated multiple vehicle traveling purchaser problem with unitary demand. *Discrete Applied Mathematics*, 288:152–170.

Bianchessi, N., Mansini, R., and Speranza, M. G. (2014). The distance constrained multiple vehicle traveling purchaser problem. *European Journal of Operational Research*, 235(1):73–87.

Bontoux, B. and Feillet, D. (2008). Ant colony optimization for the traveling purchaser problem. *Computers & Operations Research*, 35(2):628–637. Part Special Issue: Location Modeling Dedicated to the memory of Charles S. ReVelle.

Burstall, R. M. (1966). A heuristic method for a job-scheduling problem. *Journal of the Operational Research Society*, 17:291–304.

Cheaitou, A., Hamdan, S., Larbi, R., and Alsyouf, I. (2021). Sustainable traveling purchaser problem with speed optimization. *International Journal of Sustainable Transportation*, 15(8):621–640.

Cuellar-Usaquén, D., Gomez, C., and Álvarez-Martínez, D. (2023). A grasp/path-relinking algorithm for the traveling purchaser problem. *International Transactions in Operational Research*, 30(2):831–857.

Gendreau, M., Manerba, D., and Mansini, R. (2016). The multi-vehicle traveling purchaser problem with pairwise incompatibility constraints and unitary demands: A branch-and-price approach. *European Journal of Operational Research*, 248(1):59–71.

Goldbarg, M. C., Bagi, L. B., and Goldbarg, E. F. G. (2009). Transgenetic algorithm for the traveling purchaser problem. *European Journal of Operational Research*, 199(1):36–45.

Golden, B., Levy, L., and Dahl, R. (1981). Two generalizations of the traveling salesman problem. *Omega*, 9(4):439–441.

Gouveia, L., Paias, A., and Voß, S. (2011). Models for a traveling purchaser problem with additional side-constraints. *Computers & Operations Research*, 38(2):550–558.

Kang, S. and Ouyang, Y. (2011). The traveling purchaser problem with stochastic prices: Exact and approximate algorithms. *European Journal of Operational Research*, 209(3):265–272.

Kucukoglu, I. (2022). The traveling purchaser problem with fast service option. *Computers & Operations Research*, 141:105700.

Laporte, G., Riera-Ledesma, J., and Salazar-González, J.-J. (2003). A branch-and-cut algorithm for the undirected traveling purchaser problem. *Operations Research*, 51(6):940–951.

Manerba, D. and Mansini, R. (2015). A branch-and-cut algorithm for the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints. *Networks*, 65(2):139–154.

Manerba, D. and Mansini, R. (2016). The nurse routing problem with workload constraints and incompatible services. *IFAC-PapersOnLine*, 49(12):1192–1197.

Manerba, D., Mansini, R., and Riera-Ledesma, J. (2017). The traveling purchaser problem and its variants. *European Journal of Operational Research*, 259(1):1–18.

Mansini, R. and Tocchella, B. (2009). The traveling purchaser problem with budget constraint. *Computers & Operations Research*, 36(7):2263–2274.

Mara, S. T. W., Norcahyo, R., Jodiawan, P., Lusiantoro, L., and Rifai, A. P. (2022). A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, 146:105903.

Ong, H. L. (1982). Approximate algorithms for the travelling purchaser problem. *Operations Research Letters*, 1(5):201–205.

Palomo-Martínez, P. J. and Salazar-Aguilar, M. A. (2019). The bi-objective traveling purchaser problem with deliveries. *European Journal of Operational Research*, 273(2):608–622.

Pearn, W. and Chien, R. (1998). Improved solutions for the traveling purchaser problem. *Computers & Operations Research*, 25(11):879–885.

Ramesh, T. (1981). Traveling purchaser problem. *Opsearch*, 18(1-3):78–91.

Riera-Ledesma, J. and Salazar-González, J. J. (2005). The biobjective travelling purchaser problem. *European Journal of Operational Research*, 160(3):599–613.

Riera-Ledesma, J. and Salazar-González, J.-J. (2006). Solving the asymmetric traveling purchaser problem. *Annals of Operations Research*, 144(1):83–97.

Riera-Ledesma, J. and Salazar-Gonzalez, J.-J. (2012). Solving school bus routing using the multiple vehicle traveling purchaser problem: A branch-and-cut approach. *Computers & Operations Research*, 39(2):391–404.

