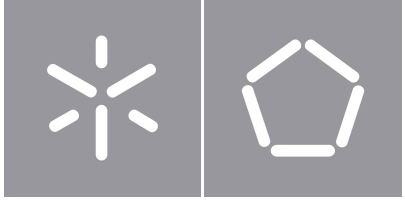
University of Minho
School of Engineering

João Miguel Pimenta da Silva

Optimization of the Inventory Routing Problem with Artificial Intelligence

november 2023



University of Minho
School of Engineering

João Miguel Pimenta da Silva

Optimization of the Inventory Routing Problem with Artificial Intelligence

Masters Dissertation
Master's in Informatics Engineering

Dissertation supervised by
César Analide Freitas Silva Costa Rodrigues

november 2023

Copyright and Terms of Use for Third Party Work

License granted to users of this work:

Acknowledgements

Firstly, I want to give thanks to my supervisor, Alexandra Alves Oliveira, for her advice and dedication over the course of the development of this dissertation.

I also want to give thanks to my professor, César Analide Freitas Silva Costa Rodrigues, who also made possible the development of this dissertation.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, november 2023

Assinado por: **JOÃO MIGUEL PIMENTA DA SILVA**
Num. de Identificação: 15976897
Data: 2023.11.20 16:27:05+00'00'

João Miguel Pimenta da Silva

Abstract

Nowadays the success of a business is dependent on the ability to effectively integrate in an intricate network of entities that are connected by material and information flows, inventory management being one of the main concerns. These flows are characterized by decision-making processes that will vary depending on the environment, entities and business models in the network. So, these networks need a decision-making system capable of providing solutions that dictate the optimal way the network and its entities provide and collect inventory in order to reduce costs and maximize profit. In the context of this dissertation, the problem arises when there is a stock disruption in the network and outside entities can no longer answer the stores' supply requests and these stores become the entities responsible for requesting and delivering products to each other. This problem is modeled as an Inventory Routing problem, since it encompasses inventory management and routing decisions. The main goal of the system can be described as maximizing the collection of products per travel distance, without causing stock-outs at any supplier, for the entire network. The problem at hand is an optimization problem.

In order to solve this optimization problem, first, the structural characteristics and key aspects were identified and studied, followed by the mathematical conceptualization, which involved the definition of the objective function and the corresponding set of constraints. The mathematical formulation allows the problem to be translated into a specific and precise mathematical language, making it possible to evaluate solutions, by means of a fitness function, and apply optimization algorithms to solve the problem. These optimization algorithms can be approximate or exact methods and their suitability to the problem depends on many factors such as the size, structure and complexity of the problem. So, the choice of the optimization algorithms must be preceded by a careful analysis of the problem at hand and its characteristics.

After this, in the implementation phase, two adaptations of the genetic algorithm, two adaptions of the simulated annealing algorithm, two adaptations of the tabu search algorithm were developed. Additionaly, another algorithm responsible for generating reference solutions was also developed (Dynamic2). In order

to test and compare the developed optimization algorithms, three different sized scenarios were generated. Each of these scenarios has a different amount of data associated with it, whether it be in the number of stores, types of products or number of requests. As to compare the results of the different instances of the algorithms fairly in each scenario, these were made to generate roughly the same number of solutions.

In scenarios 1 and 3, all the optimization algorithms developed were successful in finding solutions with higher fitness values than the baseline Dynamic2 solution. In scenario 2, due to time constraints and computational complexities, only the Genetic algorithm and the Genetic algorithm with Elitism using an initial population consisting of solutions generated by the Dynamic2 algorithm, managed to find solutions with higher fitness value than the baseline solution. In this scenario, the developed optimization algorithms were also tested using feasible solutions generated through random mechanisms as initial solutions. These instances also achieve solutions with improved fitness values when compared to their respective initial solutions or populations.

Furthermore, in scenarios 1 and 3, the Genetic algorithm with an initial population consisting of feasible solutions generated through random mechanisms was the optimization algorithm that found the best solutions, with these solutions having fitness values 56.14% and 92.07% greater than the baseline Dynamic2 solution's, respectively. In scenario 2, the Genetic algorithm with Elitism, utilizing an initial population consisting of solutions generated by the mentioned Dynamic2 algorithm, found the solution with the highest fitness value, being approximately 1.00% higher than the baseline solution.

Keywords Optimization, Inventory Routing Problem, Approximate Methods, Exacts Methods, Metaheuristics

Resumo

Atualmente, o sucesso de um negócio depende da capacidade de se integrar de uma forma eficaz numa rede de entidades conectadas por fluxos de materiais e informações, sendo a gestão de inventário uma das principais preocupações. Esses fluxos são caracterizados por processos de tomada de decisão que variam dependendo do ambiente, entidades e modelos de negócios na rede. Estas redes precisam de um sistema de tomada de decisão capaz de fornecer soluções que otimizem a forma como a rede e as suas entidades fornecem e recolhem inventário para reduzir custos e maximizar lucros. No contexto desta dissertação, o problema surge quando há uma interrupção de *stock* na rede e as entidades externas já não conseguem responder às solicitações de abastecimento das lojas e essas lojas tornam-se as entidades responsáveis por solicitar e entregar produtos entre si. Este problema é modelado como um *Inventory Routing Problem*, pois engloba decisões de gestão de inventário e *routing*. O objetivo principal do sistema pode ser descrito como maximizar a recolha de produtos por distância percorrida, sem causar *stock-outs* nos fornecedores, para toda a rede. O problema em questão é um problema de otimização.

Para resolver este problema de otimização, primeiro foram identificadas e estudadas as características estruturantes do problema, seguido de uma conceptualização matemática do mesmo, através da definição da função objetivo e do conjunto de restrições associadas. A formulação matemática permite que o problema seja traduzido para uma linguagem específica e precisa, tornando possível avaliar e comparar soluções, através da função objetivo, e aplicar algoritmos de otimização para resolver o problema. Esses algoritmos de otimização podem ser métodos aproximados ou exatos e sua adequação ao problema depende de muitos fatores, como tamanho, estrutura e complexidade do problema. Então, a escolha dos algoritmos de otimização deve ser precedida por uma análise cuidadosa do problema em questão e suas características.

Posteriormente, na fase de implementação, foram desenvolvidas duas adaptações de algoritmos genéticos, duas de simulated annealing e duas de tabu search, juntamente com outro algoritmo, responsável

por gerar soluções de referência (*Dynamic2*). Com o intuito de testar e comparar os algoritmos de otimização desenvolvidos, foram criados três cenários de tamanhos diferentes. Cada um destes cenários possui diferentes quantidades de dados associados, seja em termos do número de lojas, tipos de produtos ou quantidade de pedidos. Para garantir uma comparação justa entre as diferentes instâncias dos algoritmos em cada cenário, estes foram configurados para gerar aproximadamente o mesmo número de soluções.

Nos cenários 1 e 3, todos os algoritmos de otimização desenvolvidos tiveram sucesso em encontrar soluções com valores de *fitness* superiores à solução de referência do algoritmo *Dynamic2*. No cenário 2, devido a limitações de tempo e complexidades computacionais, apenas o algoritmo Genético e o algoritmo Genético com Elitismo, utilizando uma população inicial composta por soluções geradas pelo algoritmo *Dynamic2*, conseguiram encontrar soluções com valores de *fitness* superiores à solução de referência. Neste cenário, os algoritmos de otimização desenvolvidos foram também testados utilizando soluções possíveis geradas aleatoriamente como soluções iniciais. Estes casos também alcançaram soluções com valores de *fitness* maiores quando comparadas às soluções iniciais ou populações respetivas.

Adicionalmente, nos cenários 1 e 3, o algoritmo Genético com uma população inicial composta por soluções possíveis geradas aleatoriamente foi o algoritmo de otimização que encontrou as melhores soluções, sendo que estas soluções tinham valores de *fitness* superiores em 56.14% e 92.07%, respetivamente, à solução de referência do algoritmo *Dynamic2*. No cenário 2, o algoritmo Genético com Elitismo, utilizando uma população inicial composta por soluções geradas pelo algoritmo *Dynamic2*, encontrou a solução com o valor de *fitness* mais elevado, sendo este aproximadamente 1.00% superior à solução de referência.

Palavras-chave Otimização, Inventory Routing Problem, Métodos Aproximados, Métodos Exatos, Metaheurísticas

Contents

List of Figures

List of Tables

Part I

Introductory material

Chapter 1

Introduction

1.1 Context

In this day and age, individual businesses no longer compete as solely autonomous entities, but rather in complex supply chains Lambert and Cooper [2000]. The success of a business depends on its ability to effectively integrate and communicate in an intricate network of organizations that are linked by material, information and financial flows Lambert and Cooper [2000], Stadtler et al. [2015]. These flows involve decision-making processes which may be different according to the business model and characteristics of the problem being tackled.

Retail Consult, which is the company this project is being developed for, provides consultancy and technological solutions services to groups of retailers and needs a decision-making system that optimizes the inventory management of a network of retailers within a supply chain. The retailers in question are all stores of the same retailer group that are connected through a network and are monitored by the mentioned decision-making system. The problem that the project aims to address arises when there is a disruption in the supply chain, where external entities are unable to fulfill the stores' supply requests. This leads to the stores within the network becoming solely responsible for requesting and delivering products to one another. This system must be capable of calculating a solution that optimizes the inventory flow between stores and consequently the whole network. The problem at hand is an optimization problem.

1.2 Motivation

After the covid-19 pandemic, supply chains are once again being put to the test by the conflict in Ukraine. During the pandemic, logistics disruptions, shortages and soaring energy prices have all contributed to supply shortages and spiraling transport costs. The conflict in Ukraine, has put additional pressure on

the already tense global supply chain Ngoc et al. [2022]. This conflict will continue to have an impact on the availability of raw materials, food and energy. Which in turn, will have an inflationary impact on the aforementioned resources' cost. These are some of the reasons why it is becoming increasingly important to have an optimized supply chain with emphasis on Inventory Management, not only due to the business competitiveness of today but also due to the world's volatile social and economic situation.

In order to tackle these problems and improve the material, information, and financial flows of the supply chain and its entities, it is necessary to find the optimal solutions for the supply chain, the best way to do things, the most efficient manner, the most economical process Pedregal [2004]. This is called optimization. Artificial Intelligence methods have been widely used to solve optimization problems. These methods, such as optimization algorithms, have the advantage that they can deal with complex problems that cannot be solved by conventional methods Ongsakul and Dieu [2013]. Traditional engineering techniques are often visualized and reasoned about by humans in two or three dimensions. Modern optimization techniques, however, can be applied to problems with millions of variables and constraints Kochenderfer and Wheeler [2019].

Every problem is different from one another, and so are the techniques, constraints, parameters and variables used in order to solve them. The objectives in each problem that needs to be optimized may also differ. So, it is crucial to understand the context and state of the art of the problem in order to optimize it successfully and effectively. The first step of optimization is going from plain words describing a problem and its context to its formulation in precise, mathematical terms. This process is of such importance that failure to carry it out accurately may result in unfeasible solutions to problems. So the success of an optimization technique greatly depends on the accurate formulation of the problem through mathematical terms Pedregal [2004]. After this process, an optimization algorithm that best fits the model must be implemented in order to get a solution. Parameters must be tuned and the final results analysed and evaluated. The optimization process is a complex one and must be carried out with attention to detail and with the appropriate measures in order to get the sought optimal results.

When an optimization process is carried out successfully, it can lead to economic profit, such as minimization of cost and maximization of profit, environmental profit, such as minimizing total emission and product wastage, social profit such as minimizing customer dissatisfaction, depending on the sought after objectives and the context of the problem.

1.3 Objectives

The project consists in developing a decision making system capable of optimizing the inventory flow of a network of stores within a supply chain. This system must be able to calculate a solution that provides, for every store in need of goods that requests products to the system, the optimal way they can collect the products they require by identifying the most suitable suppliers, determining the most efficient order of visits, and selecting the most appropriate types and quantities of products to be collected from each supplier.

The main goal of this optimization is to maximize the stores' on-shelf availability without creating stock-outs for any of the network's entities, all while factoring in collection costs. In order to achieve this goal, it can be divided into smaller and more specific objectives:

1. Identification of critical aspects of the problem, modeling and definition of a framework for solving the problem adapted to the context discussed in the previous sections;
2. Development of a mathematical formulation for the optimization problem at hand;
3. Development of appropriate optimization algorithms;
4. Testing, evaluation and comparison of the results in different scenarios with different sizes.

These objectives are presented in order of completion and are all of equal importance, the results from each one will affect the next.

1.4 Thesis Contributions

This thesis contributed to the definition of the optimization problem for the inventory routing problem in stock disruption scenarios.

- Requirement elicitation.
- Definition of the structure of a solution for the problem.
- Mathematical formulation of the problem.
- Generation of different sized scenarios and respective data.
- Development of a wide range of different optimization algorithms.

- Development of a strategy for comparing the solutions and algorithms developed.
- Development of a tool for visualization of the characteristic of the problem, statistics of the solution, representation of the solution.

1.5 Document Structure

In chapter 2, it is presented the background as well as the state of the art of the whole process of optimization problems with emphasis on the inventory routing problem, which is the model of the problem at hand. In the section 2.1, the fundamental concepts of optimization problems are presented and explained as well as the importance of the optimization process. Furthermore, the process of going from plain words to describing and representing an optimization problem and its context in precise mathematical terms is explained in the subsection 2.1.1. In the subsection 2.1.2, combinatorial optimization problems are explained with special emphasis on the inventory routing problem, which is the model of the problem tackled in this project. The main concepts, variations and characteristics of the inventory routing problem are described.

In section 2.2, exact methods for obtaining optimal solutions in optimization problems are discussed. The main concepts, advantages, disadvantages are presented. The exact algorithms that fall under the category of exacts methods are shown and some, such as the branch and bound and dynamic programming are explained along with practical examples of the implementation of these exact algorithms.

In section 2.3, approximate methods for obtaining optimal or near optimal solutions for complex optimization problems are discussed as well as the ramification of these methods, such as: approximation algorithms in subsection 2.3.1 and heuristics and metaheuristics in subsection 2.3.2. The main concepts, advantages, disadvantages, and appropriate use are all discussed in the mentioned section and subsections. In the subsection 2.3.2, single-solution-based and population-based metaheuristics are explained and particular metaheuristics from both of these groups are presented with practical examples of their implementation.

In chapter 3, the methodology of how the objectives of the problem at hand are going to be achieved is presented in section 3.1, the context and characteristics of the optimization problem proposed to solve are discussed in more detail in section 3.2, the mathematical formulation of the problem is shown in section

3.3, and the optimization algorithms developed, in section 3.4, are presented and discussed.

In chapter 4, the results of the developed optimization algorithms for three different sized scenarios are shown and discussed.

In chapter 5, the conclusions of this dissertation are presented.

Chapter 2

Background and Literature Review

In this chapter, it will be described the background methods of this dissertation as well as the state of the art of optimization problems surrounding the Inventory Routing Problem and the approach to solve them.

2.1 Optimization problems

An optimization problem consists in finding values for the variables that optimize an objective function subject to constraints, with the goal of finding the best solution amongst a set of feasible solutions. The solution that minimizes (or maximizes) this function is said to be an optimal solution and the value of the objective function, regarding this solution, is the optimal value Klein and Young [2010]. The process of optimization has various phases, from the mathematical formulation of the problem to the application of the optimization algorithms, tuning and evaluation of the results. Each of these phases is dependent from one another and therefore each one must be done thoroughly so the results are not compromised.

The importance of optimization lies not in trying to find out all about a system but in finding out, with the least possible effort, the best way to adjust the system. If this is carried out well, systems can have a more economic and improved design, they can operate more accurately or at less cost, and the system designer will have a better understanding of the effects of parameter interaction and variation on his design Adby [2013].

According to Talbi [2009], an optimization problem may be defined by the set of feasible solutions S and the objective function f to optimize, $f : S \rightarrow R$. The objective function assigns to every solution $s \epsilon S$ of the search space a real number R indicating its worth. The objective function f allows to define a total order relation between any pair of solutions in the search space.

The main goal of solving an optimization problem is to find a global optimum solution *s**. There are some optimizations problems where there exists more than one global solution. In these cases, the problem may be defined as finding all global solutions.

An objective function can be a complex function that has several optima throughout the search space. These optima can be local or global and during search phase the algorithm can get stuck on the local optimum instead of reaching the global optimum or at least an approximation of it, which is the main goal. Figure 1 represents the objective function in terms of the search space where are depicted two local and one global optima, in the context of a minimization problem.

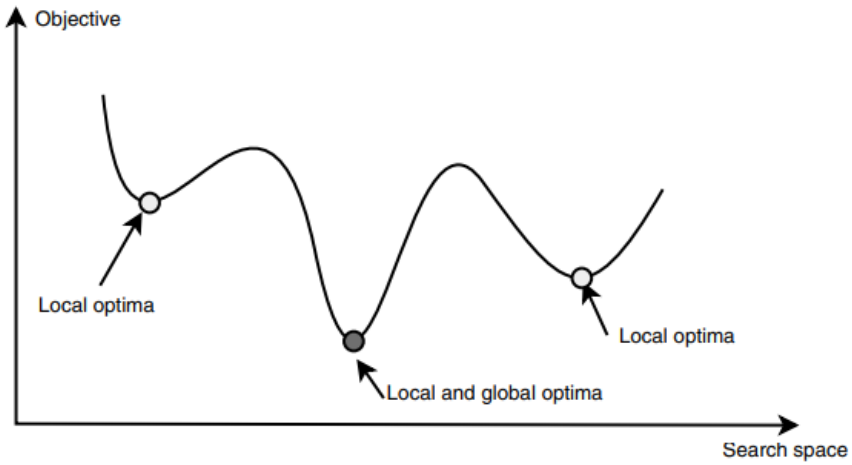


Figure 1: Local optimum and global optimum in a search space. Source: Talbi [2009].

Formally, global and local optimum can be defined as:

Global optimum: A solution $s^* \epsilon S$ is a global optimum if it has a better objective function than all solutions of the search space, that is, $\forall s \epsilon S$, $f(s^*) \leq f(s)$, in the case of a minimization problem.

Local optimum: Relatively to a given neighboring function N, a solution $s^* \epsilon S$ is a local optimum if it has a better quality than all its neighbors; that is, $f(s) \leq f(s')$, $\forall s' \epsilon N(s)$.

Regarding the application of the optimization algorithms, exact and approximation methods can be used. Exact methods obtain optimal solutions and guarantee their optimality while approximate methods generate high quality solutions in a reasonable time for practical use, but there is no guarantee of finding

a global optimal solution Talbi [2009]. These topics will be explained in more detail further down in the document.

Problems can be classified into P or NP complexity classes, where a complexity class represents a set of all problems that can be solved using a given amount of computational resources. The complexity class P represents the set of all decision problems that can be solved by a deterministic machine in polynomial time and the problems under this class are relatively easy to solve Talbi [2009]. The complexity class NP represents the set of all decision problems that can be solved by a nondeterministic algorithm in polynomial time. According to Talbi [2009], a decision problem $A \in$ NP is NP-complete if all other problems of class NP are reduced polynomially to the problem A. Furthermore, NP-hard problems are optimization problems whose associated decision problems are NP-complete. Unless (P=NP), these problems require exponential time to be solved optimally. Approximate methods can play an important role solving these kind of problems, specially when the problems are larger of size and with a complicated structure. Even if this kind of method does not guarantee optimality, it can provide high quality solutions within reasonable time.

2.1.1 Mathematical Formulation of an Optimization Problem

As mentioned in the introduction, the first step of an optimization process is going from plain words describing a problem and its context to its formulation in precise, mathematical terms. This means identifying the parameters, decision variables, constraints and formulating the objective function of a given optimization problem. Since the following phases of the optimization process depend on the results of the mathematical formulation, it is crucial that the constraints and objective function accurately represent the conditions and objectives of the optimization respectively, and that there are no missing decision variables, constraints, parameters so that the whole scope of the problem is reached, and consequently there are no unfeasible solutions.

The formulation of an optimization problem requires the specification of the following components:

- **Objective functions**, which represent the value that is going to be optimized, whether maximized or minimized. The objective function f formulates the goal to be achieved. It associates with each solution of the the search space a real value that describes the fitness of the solution, $f : S \rightarrow R$. It represents an absolute value and allows a complete ordering of all solutions of the search space. The objective function will guide the search toward "good" solutions of the search space, and if it

is improperly defined, it can lead to non acceptable solutions whatever metaheuristic is used Talbi [2009].

- **Decision variables**, which represent the variables whose values can vary in order to increase or decrease the value of the objective function. These variables reflect aspects of the problem that the decision maker has control over and can take on discrete or continuous values depending on the context of the problem.
- **Constraints**, which represent any kind of limitation on the values that the decision variables can take. They are used to avoid unfeasible system responses Maier et al. [2019].

A solution is defined as a set of selected values of the decision variables, and a feasible solution is one that satisfies all problem constraints Maier et al. [2019].

2.1.2 Combinatorial Optimization Problems

Optimization problems divide naturally in two categories: those with continuous variables and those with discrete variables, which are called combinatorial. Continuous problems, consist of finding a set of real numbers or even a function, whilst the combinatorial problems consist of finding an optimal object from a finite set of objects, where the set of feasible solutions is discrete or can be reduced to a discrete set Papadimitriou and Steiglitz [1998]. Regarding the combinatorial optimization problems, the space of solutions is typically too large to search exhaustively using brute force.

Some famous examples of combinatorial optimization problems are:

- Travelling Salesman Problem
- Vehicle Routing Problem
- Minimum Spanning Tree
- Inventory Routing Problem
- Knapsack Problem

Inventory Routing Problem

The inventory routing problem can be described as a combination of vehicle routing and inventory management problems, in which a supplier has to deliver products to a number of geographically dispersed costumers, subject to side constraints. A generic representation of the Inventory Routing Problem is presented in Figure 3. Inventory control and routing decisions have to be made simultaneously. This approach provides integrated logistics solutions by simultaneously optimizing inventory management, vehicle routing and delivery scheduling according to the distribution policies of a given organization Coelho et al. [2014]. The objective of this approach is to minimize the costs of transportation and inventory holding whilst guaranteeing the storage capacity limitations are respected and stock-outs are avoided Baldacci et al. [2008].

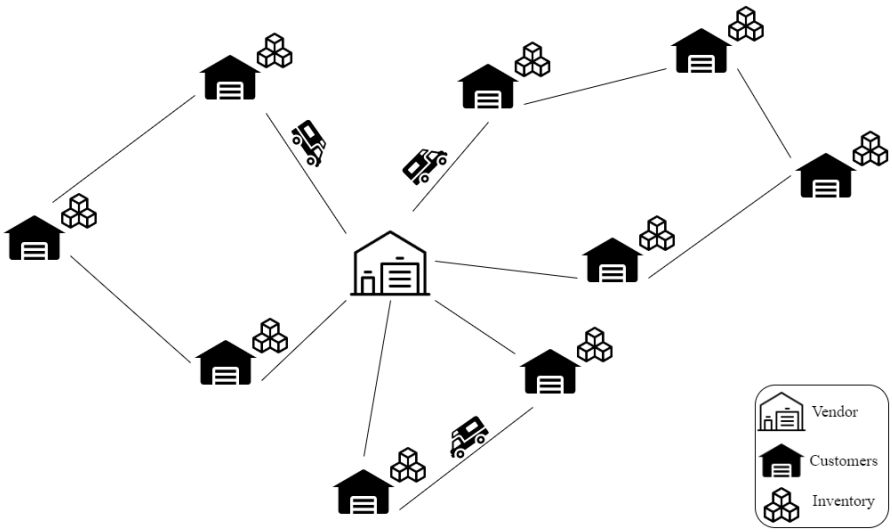


Figure 2: Generic representation of the Inventory Routing Problem

The Inventory Routing Problem arises from the context of the Vendor Managed Inventory, a business practice aimed at reducing logistics costs and adding business value. With this strategy, a supplier makes the replenishment decisions for products delivered to costumers, based on specific inventory and supply chain policies. This benefits the vendors, because they save on distribution costs since they are the ones to coordinate the shipments made to the customers and also benefits the buyers since they do not have to allocate resources and efforts in to inventory control Coelho et al. [2014].

Inventory routing problems share some basic characteristics with each other. They all consider environments in which products are shipped from a supplier to one or more costumers by means of, usually

capacitated, vehicles Baldacci et al. [2008]. In all of these type of problems, the costs are incurred, in part, from the distanced travelled by the vehicles, and these costs are included in the objective function. This characteristic stems from the routing component of the problem. The inventory component of the problem adds complexity to the routing component since product is consumed or moved to other entities over time, there is limited storage capacity and the supplier, whilst managing the inventory, has to ensure stock-outs and over stocking does not occur. The limited inventory holding capacity at the supplier or the customers has to be taken into account when deciding on delivery quantities and inventory holding costs, at the supplier or at the customers, may be incurred which as to be accounted for in the objective function Baldacci et al. [2008]. All these factors, influence and add complexity to the routing decisions.

The Inventory Routing problems can be classified according to some criteria, such as: time horizon, structure, routing, inventory policy, inventory decisions, fleet composition and fleet size. The following table 1, represents the possible options under the criteria of a given inventory routing problem:

Criteria	**Possible options**		
Time horizon	Finite	Infinite	
Structure	One-to-one	One-to-many	Many-to-many
Routing	Direct	Multiple	Continuous
Inventory policy	Maximum level (ML)	Order-up-to level (OU)	
Inventory decisions	Lost sales	Back-order	Non-negative
Fleet composition	Homogeneous	Heterogeneous	
Fleet size	Single	Multiple	Unconstrained

Table 1: Structural Variants of the Inventory Routing Problem. Source: Adapted from Coelho et al. [2014].

According to Coelho et al. [2014], the time refers to the horizon taken into account by the model, it can either be finite or infinite. The number of suppliers and customers may vary depending on the context, and therefore the structure can be one-to-one when there is only one supplier serving one customer, one-to-many, being the most common case, when there is one supplier serving multiple customers, and many-to-many, being the less frequent case, when there are multiple suppliers and multiple customers. Routing can be direct, when there is only one customer per route, multiple, when there are multiple customers per route, and continuous when there is no central depot. Inventory policies define pre-established

rules for replenishing customers. The two most common are the maximum level policy and the order-up-to level policy. Under a maximum level policy, the replenishment level is flexible but bounded by the capacity available at each customer. Under an order-up-to level policy, whenever a customer is visited, the quantity delivered is that to fill its inventory capacity. Inventory decisions determine how inventory management is modeled. If the inventory is allowed to become negative, then back-ordering occurs and the corresponding demand will be served at a later stage; if there are no back-orders, then the extra demand is considered as lost sales. In both cases there may exist a penalty for the stock-out. Finally, the last two criteria refer to fleet composition and size. The fleet can either be homogeneous or heterogeneous, and the number of vehicles available may be fixed at one, fixed at many, or be unconstrained.

According to Baldacci et al. [2008], there are a variety of characteristics and assumptions that may also change how the Inventory Routing Problem is approached, such as:

- Inventory holding costs may or may not be considered.
- Inventory holding costs may be charged at the supplier only, at the supplier and the customers, or at the customers only.
- the production and consumption rates can be deterministic or stochastic.
- production and consumption take place at discrete time instants or take place continuously.
- production and consumption rates are constant over time or vary over time.

In this system, the decisions to be made are:

- When to deliver to each costumer.
- How much to deliver to a costumer each time it is served.
- How to route the vehicles so as to minimize the total cost.

The objective of the problem is to minimize the total inventory-distribution cost while meeting the demand of each customer Coelho et al. [2014]. The replenishment plan is subject to the following constraints:

- The inventory level at each customer can never exceed its maximum capacity.