Singh, K. N. and van Oudheusden, D. L. (1997). A branch and bound algorithm for the traveling purchaser problem. *European Journal of operational research*, 97(3):571–579.

Voß, S. (1996). Dynamic tabu search strategies for the traveling purchaser problem. *Annals of Operations Research*, 63:253–275.

Appendix

Section A.1 presents the pseudocodes for the neighborhood searches. Section A.2 presents the procedure used to generate the Euclidean distances in the Class 3 instances.

A.1 Neighborhood Search Algorithms

Algorithms A.1, A.2, A.3, A.4, and A.5, present the search procedures of neighborhoods $\mathcal{N}_{\text{add}}(x)$, $\mathcal{N}_{\text{drop}}(x)$, $\mathcal{N}_{\text{exchange}}(x)$, $\mathcal{N}_{\text{move}}(x)$, and $\mathcal{N}_{\text{switch}}(x)$, respectively.

Algorithm A.1 Search within neighborhood $\mathcal{N}_{\text{add}}(x)$.

Require: initial solution x
1: incumbent $\leftarrow x$
2: $z = Z(x)$
3: **for** i **in** markets unvisited in x **do**
4: **for** $j = 1$ **to** 1 + number of markets visited in x **do**
5: $y \leftarrow x$
6: Add market i at position j in the route of solution y
7: **if** $Z(y) < z$ **then**
8: incumbent$\leftarrow y$
9: $z = Z(y)$
10: **end if**
11: **end for**
12: **end for**
13: $x \leftarrow$ incumbent

Algorithm A.2 Search within neighborhood $\mathcal{N}_{\text{drop}}(x)$.

Require: initial solution x
1: incumbent $\leftarrow x$
2: $z = Z(x)$
3: **for** i **in** markets visited in x **do**
4: $y \leftarrow x$
5: Remove market i from the route of solution y
6: **if** y covers the item list $\wedge$ $Z(y) < z$ **then**
7: incumbent $\leftarrow y$
8: $z = Z(y)$
9: **end if**
10: **end for**
11: $x \leftarrow$ incumbent

Algorithm A.3 Search within neighborhood $\mathcal{N}_{\text{exchange}}(x)$.

Require: initial solution x
1: incumbent $\leftarrow x$
2: $z = Z(x)$
3: **for** i **in** markets visited in x **do**
4: **for** j **in** markets unvisited in x **do**
5: $y \leftarrow x$
6: Exchange market i in the route of solution y with market j
7: **if** y covers the item list $\wedge$ $Z(y) < z$ **then**
8: incumbent $\leftarrow y$
9: $z = Z(y)$
10: **end if**
11: **end for**
12: **end for**
13: $x \leftarrow$ incumbent

Algorithm A.4 Search within neighborhood $\mathcal{N}_{\text{move}}(x)$.

Require: initial solution x
1: incumbent $\leftarrow x$
2: $z = Z(x)$
3: **for** i **in** markets visited in x **do**
4: $y \leftarrow x$
5: **for** each position j in the route of solution y **do**
6: Move market i to position j in the route of solution y
7: **if** $Z(y) < z$ **then**
8: incumbent $\leftarrow y$
9: $z = Z(y)$
10: **end if**
11: **end for**
12: **end for**
13: $x \leftarrow$ incumbent

Algorithm A.5 Search within neighborhood $\mathcal{N}_{\text{switch}}(x)$.

Require: initial solution x
incumbent $\leftarrow x$
$z = Z(x)$
for $i = 1$ **to** number of markets visited in x **do**
  for $j = i + 1$ **to** number of visited markets in x **do**
    $y \leftarrow x$
    Switch the markets at the positions i and j in route of solution y
    if $Z(y) < z$ **then**
      incumbent $\leftarrow y$
      $z = Z(y)$
    end if
  end for
end for
$x \leftarrow$ incumbent

A.2 Generation of the Euclidean Distances

The Class 3 Euclidean distances are calculated using the procedure presented in Figure A.1. This procedure was taken from the TPP website https://webpages.ull.es/users/jriera/TPP.htm.

Generation of distances in Class 3

```
------------------------------------------------------------
int TPPLIB::euc_2d(int i,int j)
{
  double xd=NodeCoords_[i].first() -NodeCoords_[j].first();
  double yd=NodeCoords_[i].second()-NodeCoords_[j].second();

  double co=sqrt(xd*xd+yd*yd);
  return int(co);
}
------------------------------------------------------------
```

Figure A.1: Generation of the Euclidean distances for the Class 3 instances.