- Inventory levels are not allowed to be negative.
- The supplier's vehicles can perform at most one route per time period, each starting and ending at the supplier.
- Vehicle capacities cannot be exceeded.

The Inventory Routing Problem is NP-hard and most papers propose heuristics for its solution since these methods are able to obtain high quality solutions to large scale and complex optimization problems. There are also some papers that propose exact algorithms for its solution.

2.2 Exact methods

As mentioned before, exact methods exact methods obtain optimal solutions and guarantee their optimality. Before applying an exact or approximation method one must look at the optimization's problem characteristics, instance size, structure and complexity. Exact algorithms are generally more time expensive than approximation methods which in turn makes the latter more appropriate for NP-hard problems in most cases Jourdan et al. [2009]. However, exact methods can still be used in NP-hard problems when dealing with small sized instances depending on the structure of the problem Talbi [2009]. Exact methods are used more for smaller sized problems, where these type of algorithms can reach an optimal solution in reasonable time and when instances become too large for these methods, approximation methods such as heuristics and in particular metaheuristics are often used Jourdan et al. [2009]. From figure 4, it can be seen the exact algorithms that fall under the category of exact methods.

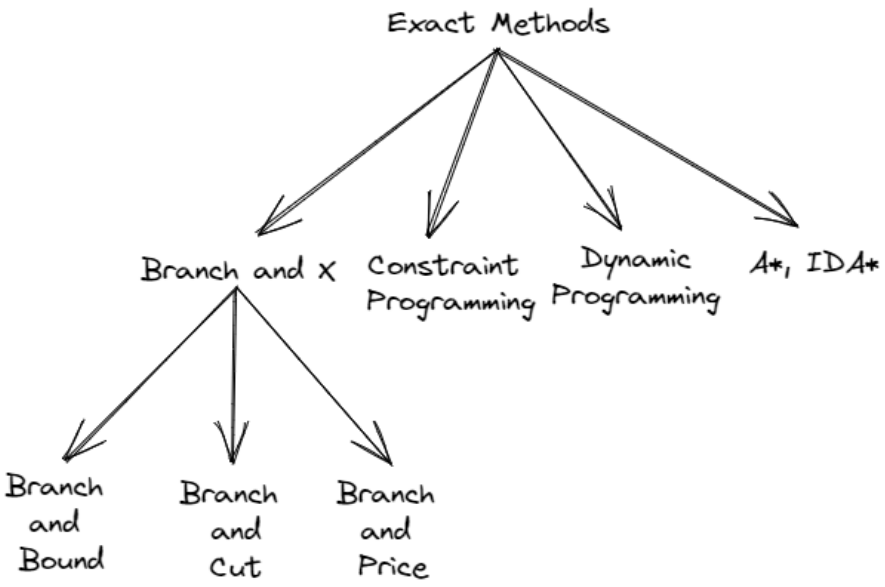


Figure 3: Diagram representation of the exact methods.

According to Talbi [2009], tree search algorithms such as Brand and X family and A* algorithms, Constraint Programming and Dynamic programming, search for a solution within the entire interesting search space. This is usually achieved by subdividing the initial problem into smaller and simpler ones.

Branch and Bound

The brand and bound algorithm is based on an implicit enumeration of all solutions of the considered optimization problem where the search space is explored by dynamically building a tree whose root node represents the problem being solved and its whole associated search space Talbi [2009]. The leaf nodes represent potential solutions and the internal nodes are subproblems of the total solution space. The subtrees that do not contain any optimal solution are pruned. In Theurich et al. [2021], two branch and bound algorithms were developed for the vehicle routing problem with customer costs where the first appends one job at the end of a route in each branching step and the second includes one job inside a route in each branching step. Both branch and bound algorithms were tested on 100 benchmarks instances and compared to the performance of the CPLEX solver. It was concluded that the branch-and-bound approaches could better handle time-dependent customer costs and both branch-and-bound algorithms solved these instances with significantly less computation time than the CPLEX.

Dynamic Programming:

Dynamic programming is based on the recursive division of a problem into simpler subproblems. It stores the solutions to these subproblems in memory as to avoid redundant work such as recalculating the same subproblems Talbi [2009]. The solutions to the subproblems are then reused and combined in order to find the solution to the original problem. In Xiao and Konak [2017], a genetic algorithm with exact dynamic programming was developed for the green vehicle routing and scheduling problem with heterogenous fleet and tardiness objective, named as the time-dependent vehicle outing and scheduling problem with CO2 emissions optimization in a time-varying traffic environment. The dynamic programming method was applied for the scheduling part, and found optimal schedules for all traveled arcs with discrete departure/arriving times and continuous to-be-traveled distance in different periods. The genetic algorithm with exact dynamic programming outperformed the restricted-CPLEX method in the three mentioned instances by 26.1%, 38.8% and 51.4%.

2.3 Approximate methods

A large number of real life optimization problems in a variety of areas such as science, engineering, economics are complex, large in scale and difficult to solve. These problems can not be solved in an exact manner within a reasonable time. To deal with this kind of problems, approximate methods are used. Approximate methods were developed in order to generate near-optimal solutions to optimization problems that could not be solved efficiently, namely NP-hard problems Gonzalez [2007]. If an optimal solution is not attainable, it is reasonable to settle for a good feasible solution that can be computed efficiently, in reasonable time. Ideally, one must sacrifice as little optimality as possible, while gaining as much as possible in efficiency. Trading-off optimality in favor of tractability is the paradigm of approximate methods Hochba [1997]. Approximate methods can be divided in to heuristic algorithms and approximation algorithms. Figure 4 shows all the components and sub-components of the approximate methods.

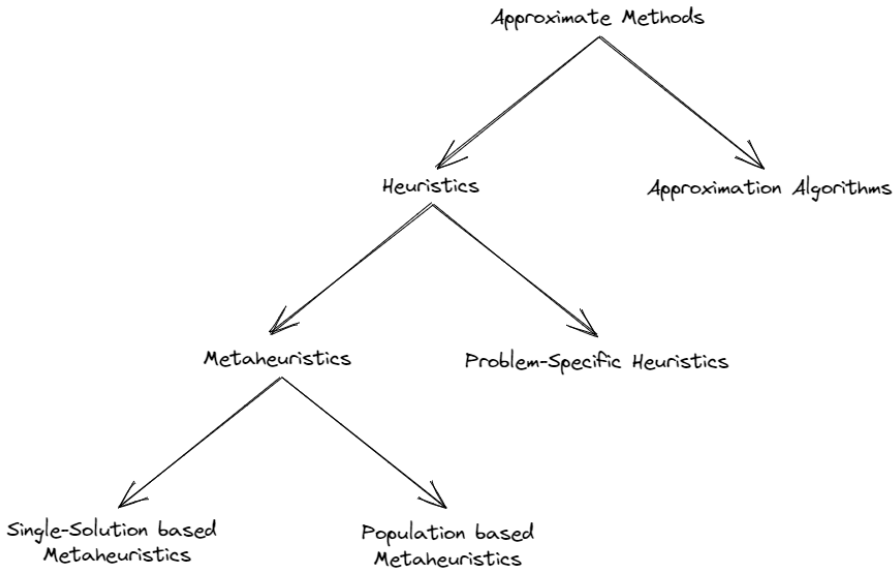


Figure 4: Diagram representation of the approximate methods.

2.3.1 Approximation Algorithms

Approximation algorithms provide provable solution quality and provable run-time bounds Talbi [2009]. Provable solution quality means that the algorithm in question can guarantee, with a certain probability or confidence level, that the approximate solution is withing a certain range or error of the optimal solution. This can be expressed as an approximation factor ϵ, which is a measure of the maximum relative error of the approximate solution generated by the algorithm. Provable run-time bounds refer to the maximum time that the algorithm will take to find the approximate solution, given a certain input size.

In approximation algorithms, there is a guarantee on the bound of the obtained solution from the global optimum Hochba [1997]. An ϵ-approximation algorithm generates an approximate solution *a* not less than a factor ϵ times the optimum solution *s* Vazirani [2001].

ϵ-Approximation Algorithms

According to Talbi [2009], an algorithm has an approximation factor ϵ if its time complexity is polynomial and for any input instance it produces a solution *a* such that [1]

[1] In a minimization context

$$a \leq \epsilon \cdot s \quad \text{if} \quad \epsilon > 1 \tag{2.1}$$

$$\epsilon \cdot s \leq a \quad \text{if} \quad \epsilon < 1 \tag{2.2}$$

where s is the global optimal solution, and the factor ϵ defines the relative performance guarantee. The ϵ factor can be a constant or a function of the size of the input instance.

An ϵ-approximation algorithm generates an absolute performance guarantee ϵ, if the following property is proven:

$$(s - \epsilon) \leq a \leq (s + \epsilon) \tag{2.3}$$

NP-hard problems differ in their approximability. The PTA (Polynomial-time Approximation) class is a well known set of approximation problems that can be approximated within any factor greater than 1.

PTAS (polynomial-time approximation scheme). A problem is in the PTAS class if it has polynomial-time (1 + ϵ)-approximation algorithm for any fixed ϵ > 0.

FPTAS (fully polynomial-time approximation scheme). A problem is in the FPTAS class if it has polynomial-time (1 + ϵ)-approximation algorithm in terms of both the input size and $1/\epsilon$ for any fixed ϵ > 0.

Some NP-hard problems are impossible to approximate within any constant factor (or even polynomial, unless P = NP) Talbi [2009].

The study of approximation algorithms can provide an insight to the inherent difficulty of a particular problem and the best ways to approach it. This type of study provides a mathematically rigorous basis on which to study heuristics. Typically, heuristics and metaheuristics alike are studied empirically, they might work well but sometimes hard to understand why. The field of approximation algorithms brings mathematical rigor to the study of heuristics, allowing to prove how well the heuristic performs on all instances, or giving some idea of the types of instances on which the heuristic will not perform well Williamson and Shmoys [2011]. However, approximation algorithms are specific to the target optimization

problem, they are problem dependent. Furthermore, in practice, attainable approximations can be too far from the global optimal solution, making them not useful for many real-life applications Talbi [2009]. Because of these reasons, this dissertation will focus more on heuristics, specifically metaheuristics since they are able to solve instances of complex problems by exploring a large solution search space and are applicable to a large variety of optimization problems.

2.3.2 Heuristics and Metaheuristics

The word heuristic has its origin in the old Greek word heuriskein, which means the art of discovering new strategies to solve problems. A heuristic represents trial and error, rule of thumb and educated guess approaches that allow making decisions or solve problems quickly and efficiently based on limited information and past experience. Unlike approximation algorithms, which provide provable solution quality and provable run-time bounds, heuristics find "good" solutions on large-size problems instances. They allow to obtain acceptable performance at acceptable costs in a wide range of problems Talbi [2009]. Heuristics can be further decomposed into specific heuristics and metaheuristics. Specific heuristics are designed to solve specific problems and instances, whilst metaheuristics are applicable to a large variety of optimization problems. The term metaheuristic was first introduced by Glover [1986] and combines the prefix meta, which means "upper level methodology" that refers to the higher level nature of these approaches that are used as guiding strategies in designing underlying heuristics to solve specific optimization problems, with the word heuristic. A formal definition of metaheuristic is proposed by Sörensen and Glover [2013]: "A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms.". In short, metaheuristics are high level strategies for exploring search spaces by using different methods.

According to Blum and Roli [2003] there are some fundamental properties that characterize metaheuristics:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore thesearch space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.

- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.

Intensification and diversification are the two general strategies and criteria that are used in metaheuristics to guide the search for solutions. These two strategies are both contrary and complementary of each other and largely determine the behaviour of a metaheuristic. Promising regions are determined by the obtained "good" solutions Talbi [2009]. Intensification refers to the process of focusing the search on a promising region of the solution space in order to find solutions with better quality. Diversification means expanding the search to cover a larger region of the solution in order to find new and potentially better solutions and ensuring that the search is not confined to a reduced number of regions. Intensification is the use of local knowledge of the search and solutions found so far so that new search moves can concentrate on the local neighborhood where the optimality may be close, however, this local optimum may not be the global optimality. Intensification tends to increase the speed of convergence, while diversification tends to decrease the convergence rate of the algorithm Yang et al. [2014]. Too much diversification increases the probability of finding the global optimality but with reduced efficiency, strong intensification can make the algorithm trapped in a local optimum. Diversification can be achieved by use of randomization Blum and Roli [2003], Yang [2010b], which enables an algorithm to have the ability to jump out of any local optimum so as to explore the search globally and can also be used for local search around the current best if steps are limited to a local region Yang et al. [2014]. It is crucial for the metaheuristics for diversification and intensification to be somewhat balanced as to avoid the algorithm being trapped in a local optimum and reduction of efficiency. Some algorithms may have intrinsically better balance among these two important components than other algorithms, that is one of the reasons why they may perform better Yang [2010b,a].

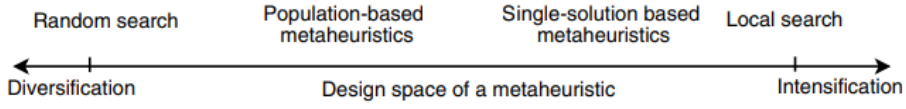


Figure 5: Two conflicting criteria in designing a metaheuristic: diversification versus intensification. Source: Talbi [2009].

When approaching an optimization problem, it is important to analyze the complexity of a problem, which gives an indication of its hardness, and know the size of input instances the algorithm in question is supposed to solve. These factors can give insight to knowing if a metaheuristic is the appropriate option to solve a given optimization problem. Even if a problem is NP-hard, small instances may be solved by an exact method. The structure of the instances plays an important role since some medium or even large-size instances with a specific structure may be solved in optimality by exact algorithms Talbi [2009]. The required search time also plays an important role in selecting an optimization algorithm for solving a problem. It is not optimal to use metaheuristics to solve problems where exact algorithms that can solve the problem efficiently are available. So for easier optimization problems, such as the polynomial class problems that can be solved with exact methods in reasonable time, one should avoid using metaheuristics. For these reasons, deciding which optimization algorithm to use to solve a problem should involve analyzing its complexity, the size of structure of the instances and the search time to solve it. If a problem can be reduced to a classical or an already solved problem in the literature, it is important to look at the state-of-the-art best known optimization algorithms solving the problem Talbi [2009].

Many of the combinatorial optimization problems belong to the NP-hard class of optimization problems. For this class of problems, in the worst case, exact algorithms require exponential time. The use of metaheuristics is justified for problems where exact algorithms cannot solve the handled instances' size and/or structure within the required search time. The notion of required time is dependent on the target optimization problem, for some problems an acceptable time may be equivalent to some seconds whereas for some other problems it may be a much larger time than that Talbi [2009].

According to Talbi [2009], the main characteristics of optimization problems justifying the use of metaheuristics are:

- An easy problem (Polynomial class) with very large instances. In this case, exact polynomial-time algorithms are known but are too expensive due to the size of instances.
- An easy problem (Polynomial class) with hard real-time constraints. In real-time optimization problems, metaheuristics are widely used. Even if efficient exact algorithms are available to solve the problem, metaheuristics are used to reduce the search time.
- A difficult problem (NP-hard class) with moderate size and/or difficult structures of the input instances.

- Optimization problems with time-consuming objective function(s) and/or constraints. Some real-life optimization problems are characterized by a huge computational cost of the objective function(s).
- Nonanalytic models of optimization problems that cannot be solved in an exhaustive manner.
- Moreover, those conditions may be amplified by nondeterministic models of optimization: problems with uncertainty and robust optimization. For some noisy problems, uncertainty and robustness cannot be modeled analytically.

Metaheuristics can be classified according to some criteria Talbi [2009]:

- **Population-based search versus single-solution based search:** Single-solution based algorithms manipulate and transform a single solution during the search while in population-based algorithms a whole population of solutions is evolved. These two families have complementary characteristics: single-solution based metaheuristics are exploitation oriented, they have the power to intensify the search in local regions. Population-based metaheuristics are exploration oriented, they allow a better diversification in the whole search space.
- **Nature inspired versus non-nature inspired:** Many metaheuristics are inspired by inspired or based on natural processes: Evolutionary algorithms that refer to the mechanisms of biological evolution such as reproduction, mutation, recombination, selection. Particle Swarm Optimization simulates the behaviour of a group of birds (swarm) looking for food. Industrial processes like simulated annealing come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to alter its physical properties.
- **Deterministic versus stochastic:** Deterministic metaheuristics follow a predetermined set of rules or behaviours, their output can be accurately predicted based on their input. Stochastic metaheuristics involve elements of randomness or probability in their search process. Their output can not be accurately predicted based on their input, the same input can produce different results over time.
- **Memory usage versus memoryless methods:** Memory usage metaheuristics use some forms of memory to store information about past search states and solutions. Memoryless metaheuristics do not rely on and do not store information about past search states and rely solely on the current state of the search process to guide the search.

- **Iterative versus Greedy:** Iterative metaheuristics start with an initial solution and repeatedly make incremental changes to the solution in order to improve it through a series of iterations. Greedy metaheuristics start with empty solutions and make decisions based on the most immediatly benificial option at each step without considering the long-term consequences of those choices.

Single-solution based Metaheuristics

Single solution based metaheuristics focus on the improvement of a single solution at a time by performing iterative procedures that move from the current solution to another one in the search space. These metaheuristics iteratively improve a solution by applying generation and replacement procedures from the current single solution. In the generation phase, there is a neighborhood or set of candidate solutions *C(s)* generated from the current solution *s* which is generally obtained by applying local transformations to the current solution. In the replacement phase, a candidate solution is selected to be the new solution, that is, $s' \epsilon C(s)$ Talbi [2009]. This whole process is iterated until a stopping criteria is satisfied. This process is illustrated by Figure 6.

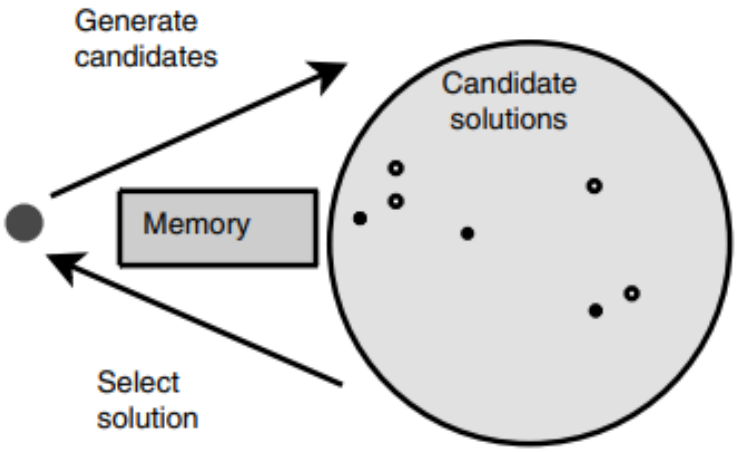


Figure 6: Concept of the Single-solution based Metaheuristics. Source: Talbi [2009].

Single solution based metaheuristics, generally follow a structure that consists of the following phases: initialization, generation, evaluation, selection, termination. The performance of the single solution based metaheuristics is highly dependent on the way these phases are approached and the type of problem at hand. The specific details of how these phases are approached depend on the problem being solved and the chosen metaheuristic. The common search concepts for single solution based metaheuristics are the

definition of the neighborhood structure and the determination of the initial solution Talbi [2009].

In the initialization phase an initial solution is generated. The initial solution can have a significant impact on the performance of single solution based metaheuristics since they rely on a single current solution to generate the neighborhood and choose the next current solution and so on. If the initial solution is of high quality, the algorithm is more likely to find a good solution efficiently, however if the initial solution is not of good quality the algorithm may get stuck in a local minimum and not able to find a good solution. There are two main strategies that are used to generate an initial solution for these type of metaheuristics: a random and a greedy approach. Ultimately, the choice between a greedy and a random initial solution will depend on the specific problem that is being tackled, the characteristics of the metaheuristic being used and the trade off in terms of the quality of solutions and computational time of both of these approaches. The larger the neighborhood, the less is the sensitivity of the initial solution to the performance of single solution based metaheuristics Talbi [2009]. Generating a random initial solution is a fast operation, however the metaheuristic may take a much larger number of iterations to converge, and in that case, a greedy heuristic may be used in order to speed up the search. Greedy heuristics often lead to a better quality local optima and have a reduced polynomial-time complexity in most cases. However, it does not mean that using better initial solutions will always lead to better local optima and that using greedy initial solutions is always better. In the case of a long term performance of a metaheuristic, a greedy initial solution may lead an algorithm to a local minimum, making it perform poorly in the long run. Whereas the random initial solution has a better chance of avoiding local minimums and might even surpass the greedy approach in the long run in terms of solution quality and search time. Despite that, the random strategy may generate high deviation in terms of obtained solutions. In order to bridge that gap and improve the robustness, a hybrid approach that combines both random and greedy strategies can sometimes be used.

In the generation phase, a neighborhood of new solutions is generated from the current solution. A neighbor is generated by the application of a move operator m that performs a small perturbation to the solution s. The definition of the neighborhood is a required common step for the design of any single solution based metaheuristic. According to Talbi [2009], the definition of a structure of a neighborhood is crucial in the performance of a single solution based metaheuristic and if not defined adequately to the problem at hand no single solution based metaheuristic will be able to solve it. The main property that characterizes the neighborhood is locality which is the effect on the solution after performing the move operator in the representation. Strong locality is when small changes made in the representation reveal small changes

to the solution and weak locality is when small changes made in the representation reveal big changes to the solution. Locality can be used as metric to guide the search towards promising areas of a search space, to ensure that the solution being evaluated is not too divergent from the current solution. The way the structure of a neighborhood is defined, the move operators used, and its size have a big impact on the performance of the metaheuristic. A larger neighbourhood allows to explore a wider range of solutions and may improve the quality of the obtained solutions but at the cost of increasing computational costs. So, in designing a single solution metaheuristics there is a trade off between the size and quality of a neighborhood and the computational complexity to search it.

In the evaluation phase, the candidate solutions are evaluated. This may involve calculating a fitness or objective function for each of these solutions in order to measure which one is the most appropriate to select. In the selection phase, the best candidate solution is selected as the next current solution. This process may also involve applying a local search phase where the neighborhood can be expanded in an effort to escape a local optimum.

In the termination phase, the metaheuristic terminates its search and stops iterating the previous phases when it reaches a stopping criterion, that may be one such as finding a satisfactory solution or reaching the maximum number of iterations established or some other criteria.

Local Search:

Local Search consists in finding the maximum or minimum value of a function within a specific region, rather than the global minimum or maximum over the entire domain. It starts with an independently obtained initial solution constructed by some heuristic algorithm. At each iteration, the heuristic will replace the current solution by a neighbor that improves the value of the defined objective function searching through the solution space that way Michiels et al. [2007]. Typically, the search stops when all the candidate solutions in the neighborhood are worse than the current solution and thus reaching a local optimum. Variants of the local search method are distinguished in the way the neighboring solutions are generated (deterministic or stochastic) and the selection strategy of the new current solution Talbi [2009].

According to Talbi [2009], in addition to the definition of the initial solution and the neighborhood, designing a local search algorithm has to address the selection strategy of the neighbor that will determine the next

current solution. Regarding the selection of the neighbor, there are some strategies:

- **Best Improvement:** The best neighbor, the neighbor which has the best value for the specific objective function, is selected. This makes the exploration of the neighborhood exhaustive since all possible moves are tried for a solution to select the best neighboring solution. This kind of exploration might be too time-consuming for large neighborhoods.
- **First improvement:** This strategy consists in choosing the first better neighbor and then selecting it to replace the current solution. This strategy aims for a partial evaluation of the neighborhood. In the worst case, there is a total evaluation of the neighborhood.
- **Random selection:** A random selection is applied to the neighbors that improve the solution.

There is a compromise in terms of quality of solutions and search time when using the first improvement strategy when the initial solution is randomly generated and the best improvement when the initial solution is generated using a greedy process. According to Talbi [2009], it has been observed that the first improving strategy leads to the same quality of solutions as the best improving strategy while using a smaller computational time and the probability of premature convergence to a local optima is less important in the first improvement strategy.

Even though local search is an easy method to understand and implement, it has a few disadvantages. The main one being the convergence towards the local optima. It also is highly dependent on the quality of the initial solution, there is no means to estimate the relative error from the global optimum and the number of iterations performed might not be known in advance. Local search is best used for instances where there are not too many local optima in the search space and the quality of the different local optima is similar. In order to tackle its main disadvantage, many alternative algorithms and variations of local search have been proposed, such as iterative local search, tabu seach and GRASP. In Singh et al. [2015], local search is used to solve an inventory routing problem occurring in bulk industrial gas distribution. An incremental approach is applied where an inventory routing problem instance is divided into many sub-problems to improve the solution quality where each sub-problem uses an initial solution obtained from solving the previous sub-problem using local-search heuristic using incremental approach. The results show a significant improvement in cost to deliver unit product called logistics ratio for the test data set.

Tabu Search:

Tabu Search was designed to manage an embedded best improvement local search algorithm and escape the local minima by explicitly using the history of the search, by means of short term memory and allowing non-improving moves and returning to previously visited solutions Blum and Roli [2003], Boussaïd et al. [2013], Burke et al. [2014]. The main characteristic of Tabu Search is that unlike Simulated Annealing it uses memory and is able to learn from the past.

This method uses a tabu list that records the last encountered solutions or important attributes of those and forbids these solutions from being visited again, as long as they remain on this same list, this being the short memory component of the algorithm. The tabu list prevents endless cycling and forces the search to accept non-improving solutions in order to escape local minima. The length of the tabu list or tabu tenure constrols the memory of the search process. If the tabu tenure is small the search concentrates on small areas of the search space and if large it will concentrate on larger areas of the search space because it forbids visiting a higher number of solutions Blum and Roli [2003]. The tabu tenure can be varied during the search which leads to a more robust algorithm.

There are some additional medium-term memory structures that can be introduced to bias moves towards promising areas of the search space (intensification), as well as long-term memory structures to encourage a broader exploration of the search space (diversification) Boussaïd et al. [2013]. Medium-term memory structures such as aspiration criteria, greatly improve the search process. The use of tabu lists can prevent attractive moves, even if there is no risk of cycling, and may lead to overall stagnation of the process. The aspiration criteria are a set of rules that are used to override tabu restrictions. If a move is forbidden by the tabu list it can still be allowed if the aspiration criteria are satisfied.

A long-term memory or frequency memory can be used in order to record how often certain attributed have been encountered in solutions on the search trajectory Boussaïd et al. [2013]. It allows the search to avoid visiting solutions that present the most encountered attributes or to visit solutions with attributes rarely encountered and thus benefiting diversification.

Search Memory	Role	Popular Representation
Tabu list	Prevent cycling	Visited solutions, moves attributes Solution attributes
Medium-term memory	Intensification	Recency memory
Long-term memory	Diversification	Frequency memory

Figure 7: The different search Memories of the Tabu Search. Source: Talbi [2009].

In Cortés et al. [2017], a tabu search metaheuristic was developed to solve the picking routing problem for large and medium size distribution centres considering the availability of inventory and heterogeneous material handling equipment. The problem is solved with three metaheuristics based on tabu search. The first metaheuristic is a generic tabu search, whereas the second and third are a hybrid tabu search that integrates two diversification strategies called 2-Opt Exchange and 2-Opt Insertion. The results produced by the tabu search approaches are compared with two benchmark approaches: Genetic Algorithm, Simulated Annealing. TS 2-Opt Insertion generates the best solution and appears to be a very appropriate approach to the picking routing problem considering K heterogeneous material handling equipment because it outperforms not only a wide variety of soft computing approaches such as the generic tabu search implementations but also other refined implementations such as TS 2-Opt Exchange or even others based on soft computing algorithms such as genetic algorithm and simulated annealing.

Iterated Local Search:

The quality of the local optima obtained by a local search method depends on the initial solution and as local optima with high variability is generated, iterated local search can be used to improve the quality of successive local optima Talbi [2009]. Iterated local search is based on the idea of instead of repeatedly applying a local search procedure to randomly generated starting solution, generating the starting solution for the next iteration by perturbing the local optimum found at the current iteration and this is done in the hope that this perturbation mechanism provides a solution located in the basin of attraction of a better local optimum Boussaïd et al. [2013]. If the perturbation is too weak, it may not be sufficient to escape the basin of attraction of the current local optimum and generate cycles in the search causing no gain, too strong of a perturbation would result in erasing the information about the search memory and the good properties of the local optimum are skipped.

Another key aspect of the Iterated Local Search is the acceptance criteria. The role of the acceptance

criterion combined with the perturbation mechanism is to enable the control of the trade off between intensification and diversification. It defines the conditions that the new local optimum has to satisfy in order to replace the current local optimum. An extreme solution in terms of intensification is to accept only improving solutions and the extreme solution in terms of diversification would be to accept any solution without any regards to its quality. The optimal acceptance criteria would be one that balances intensification and diversification.

In Vansteenwegen and Mateo [2014], an iterated local search metaheuristic is developed for the Single-Vehicle Cyclic Inventory Routing Problem. In this article, the iterated local search metaheuristic exploits typical characteristics of the problem to reduce the computation time. Experimental results on 50 benchmark instances show that the algorithm improves the results of the best available algorithm on average by 16.02% and 32 new best known solutions are obtained.

Simulated Annealing:

The simulated annealing algorithm is inspired by the annealing process in metallurgy, where a substance undergoes heating and then slowly cooling in order to obtain a stronger structure. It simulates energy changes to a substance subjected by way of slow cooling until it reaches an equilibrium state and converges. It is a stochastic algorithm that at its core is subject to an acceptance criterion, which determines if a new solution is accepted or not, which in turn enables, under some conditions, the degradation of a solution. This is done in order to escape local optima and delay convergence. Unlike the tabu search algorithm, for example, simulated annealing is a memoryless algorithm since it does not use previously gathered information during the search.

During each iteration of the algorithm, a random neighbor from the current solution is generated. The moves that improve the current solution's fitness value will always be accepted and thus a new current solution is chosen. If a neighbor with a worse fitness value is generated, it will be selected with a given probability that depends on the current temperature and the difference in the fitness value of the current and generated neighboring solution. The algorithm initially has a higher probability of accepting worse solutions, but as the temperature decreases, the algorithm becomes more selective and favors solutions with better fitness value as the probability of accepting worse solutions also decreases, striking a balance between diversification and intensification when exploring the search space. This probability or acceptance

criterion follows the Boltzmann distribution:

$$P(\Delta E, T) = e^{-\frac{f(s')-f(s)}{T}} \tag{2.4}$$

Along with the acceptance criteria, the cooling schedule is the other aspect of simulated annealing that has the most impact on its success. There are a few parameters to consider: the initial temperature, cooling rate and stopping temperature. If the starting temperature is too high, the search can become more or less a random local search, and if too low it will be more or less a first improving local search algorithm, so it is of important that the initial temperature is not too high but high enough to be able to allow moves to almost neighborhood state Talbi [2009]. The cooling rate value encompasses a trade off between quality of the obtained solutions and the convergence rate or computational cost. The slower the temperature is decreased the better solutions will be found but at the cost of a higher computational time. The stopping temperature must go low enough so that when the the search is stopped the probability of accepting a move is negligible so that during the search we explore both diversification and intensification properly.

In Bent and Van Hentenryck [2004], a two-stage hybrid algorithm for the vehicle routing problem with time windows is developed where a simulated annealing algorithm is used in order to minimize the number of routes and vehicles used and then the travel cost by using a large neighborhood search, where the algorithm's effectiveness was demonstrated through the results, showcasing the ability to improve 13 out of the 58 best published solutions to the Solomon benchmarks and also matching or improving the best solutions in 47 problems.

Population-based Metaheuristics

Unlike single-solution based metaheuristics, population based metaheuristics deal with a set or population of solutions rather than a single solution. These type of metaheuristics start from an initial population of solutions, then iteratively apply the generation of a new population and the replacement of the current population. In the generation phase a new population of solutions is created and in the replacement phase a selection is carried out from the current and new populations Talbi [2009], which means that, taking into account the chosen selection methods and the desired attributes of the solutions, the result

of the current iteration will be a set of the best solutions from both populations. This process will iterate until a stopping criteria is satisfied. The generation and replacement phases in some population-based metaheuristics may be memoryless, which means these two procedures are only based on the current population. If not memoryless, some history of the search stored in memory can be used in the generation of new population and in the replacement of the old. Population-based metaheuristics differ from each other in the way they perform the generation and selection procedures and the search memory they are using during the search. These type of metaheuristics focus more on diversification of the search space, due to the large diversity of initial populations, where as single-solution based metaheuristics are more focused on intensification Talbi [2009].

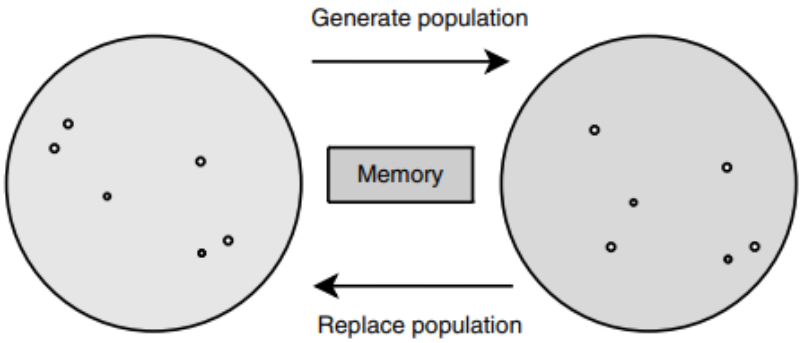


Figure 8: Main principles of population-based metaheuristics. Source: Talbi [2009].

The common search concepts for population-based metaheuristics are the determination of the initial population and the definition of the stopping criteria Talbi [2009]. The initial population and stopping criteria are the key components of this search concept since they determine the starting and ending point of the exploration of the search space regarding, used to establish the limit and scope of the search and influence the effectiveness and performance of the algorithm. These two components are chosen based on the characteristics of the optimization problem at hand. Regarding the generation of the initial population, the main criterion to deal with is diversification because if the initial population is not well diversified it may lead to a premature convergence for any population-based metaheuristic, this might happen when, for example, the initial population is generated by a single-solution based metaheuristic or a greedy heuristic for each solution of the population Talbi [2009]. Strategies dealing with the initialization of the population can be classified into four categories:

- **Random Generation:** This means that the initial population can be generated randomly. The random generation can be performed according to pseudo-random numbers or a quasi-random

sequence of numbers. Since it is impossible to generate algorithmically truly independent random numbers Talbi [2009], pseudo-random number generators algorithms are used to generate a sequence of number that approximate the properties of true random numbers. However, care should be taken so that pseudo-random generators provide good properties, so that the results are not highly correlated which leads to a poor exploration of the search space. Quasi-random number generators are methods with goals related not only to the independence between the successive numbers but also their dispersion. In quasi-random sequence, the diversity of the population is generally better than in pseudo-random generation, since they provide more coverage of the search space.

- **Heuristic Initialization:** Any heuristic can be used in order to initialize the population. In the case of population-based metaheuristics, it is important to use a metaheuristic that does not limit too much the diversification of the population. A greedy algorithm, for example, may or not be a better option than a random one for generating an initial population in the case of a single solution based metaheuristic but if it was used in the context of a population-based metaheuristic, it might not be a good approach since it could lead to a loss of diversity, which is crucial for the performance of population-based metaheuristics.

- **Sequential Diversification:** The initial population can be uniformly sampled in the decision space. In sequential diversification, the solutions are generated in sequence in such a way that the diversity is optimized.

- **Parallel Diversification:** Parallel diversification is a method where multiple solutions are independently and simultaneously modified to generate a diverse set of solutions.

From the results displayed in Figure 9, it is observable that sequential and parallel diversification strategies provide in general the best diversity, followed closely by the quasi-random approach. The heuristic initialization provides the best solutions but with a high computational cost and low diversity. However, this will be highly dependent on the characteristics of the optimization problem, heuristic and fitness landscape of the tackled optimization problem.

Strategy	Diversity	Computational Cost	Quality of Initial Solutions
Pseudo-random	++	+++	+
Quasi-random	+++	+++	+
Sequential diversification	++++	++	+
Parallel diversification	++++	+++	+
Heuristic	+	+	+++

Figure 9: Analysis of the different Initialization Strategies. Source: Talbi [2009].

Regarding the stopping criteria, there are many that can be used, some of them very similar to the single-solution based ones. According to Talbi [2009], population-based metaheuristics have two types of procedures:

- **Static Procedure:** The end of the search is known a priori. Examples of this are using a fixed number of generations, placing a limit on CPU resources or even defining a maximum number of objective function evaluations.
- **Adaptive Procedure:** The end of the search is not known a priori. One can use a fixed number of generations without improvement, when an optimum or satisfactory solution is reached.

There are some stopping criteria specific to population-based metaheuristics and they are mostly related to the diversity of the population. It consists in stopping the criteria when the diversity measure falls bellow a designated threshold.

Evolutionary Algorithms

Evolutionary Algorithm is the general term for several optimization algorithms that are inspired by the Darwinian principles of nature's capability to evolve living beings well adapted to their environment, natural selection and genetics Boussaïd et al. [2013]. Genetic algorithms, evolution strategies, evolutionary programming and genetic programming are all domains grouped under the term of Evolutionary Algorithms. Evolutionary algorithms are based on the notion of competition and survival of the fittest and represent a class of iterative optimization algorithms that simulate the evolution of a population of individuals Talbi [2009]. Every iteration of these type of algorithms corresponds to a generation, where a population of candidate solutions to a given optimization problem, called individuals, is capable of reproducing and is subject to genetic variations followed by the environmental pressure that causes natural selection or survival of the fittest Boussaïd et al. [2013]. Initially, the population of individuals is usually generated randomly. An objective function associated a fitness value with every individual of the population which

indicates the individual's suitability to the problem at hand. At each iteration, the individuals with better fitness have a higher probability to be selected as parents that will generate new offspring through some variation operators such as crossover and mutation. Then, a replacement scheme is applied in order to determine which individuals of the population will survive from the parents and their offspring, a generation of a new population. This process will iterate until a stopping criteria is satisfied. Figure 10 illustrates the procedures during a generation in evolutionary algorithms.

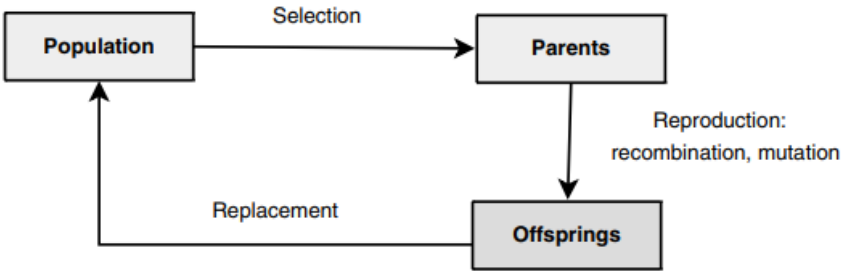


Figure 10: A generation in evolutionary algorithms. Source: Talbi [2009].

Genetic Algorithms:

In genetic algorithms, the basic implementation can be very generic and there are many aspects that can be implemented differently according to the problem at hand: representation of the solution (chromosomes), selection strategy, the type of crossover and mutation operators Boussaïd et al. [2013]. Traditionally, genetic algorithms use a binary or discrete representation of solutions and the initial population is generated randomly, in an attempt to cover the entire range of possible solutions or search space and promoting diversity. During each successive generation, a portion of the existing population is selected through a fitness function to breed a new generation. The fitter solutions are the ones who are more likely to be selected, it works as a probabilistic selection. The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover, and/or mutation. Emphasis is mainly concentrated on crossover as the main variation operator, that combines multiple (usually two) individuals that have been selected together by exchanging some of their parts, where an exogenous parameter called crossover rate indicates the probability per individual to undergo crossover Boussaïd et al. [2013]. After crossover, individuals can be subjected to mutation. Mutation introduces a degree of randomness into the search in order to prevent the optimization process getting stuck in local optima and promoting diversity. It consists in performing a slight perturbation to the resulting solution, usually with a low probability Boussaïd et al. [2013]. Then, starts the replacement procedure acting as the survival of the fittest in order to identify the parents and children to maintain for successive generations and assure

the survival of the fittest individuals. This is accomplished by evaluating each of these individuals with a defined fitness function. This generational process is repeated until a termination condition has been reached. The most common stopping criteria being when a good quality solution is reached or the number of maximum generations is met. According to Kumar et al. [2010], these are some of the possible stopping criteria for genetic algorithms:

- A solution is found that satisfies minimum criteria;
- Fixed number of generations reached;
- Allocated budget (computation time/money) reached;
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results;
- Manual inspection;
- Combinations of the above.

Genetic algorithms are characterized as being a stochastic algorithm, since randomness plays an essential role in this type of algorithms as a way of promoting diversity on the search space. Both the selection, crossover and mutation utilize random procedures. According to Sivanandam and Deepa [2008], genetic algorithms are robust algorithms that can be used in a wide variety of problems and can provide good results to complex and large instance problems. However, since they are extremely general, in some situations, specific techniques for solving particular problems can out perform genetic algorithms.

In Park et al. [2016], a genetic algorithm for the inventory routing problem with lost sales under a vendor-managed inventory strategy in a two-echelon supply chain comprised of a single manufacturer and multiple retailers is developed. The proposed genetic algorithm is compared with the CPLEX optimization model, and demonstrated solutions that in the case of small sized problems the results are very similar to those of the CPLEX and also showed solutions that remained within 3.2% of those obtained using the optimization model for large problems, requiring a much shorter computation time of 7 to 47 seconds.

Scatter Search:

Scatter Search is an evolutionary and population-based metaheuristic that recombines solutions selected from a reference set to build others Laguna et al. [2003]. This method starts by generating an initial

population satisfying the criteria of diversity. Then, the reference set is constructed by selecting good representative solutions from the population. The notion of best is not limited to a measure given exclusively by the evaluation of the objective function but covers the diversity of solutions, for example, a solution may be added to the reference set if the diversity of the set improves even when the value of the solution is inferior to that of other competing solutions Boussaïd et al. [2013]. These selected solutions are then combined to provide starting solutions to an improvement procedure based on a single-solution based metaheuristic or other search intensification procedure Talbi [2009]. According to the results from the mentioned procedure, the reference set and the population of solutions are updated to incorporate both diverse and high quality solutions. This whole process is iterated until a stopping criteria is satisfied, for example, until the reference set does not change anymore Boussaïd et al. [2013].

In Soman and Patil [2020], a scatter search metaheuristic for a heterogeneous fleet vehicle routing problem with release and due dates in the presence of consolidation of customer orders and limited warehousing capacity is developed. In this article, it is concluded that the use of adaptive memory, problem specific improvement and strategic oscillation in the scatter search frame work improves the quality of the solutions considerably, such that it outperforms the iterated local search method that does not consider those factors. The scatter search developed also finds solutions in shorter time when compared to the CPLEX otimization.

Blackboard-based:

In blackboard-based algorithms, the solutions of the population have a part in the construction of a shared memory, and this shared memory will be the main input in the process of generating new populations of solutions Talbi [2009]. Unlike evolutionary based algorithms, blackboard-based algorithms generate new populations based on the shared memory of the previous generations and not from the attributes of the solutions' parents. Some blackboard-based algorithms are:

Ant Colony Optimization:

Ant Colony Optimization is based on the cooperative and foraging behaviours of ant colonies. When searching for food, the ants initially explore the area surrounding their nest by performing a randomized walk and along their path the ants deposit a pheromone trail on the ground in order to mark some favorable path that can guide other ants to the food sources Dorigo and Blum [2005]. After some time, the shortest path between the nest and the food source presents a higher concentration of pheromones, which in turn,

attracts more ants. This chemical substance has a decreasing action over time and the quantity left by one ant depends on the amount of food Talbi [2009]. This type of metaheuristic takes advantage and exploits the characteristics of the behaviour of ant colonies in to optimization problems and exchange information on their quality through a communication scheme similar to the one of adopted by real ants Dorigo et al. [2006]. The algorithm is composed by two main steps after initializing the pheromone trails:

- **Solution Construction using the pheromone trail:** The construction of solutions is done according to a probabilistic state transition rule. Artificial ants can be considered as stochastic greedy procedures that construct a solution in a probabilistic manner by adding solution components to partial ones until a complete solution is derived and the target optimization problem can be seen as decision graph where an ant will construct a path Talbi [2009]. This process will iterate and will take into account the following:
  - **Pheromone trails:** The pheromone trails will memorize the characteristics of the best generated solutions, that in turn, will guide the construction of new solutions. The pheromone trails will change during the search as more knowledge is acquired and this change represents the shared memory of this whole ant search process.
  - **Problem-dependent heuristic information:** Problem dependent information can give the ants more clues for their decision process and construction of solutions.
- **Update the pheromone trails:** Updating the pheromone trails is done in two phases:
  - **Evaporation phase:** In an evaporation phase, the pheromone trails decreases automatically. The goal of this phase is to avoid for all ants a premature convergence towards good solutions and to encourage diversification in the search Talbi [2009].
  - **Reinforcement phase:** In this phase, the pheromone deposit is usually applied after all ants have finished constructing a solution. The pheromone values are increased on solution components that are associated with a chosen set of high quality solutions and the goal is to make these solution components more attractive for ants in the following iterations Boussaïd et al. [2013].

This whole process will iterate until a stopping criteria is satisfied. In Li et al. [2019], an improved ant colony optimization algorithm is developed for the green vehicle routing problem with multi depot and multiple objectives. The improved ant colony optimization model developed in this research use an innovative approach regarding the update of the pheromones that results in higher quality solutions when compared

to the conventional ant colony optimization model.

Particle Swarm Optimization:

Particle Swarm Optimization is a stochastic metaheuristic that mimics the social behaviour of natural organisms such as bird flocking and fish schooling. In this metaheuristic, a population of potential solutions, called particles, are stochastically initialized and then moved through the search space, adjusting their position towards the global optimum guided by the current best position found by the entire population as well the best position found by each individual particle. Each particle is a candidate solution to the problem, and is represented by a velocity, a location in the search space and has a memory which helps it remembering its previous best position Boussaïd et al. [2013]. Optimization takes advantage of the cooperation between the particles, the success of some particles will influence the behaviour of their peers. In Belmecheri et al. [2013], a particle swarm optimization algorithm with local search is developed for the vehicle routing problem with heterogenous fleet, mixed backhauls and time windows. In this article, it was shown that the developed particle swarm optimization algorithm had high quality solutions for small and large problems and it was able to, in some instances, improve the distance to the optimal solution by over 5.6% when comparing its results with an exact method and best known solutions of the literature regarding the vehicle routing problem with time windows.

2.3.3 Hybrid Metaheuristics

As mentioned before, some metaheuristics focus on intensification while others focus on diversification strategies in order to guide the search for solutions. Both these strategies are contrary and complementary of each other. Hybrid metaheuristics allow to combine or use metaheuristics with different optimization methods in order to balance and deal with the trade-of between intensification and diversification strategies. Hybridization allows enhancing the strengths and compensating the weaknesses of two or more methods with the aim of generating better solutions by combining the key elements of competing methodologies Baños et al. [2013]. According to Talbi [2009], hybrid metaheuristics, at a first level, can be distinguished as low-level and high-level hybridizations:

- **Low-level:** Low-level hybridizations address the functional composition of a single-optimization method where a given function of a metaheuristic is replaced by another metaheuristic.
- **High-level:** In High-level hybridizations the different metaheuristics are self-contained, which means there is no direct relationship to the internal workings of a metaheuristic.

On a second level:

- **Relay hybridization:** In relay hybridization, a set of metaheuristics is applied sequentially, each using the output of the previous one as input, reminiscent of a pipeline.
- **Teamwork hybridization:** Teamwork hybridization represents cooperative optimization models in which cooperating agents evolve in parallel, where each one carries out a search in the solution space.

Which results in the following hierarchical classification:

- **Low-level relay hybrid:** This class of hybrids represents algorithms in which a given metaheuristic is embedded into a single solution-based metaheuristic.
- **Low-level teamwork hybrid:** This class represents algorithms in which metaheuristics are embedded into population-based metaheuristics. Such as embedding single solution-based metaheuristics, which are good at intensification, into population-based metaheuristics, which in turn are good in diversification.
- **High-level relay hybrid:** - In which the self-contained metaheuristics work in sequence where the first ones' output represents the seconds' input.
- **High-level teamwork hybrid:** This scheme involves several self-contained algorithms performing a search in parallel, cooperating to find an optimum.

Chapter 3

Implementation

In this chapter, the methodology, which involves the whole implementation process of this dissertation, from the requirement elicitation to the testing of the optimization algorithms. A more in depth description of the problem's context, characteristics and its challenges, mathematical formulation and the optimization algorithms made are presented. Important and necessary steps to implement in order to find solutions for the described problem at hand.

3.1 Methodology

In order to achieve the objectives of this dissertation and taking into account the characteristics and description of the modeled problem, a methodology is proposed:

- Requirement elicitation with domain experts.
- Describe the main concepts of the optimization problem, such as: representation of the variables, objective function, constraint handling according to the state of the art and the domain's expert information and the characteristics of the problem at hand.
- Implementation and development of appropriate optimization algorithms and evaluation and comparison of their performance with attention to the problem's characteristics and the machine used to run these algorithms.

After a study of the background and state of the art of the whole process of optimizing problems, with emphasis on the inventory routing problem, a requirement elicitation with the domain experts was realized in order to identify the key aspects of the problem, the characteristics and specific objectives of the problem in the detail. This problem was then modeled and a mathematical formulation of the problem was performed. This formulation is a crucial step in the context of an optimization problem that, once completed, will allow for the implementation of the appropriate optimization algorithms.

3.2 The problem and its challenges

As mentioned before, in this project it is considered that the stores of the same retailer group are connected in a network and are monitored by a central system. The problem arises in the context of the network's stock disruption when other outside entities are not able to respond to the stores' supply requests and thus the stores become the entities responsible for requesting and delivering products to each other. This system identifies the network's stores that are in need of stock of a given product to satisfy their demand and the ones that can supply the goods that are in need. This system must decide the optimal way the stores in need can get the items they request in the network. The stores that are in need of goods send requests to the system, which in turn, taking into account the necessities and guaranteeing that there are no stock-outs for every store in the network, provides a solution. This solution provides for every store that requested the system, which other stores must be visited and in what order, as well as which type of products and in what quantity to collect from each. With this solution, the stores that requested goods will employ a vehicle to pick up the products in the designated places following the provided route and respecting the system's decisions.

The optimization problem at hand can be described as a combination of routing and inventory management decisions, problem which in the literature is presented as the inventory routing problem. The main goal of this optimization is to maximize the stores' on-shelf availability during the planning horizon, without causing stock-outs and taking into consideration the collection costs.

This variation of the inventory routing problem is dynamic, with routes planned daily based on end-of-day stock levels. The problem size varies with daily demand, as it affects end-of-day stock levels at each store. As mentioned before, stores, when in need of goods, send requests to the system, which in turn will calculate the best solution to answer these requests. Each day, multiple requests can be issued by the stores and multiple stores can answer to a store's request. This problem deals with multiple types of products and the maximum capacity of the vehicle employed by each store is not considered. Whenever a request can not be completely fulfilled, it will be partially fulfilled.

3.3 Formulation of the Problem

In this section, the mathematical formulation of the problem is presented, aiming to establish a representation of the underlying optimization framework. This formalization encompasses the essential notations, decision variables, parameters, objective function and constraints that define the problem.

3.3.1 Notation

R - the set of retailers $R = \{S_1, \ldots, S_{|R|}\}$.

K - the set of retailers with the lack of stock of at least one product, $K \subseteq R$, also known as the set of customers, $K = \{S_1, \ldots, S_{|K|}\}$.

J - the set of suppliers with at least one available product to supply to the customers, $J \subseteq R$, $J = \{S_1, \ldots, S_{|J|}\}$.

$|J|_k$ - the number of suppliers or visited stores for the requesting store k.

I - set that indicates the type of products in need to be collected, $i \in \{I_1, \ldots, I_{|I|}\}$.

$|I|_k$ - the number of different products needed by requesting store k.

3.3.2 Decision Variables

q_{ij}^k - order quantity of products i from requesting store k to supplier j, $k \neq j$.

y_{jl}^k - binary variable which takes 1 if supplier l is visited immediately after supplier j and 0 otherwise where $j \in R$ and $l \in R$ and $j \neq l$, $j \neq k$, $l \neq k$.

g_i^k - required quantity of product i from requesting store k.

3.3.3 Parameters

P_{ij}^k - available quantity of product i in supplier j in moment of retailer's k request.

d_{kj} - distance from the supplier k to the supplier j, $k \neq j$.

3.3.4 Assumptions

For the model development, the basic assumptions or requirements are made as follows:

1. A request is represented by the quantity needed by a retailer k of each product i.
2. The decisions are made by a regulatory centralized system.
3. The store type is homogeneous, being the available stock of each product the only discerning characteristic of the stores.
4. The value of collecting products from a store is estimated as the number of products available in the supplier per unit of the distance traveled to collect them.
5. The transportation is provided by the retailer k that requests the items.
6. Every visited store should be entered and left only once and by the same vehicle in a trip.
7. The collection of the available items is done in one trial.
8. Each requesting store will only have one request in a solution.

3.3.5 The Objective Function

$$F_k = \sum_{j=0}^{|J|_k} \sum_{l=1}^{|J|_k} \sum_{i=1}^{|I|_k} y_{jl}^k \frac{q_{il}^k}{d_{jl}}, \tag{3.1}$$

where $j = 0$ represents the requesting store k.

$$F = max \sum_{k=1}^{|K|} F_k \tag{3.2}$$

The objective function represents the maximization of the collection of products per travel distance, without causing stock-outs at any supplier, for the entire network.

Constraints

1. A vehicle leaves a store that it enters.

   Ensure that the number of times a vehicle enters a visited store is equal to the number of times it leaves that store:

   $\sum_{j=1}^{|J|_k} y_{jl}^k = \sum_{j=1}^{|J|_k} y_{lj}^k$

2. Ensure that every visited store is entered once for each request.

   $\sum_{j=1}^{|J|_k} y_{jl}^k = 1, \forall l \in \{2, ..., |J|_k\}, \forall k \in \{1, .., |K|\}$

   Together with the first constraint, it ensures that every visited store is entered only once.

3. A product request can be satisfied by multiple stores and can also be only partially satisfied.

   $g_i^k \geq \sum_{j=1}^{|J|_k} q_{ij}^k, j \neq k$

4. It is not possible to collect from a store more than it has in stock.

   $q_{ij}^k \leq P_{ij}^k$

5. A supplier's stock must be updated whenever products are picked up from that same store.

   $P_{ij}^k = P_{ij}^{k-1} - q_{ij}^{k-1}, k \in \{2, ..., |K|\}, \forall i \in \{1, ..., |I|_k\}, \forall j \in \{1, ..., |J|_k\},$

6. A requesting store can not collect products it has not requested.

   $q_{ij}^k \leq g_i^k$

7. It is not possible for a requesting store to collect products that the visited store has also requested.

   For a given product i let

   $$K_i = \{S_1^i, \ldots, S_{|K|}^i\}$$

   be the set of requering stores of product i and

   $$J_i = \{S_1^i, \ldots, S_{|J|}^i\}$$

   be the set of suppliers or visited stores, then

   $$K_i \cap J_i = \varnothing.$$

8. The vehicle issued by the requesting store will visit the requesting store last and will not take products from it.

   $$y_{jk}^k = 1, \qquad q_{ik}^k = 0 \qquad \text{when} \qquad j = |J^k|, \forall i.$$

3.4 Algorithms

In this section, the different optimization algorithms made to solve the problem are presented as well as the characteristics and thought process behind each one.

3.4.1 Baseline Algorithm (Dynamic2)

The baseline algorithm or Dynamic2, described in Algorithm 1, is a greedy algorithm made to generate feasible and promising initial solutions that can be used by the optimization algorithms as a good starting point. It receives as input the stores in the network, the requests, the distance between stores and a rank. This rank can go from one to the number of stores in the network. For every requesting store, the stores with products that the corresponding requesting store is in need of are selected as potential visited stores. Starting in the requesting store, the algorithm selects the next visited store based on its fitness value among the potential visited stores, considering the rank parameter. The rank determines the position of the store in the sorted list of potential visited stores, with higher fitness values corresponding to lower ranks. For instance, if the rank is set to 3, the algorithm chooses the next visited store with the third highest fitness from the potential visited stores. This fitness is evaluated according to the quantity of products needed by the requesting store present in a store and the distance from the current store to it. From this next visited store, the maximum amount of quantity is taken of the needed products without breaking constraints in order to avoid getting an unfeasible solution. It is important to mention that a product that a visited store is requesting is not taken from that store. The stocks, current and last visited store are updated. After this process is concluded for all the requesting stores, the solution is returned.

Since the number of Dynamic2 solutions for a given request list is limited, changing the order of the requesting stores in requests and rank will allow to get more Dynamic2 solutions. This proves useful and will be explored more in the next sections, since it can be used in conjunction with other algorithms in order to be able to explore the solution space more efficiently with Dynamic2 solutions. The Dynamic2 algorithm provides good initial solutions, even though greedy, since it takes from each visited store the maximum amount of product possible and the selection of the visited store process takes into account both the quantity available to collect and distance which are the most influential direct factors to the fitness function's values. Because of this it can provide solution with high quantities of product collected and routes with relatively low distances travelled, which is a good starting point for the optimization algorithms to improve on.

Algorithm 1 Dynamic2 Algorithm for the IRP

```
1: Input: total_stores, requests, distance_matrix, rank
2: // stores_with_prods tracks the stores with products that the requesting stores need, vs_stock keeps track of
   the stock of visited stores, total_qnt the quantity taken of each product by requesting stores
3: Initialize data structures: stores_with_prods, solution, vs_stock, total_qnt
4: for request in requests do
5:     Assign to stores_with_prods the stores that have the products it is requesting and the respective products
6:     last_store = requesting_store
7:     while stores_with_prods do
8:         // Fitness evaluated based on requested product quantities and distance from last_store
9:         Assign to chosen_store the store among stores_with_prods with the best fitness value
10:        for each product of the chosen_store do
11:            Assign to req_qnt the quantity requested of product by the chosen_store
12:            if total_qnt of requesting_store's product == req_qnt then
13:                continue
14:            end if
15:            if vs_stock of chosen_store's product > 0 then
16:                if chosen_store in requests' requesting stores then
17:                    if chosen_store is not requesting product then
18:                        Assign to qnt max amount possible of product without breaking constraints
19:                        Update total_qnt, vs_stock and update solution.
20:                    end if
21:                else
22:                    Assign to qnt max amount possible of product without breaking constraints
23:                    Update total_qnt, vs_stock and update solution.
24:                end if
25:            end if
26:        end for
27:        delete chosen_store entry from stores_with_prods
28:        last_store = chosen_store
29:    end while
30: end for
31: Output: solution
```

3.4.2 Simulated Annealing

Two renditions of the simulated annealing algorithm were implemented. In both implementations, like the classic algorithm, they receive as input an initial solution, an initial temperature, a cooling rate, temperature iterations and stopping temperature and return the best found solution.

In the first implementation 2, during each iteration of the algorithm, the temperature will be decreased by the cooling rate until it reaches a value equal or lower than the stopping temperature and consequently the algorithm satisfies its stopping condition. For each temperature reached during the course of the simulated annealing algorithm, and for the number of temperature iterations, a new solution, more specifically in this case, a neighbor solution, is generated by applying a perturbation or move operator to the current solution. The move operator in this case is shuffling the order of the visited stores for some of the requesting stores in the current solution, in hopes of finding better routes and consequently achieve less distance travelled for the requesting stores' employed vehicles. This move operator only generates feasible neighbors, solutions which do not break any of aforementioned constraints. After the neighbor solution is generated, the acceptance criterion, which is one of the core design issues of the simulated annealing algorithm, comes into play. If the newly generated solution has a higher objective function value than the current solution, the new solution will be accepted as the current solution. If not, it will be selected under a probability 2.4 that depends on the current temperature and the difference in the fitness value of the current and generated neighboring solution. As mentioned before, the algorithm initially has a higher probability of accepting worse solutions in order to escape local optima and as the temperature decreases it becomes more selective and favors solutions with a higher fitness value, ultimately striking a balance between intensification and diversification during the course of the simulated annealing algorithm.

Upon reaching the conclusion of the temperature iterations cycle, a jump in the solution search is performed in order to more effectively search the solution search space. This diversification move is done so that instead of just starting with an initial solution and applying small perturbations generating neighboring solutions in an area of the search space, even though there is the possibility of degradation of a solution and explore more than if there was not, we can do this to more areas of the solution search space that given the complexity of the problem at hand can be quite extensive. This jump in the solution search space is done by changing, through random mechanisms, the values of some of the requesting stores of the solution, which encompasses changing visited stores, products and quantities, without breaking the constraints and thus maintaining a feasible solution.

Algorithm 2 Simulated Annealing Algorithm for the IRP

Input: initial_solution, initial_temp, stopping_temp, cooling_rate, temp_iterations, total_stores, requests, distance_matrix
current_solution = initial_solution
best_solution = current_solution
temperature = initial_temp
while temperature $>$ stopping_temp **do**
    for each iteration in temp_iterations **do**
        Generate neighbor solution as new_solution
        Calculate fitness of new_solution as new_solution_fit
        Calculate fitness of current_solution as current_solution_fit
        if new_solution_fit > current_solution_fit **then**
            current_solution = new_solution
            if new_solution_fit $>$ fitness of the best_solution **then**
                best_solution = new_solution
            end if
        else
            Calculate $P(\Delta E, T)$ //probability of accepting a worse move
            if random(0, 1) $< P(\Delta E, T)$ **then**
                current_solution = new_solution
            end if
        end if
    end for
    Jump in the solution search space and assign solution to current_solution
    if fitness of current_solution $>$ best_solution **then**
        best_genome = current_solution
    end if
    temperature *= cooling_rate
end while

Output: best_solution

The second implementation 3 is identical to the first one 2, however the key difference between the two is in the approach to jump in the solution search space. In this implementation, the jump in the solution search space is accomplished by generating a non-duplicate Dynamic2 1 solution with a random but eligible rank. As mentioned before in subsection 3.4.1, the number of Dynamic2 solutions is limited and in order to generate more Dynamic2 solutions and explore different paths of the solution search space the rank mechanism was developed. With this approach, instead of doing a random jump in the search space, the jumps are more guided since it is jumping to generally more logical and better starting points in Dynamic2 solutions. From there, it improves these solutions by finding better routes and consequently lowering the distance travelled, where the distance travelled has a substantial impact in the fitness value of the solutions, which is evident by analyzing the objective function 3.2.

However, compared to the first implementation, the search space is not nearly as explored since the number of Dynamic2 solutions are limited, and that number depends on the number of stores and requests. So, this algorithm might be capable of converging at a faster rate and finding better solutions earlier because Dynamic2 solutions are, as explained before, a good starting point to improve on, but much more limited when searching the solution search space and consequently miss promising areas and solutions.

This implementation also introduces two additional stopping criteria, it stops when it has already jumped to every possible Dynamic2 solution possible for the specific instance and when it reaches a maximum number of iterations without finding a new Dynamic2 solution, since the jump is to non-duplicate Dynamic2 solutions.

Algorithm 3 Simulated Annealing Algorithm with jumps to Dynamic2 Solutions for the IRP

Input: initial_solution, initial_temp, stopping_temp, cooling_rate, temp_iterations, total_stores, requests, distance_matrix, max_iters_without_new_solution
iters_without_new_solution = 0
current_solution = initial_solution
best_solution = current_solution
temperature = initial_temp
// Maximum number of solutions generated by the dynamic2 algorithm for every rank
max_d_solutions = (number of requesting stores)! * number of stores

while temperature $>$ stopping_temp **and** max_d_solutions not reached **and** iters_without_new_solution < max_iters_without_new_solution **do**
    for each iteration in temp_iterations **do**
        Generate neighbor solution as new_solution
        Calculate fitness of new_solution as new_solution_fit
        Calculate fitness of current_solution as current_solution_fit
        if new_solution_fit > current_solution_fit **then**
            current_solution = new_solution
            if new_solution_fit $>$ fitness of the best_solution **then**
                best_solution = new_solution
            end if
        else
            Calculate $P(\Delta E, T)$ //probability of accepting a worse move
            if random(0, 1) $< P(\Delta E, T)$ **then**
                current_solution = new_solution
            end if
        end if
    end for
    Jump to a new non-duplicate Dynamic2 solution and assign it to current_solution
    Update iters_without_new_solution
    if fitness of current_solution $>$ best_solution **then**
        best_genome = current_solution
    end if
    temperature *= cooling_rate
end while

Output: best_solution

3.4.3 Tabu Search

Two distinct renditions of the Tabu Search algorithm were developed to tackle the Inventory Routing Problem. The first variation, Tabu Search with Diversification 4, makes use of short-term memory via tabu_list, which records recently explored solutions, preventing revisiting these solutions. Moreover, a diversification mechanism is also incorporated, allowing the exploration of diverse areas of the search space and aiding in the avoidance of local optima. In contrast, the second rendition of the tabu search algorithm, Tabu Search with Intensification and Diversification 5, not only includes short-term memory and diversification but also introduces medium-term memory through the elite_list. This elite_list acts as a mechanism for intensification by preserving promising solutions across generations, focusing the search on promising areas of the search space.

The first implementation 4, receives as input the initial solution, the tabu list size, the number of generations, the number of candidate solutions generated in each generation and the maximum number of generations without improvement in the same area of the solution search space. Firstly, the tabu_list, current_solution, best_solution and the number of iterations_without_improvement are initialized. At the beginning of each iteration of the algorithm, the diversification mechanism criteria is checked. If the algorithm has not found a solution that improves the best found solution for the previously mentioned maximum number of iterations, iterations_without_improvement is set to 0 and the diversification mechanism is applied. This diversification move, like in the Simulated Annealing algorithm 2, is a jump in the solution search space by means of changing, through random mechanism, the values of some of the requesting stores of the current solution. This move encompasses changing visited stores, products and quantities without breaking the constraints, like previously mentioned. It is done to explore more areas of the solution search space and escape local optima.

After the diversification criteria is checked, from the current solution the candidate or neighbor solutions are generated. The size of the neighborhood is defined by the subset_size variable. The move operator applied in order to generate the neighbor solutions is shuffling the order of the visited stores for some of the requesting stores and consequently changing the route for the vehicles employed by these requesting stores, in hopes of achieving less distance travelled. After generating the neighborhood, a local search is performed in order to find the best candidate solution to become the next current solution. In order for a solution in the neighborhood to be selected it must not be in tabu_list, which contains the recently selected solutions, or be in it but have a higher fitness value than the best found solution. When the

best candidate solution is found it becomes the current solution and the old current solution is appended to the tabu_list. If the tabu_list is full the oldest solution is removed. Lastly, the best_solution and iterations_without_improvement are updated. The algorithm will iterate until the number of generations is reached, which is the stopping criteria.

Algorithm 4 Tabu Search Algorithm with Diversification for the IRP

```
1:  Input: initial_solution, tabu_size, generations, subset_size, max_iter_without_imp, total_stores, requests,
    distance_matrix
2:  tabu_list = []
3:  current_solution = initial_solution
4:  best_solution = current_solution
5:  iterations_without_improvement = 0
6:  for i in range(generations) do
7:      if iterations_without_improvement == max_iter_without_imp then
8:          iterations_without_improvement = 0
9:          Jump in the solution search space and assign solution to current_solution
10:     end if
11:     Generate neighbor_solutions of current_solution and assign result to neighborhood
12:     Reset best_neighbor and best_neighbor_fit
13:     for neighbor in neighborhood do
14:         if neighbor_fit > best_neighbor_fit then
15:             if neighbor not in tabu_list or (neighbor in tabu_list and neighbor_fit > best_solution_fit) then
16:                 best_neighbor = neighbor
17:                 best_neighbor_fit = neighbor_fit
18:             end if
19:         end if
20:     end for
21:     if best_neighbor then
22:         current_solution = best_neighbor
23:         Remove oldest solution from tabu_list if full
24:         Append old current_solution to tabu_list
25:     end if
26:     Update best_solution and best_solution_fit if found
27:     Update iterations_without_improvement
28: end for
29:
30: Output: best_solution
```

In the second implementation 5, medium-term memory and consequently an intensification mechanism is implemented, which is what differentiates this implementation from the first 4. In this variation of the algorithm, there is a balance between diversification and intensification when exploring the solution search space.

The intensification mechanism is done through an elite_list, where the size of this list is dictated by the variable elite_size, received as input. The elite_list stores the latest elite_size best solutions found. When a maximum number of iterations without improvement of the best found solution at a certain point is reached, under a probability, either the diversification or intensification mechanism is applied. The intensification mechanism involves assigning one of the best previously found solution in the medium-term memory as the next current solution. This is done in order to intensify the search in promising areas of the solution search space. The probability of applying the diversification or intensification mechanism impacts the balance between these factors, it is important that the solution space is explored extensively specially because of its dimension, but it is also important to have the opportunity to guide the search in more promising regions.

The first implementation 4, shows a higher level of exploration by jumping more frequently in the solution search space. This level of exploration makes it cover a greater range of potential solutions. On the other hand, the second implementation 5, strikes a compromise between intensification and diversification. Even though it does not jump in the search space as extensively, it uses its medium-term memory, allowing it to focus more on promising regions of the search space, and potentially find better solutions.

Algorithm 5 Tabu Search Algorithm with Intensification and Diversification for the IRP

```
Input: initial_solution, tabu_size, generations, subset_size, elite_size, max_iter_without_imp, total_stores, requests, distance_matrix
Initialize tabu_list and elite_list
current_solution = initial_solution
best_solution = current_solution
iterations_without_improvement = 0
for i in range(generations) do
    if iterations_without_improvement == max_iter_without_imp then
        if random(0,1) < 0.7 or not elite_list then
            Jump in the solution search space and assign solution to current_solution
        else
            Assign to current_solution random element from elite_list
        end if
        iterations_without_improvement = 0
    end if
    Generate neighbor_solutions of current_solution and assign result to neighborhood
    Reset best_neighbor and best_neighbor_fit
    for neighbor in neighborhood do
        if neighbor_fit > best_neighbor_fit then
            if neighbor not in tabu_list or (neighbor in tabu_list and neighbor_fit > best_solution_fit) then
                best_neighbor = neighbor
                best_neighbor_fit = neighbor_fit
            end if
        end if
    end for
    if best_neighbor then
        current_solution = best_neighbor
        Remove oldest solution from tabu_list if full
        Append old current_solution to tabu_list
    end if
    Update best_solution and best_solution_fit if found
    Update iterations_without_improvement
end for

Output: best_solution
```

3.4.4 Genetic Algorithm

Two distinct renditions were also developed for the Genetic Algorithm. The key difference between these two variations of the genetic algorithm is that the second implementation, Genetic Algorithm with Elitism 7, uses a list of elite solutions to preserve the best-found solutions through generations as an intensification mechanism in the solution search space. Both implementations receive as input the number of generations of the algorithm, the initial population, the population size, which is the number of solutions to be selected in each generation of the algorithm, the crossover rate, which represents the probability of a crossover between two solutions happening, and the mutation rate, which represents the probability of mutating a solution and generating a new one.

The first implementation, Genetic algorithm 6, starts by initializing the hall_of_fame, which will contain the best solutions found in each generation. At the start of each generation of the algorithm, a selection process is performed. The selection process will pick the solutions to remain from the current population and generate a new population, named s_population. In this particular case, the selection method used is the tournament selection. This method consists in randomly selecting x solutions, where α represents the size of the tournament group. A tournament is then applied to the α members of the tournament group in order to select the one with the higher fitness value. This process is then repeated until the desired population size is obtained.

After the selection process is concluded, for half of the population size iterations, under a certain probability, the crossover rate, two random elements from the selected population are chosen as parents to perform a crossover operation. This crossover operation will result in the generation of two offspring that will be added to the selected population. The crossover operation in question can be described as randomly selecting requesting stores and switching the values of these requesting stores between the two mentioned parents. These values encompass the route of visited stores, as well as the products and quantity taken when visiting each of these stores. It is important to mention that the switching of requesting stores' values is between the same requesting stores of the two parents. This makes the probability of generating unfeasible solutions lower than switching between different requesting stores.

After the crossover process is completed, for population_size iterations, under the mutation rate, the mutation process starts. In this process, a random solution from the selected population is chosen for mutation. This solution is then mutated and a new solution is generated, which is also appended to the se-

lected population. The mutation operator consists in switching the order of the visited stores for a random number of requesting stores of a solution, in hopes of finding a better route and achieving less travelled distance. At the end of a generation, the best solution found in that iteration is added to hall_of_fame. The algorithm will iterate until the gen_limit is reached.

The second implementation, Genetic Algorithm with Elitism 7, differs from the first one in the sense that it implements an intensification mechanism by the way of an elite solution list. In each generation, the best solutions from the previous generation's selected population, which are stored in the list of elite solutions elites, are directly passed on to the next generation. Preserving the elite solutions makes the algorithm intensify the search over promising regions of the solution search space and encouraging the exploitation of the best solutions found.

Algorithm 6 Genetic algorithm for the IRP

```
1: Input: population, population_size, total_stores, requests, gen_limit, distance_matrix,
   crossover_rate, mutation_rate
2: Initialize hall_of_fame
3:
4: for i in range(gen_limit) do
5:     Perform selection of population and assign result to s_population
6:     Evaluate fitness of the s_population
7:     Append the solution with best fitness to the hall_of_fame
8:     for i in range(population_size//2) do
9:         if random(0,1) < crossover_rate then
10:            Assign two random elements from s_population as parents
11:            Perform crossover with the two parents resulting in the generation of two offspring
12:            Append the offspring to the s_population
13:        end if
14:    end for
15:    for i in range(population_size) do
16:        if random(0,1) < mutation_rate then
17:            Assign a random element from s_population for mutation
18:            Perform mutation resulting in the generation of an offspring
19:            Append the offspring to the s_population
20:        end if
21:    end for
22:    population = s_population
23: end for
24: Assign to best_solution the solution with best fitness in the hall_of_fame
25:
26: Output: best_solution
```

Algorithm 7 Genetic algorithm with Elitism for the IRP

```
1: Input: population, population_size, total_stores, requests, gen_limit, distance_matrix,
   crossover_rate, mutation_rate, elitism_size
2: Initialize hall_of_fame and elites
3:
4: for i in range(gen_limit) do
5:     Perform selection of population and assign result to s_population
6:     Evaluate fitness of the s_population
7:     Append the elitism_size solutions with the best fitness to elites
8:     Append the solution with best fitness to the hall_of_fame
9:     for i in range(population_size//2) do
10:        if random(0,1) < crossover_rate then
11:            Assign two random elements from s_population as parents
12:            Perform crossover with the two parents resulting in the generation of two offspring
13:            Append the offspring to elites
14:        end if
15:    end for
16:    for i in range(population_size) do
17:        if random(0,1) < mutation_rate then
18:            Assign a random element from s_population for mutation
19:            Perform mutation with the two parents resulting in the generation of an offspring
20:            Append the offspring to elites
21:        end if
22:    end for
23:    population = elites
24: end for
25: Assign to best_solution the solution with best fitness in the hall_of_fame
26:
27: Output: best_solution
```

Chapter 4

Results

In this chapter, the results of the developed algorithms are presented in three different scenarios, as shown in the table 2. Using the values of the columns in the mentioned table as a basis, the stores in the network, requests and distances between the stores are randomly generated. The Number of Stores represents the number of stores in the whole network, Max Number of Products represents the maximum number of different types of products a store can have, Max Quantity of Product represents the maximum quantity of a product a store can have in stock, Store Distance Range (km) represents the values that the distance between two stores in the network can have and the Number Of Requests represents the number of requesting stores in the network. Each of these scenarios has a different amount of data associated with it, whether it be in the number of stores, types of products or number of requests.

In each of these scenarios, the characteristics of the scenario, solutions, and results obtained from the developed algorithms are discussed, along with a comparison between them. The characteristics of the problem contain a description of the requests, the stock in the stores of the network, and the distance between them. The solutions and results contain, for each algorithm, a graphic representing the fitness of the solutions found throughout the iterations, statistics for the best solution found and for each of the store requests and a representation of this solution. The representation of the solution contains, for each requesting store, two figures that represent the route and the products and quantity taken at each visited store in order to answer the corresponding request. Lastly, in the subsection Comparison of Solutions, a table with the instances of the algorithms previously presented, as well as their fitness, number of solutions generated, parameters, iterations, and initial solution or population, is shown. From this table, a comparison of solutions is made. In order to compare the results of the different instances of the algorithms fairly, these were made to generate roughly the same number of solutions.

The tests were performed in a Lenovo L480 with an Intel(R) Core(TM) i5-8250U processor, featuring a

base clock speed of 1.60GHz and maximum frequency of 1.80 GHz. The machine is equipped with 16GB of RAM and a solid-state drive (SSD).

Table 2: Scenario Table

Scenario	Number Of Stores	Max Number Of Products	Max Quantity Of Product	Store Distance Range (km)	Number Of Requests
1	10	50	1-1000	10-100	7
2	250	50	1-500	250-500	113
3	10	300	500-1500	10-100	4

4.1 Scenario 1

4.1.1 Characteristics of the Problem

Products and Quantity Needed for each Requesting Store

In the network of stores, seven stores are in need of at least one product. There is a necessity for 47 types of products, most of them requested by more than one of the seven requesting stores. Only products 5, 14, 24, and 37 are requested by just one store. The accumulated requested quantities range from approximately 200 to more than 2500 units of a product (figure 11).

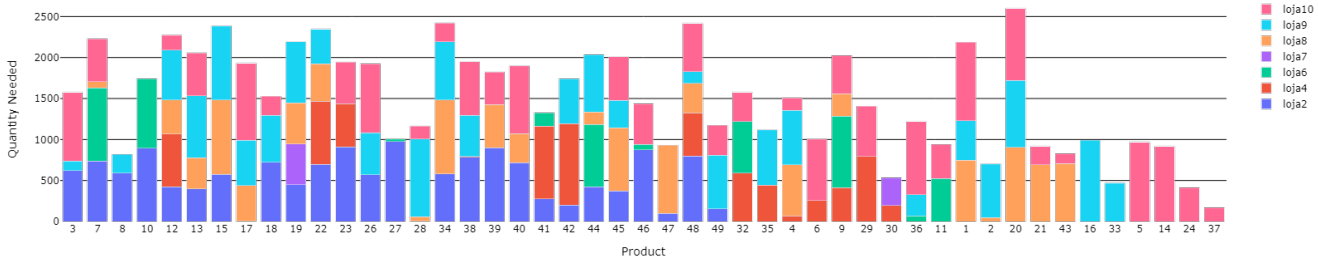


Figure 11: Bar chart representing the needs of the requesting stores.

Stock of the Requested Products

In the network of stores, there are 47 types of products in stock, and the accumulated quantities of a product in at least one store vary approximately from 300 to 4000 units. It is important to mention that the distribution of products and their quantities is not uniform across all stores in the network (figure 12).

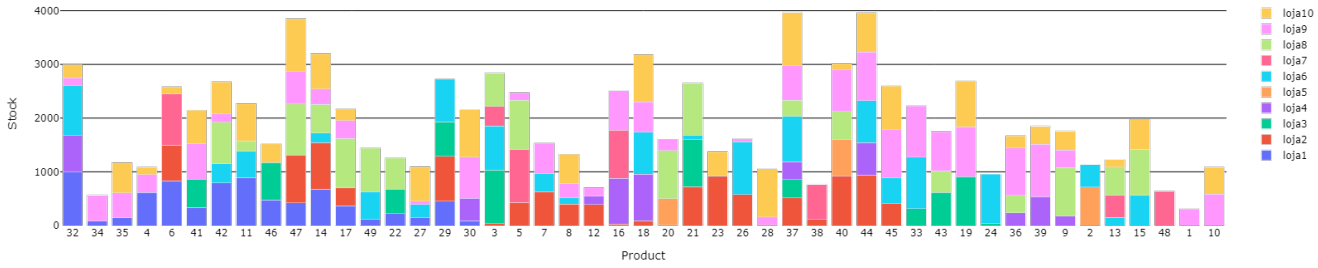


Figure 12: Bar chart representing the stock of the requested products.

Distances between the Stores

The distance between two stores is distributed randomly from a minimum of 10 kilometers to a maximum of 100 kilometers (figure 13).

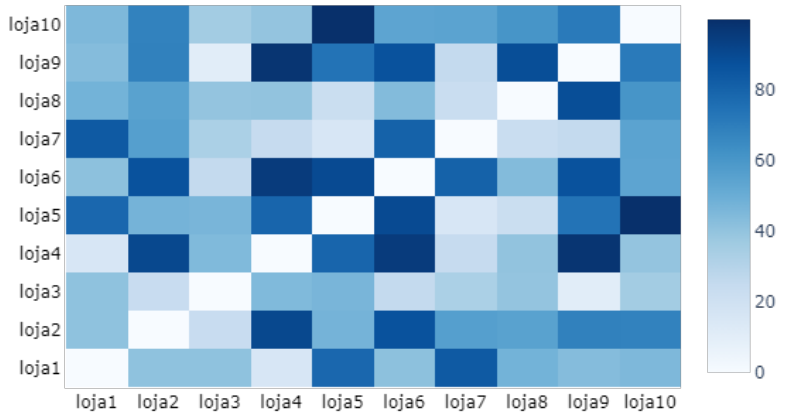


Figure 13: Heatmap representing the distances between the stores in the network.

4.1.2 Solutions and Results:

Dynamic2

Statistics of the Solution

The Dynamic2 algorithm's solution has a fitness of 121.77. It was requested for 133 products across all requesting stores, of which 60 were completely fulfilled. In total, 38339 units were collected, accounting to an average fulfillment of 56%. There were 40 store visits, accumulating a travel distance of 2106.37 kilometers (table 3).

Table 3: Statistics for each request with the Dynamic2 solution.

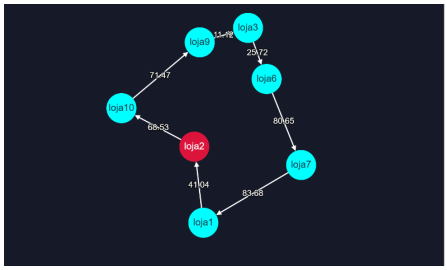
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	60	40	2106.37	38339	0.56	121.77	70739

The requesting store that accounted for the highest value of the fitness function was the store loja2, which requested 27 products, amounting to 14742 units. The vehicle employed by loja2, visited 6 distinct stores and collected 12309 units, traveling 382.21 kilometers. The second store with the highest fitness value is loja4, which requested 14 types of products, amounting to 7119 units. The vehicle employed by this store, visited 4 distinct stores and collected 4554 units, traveling 205.44 kilometers. The store with the lowest fitness value is the store loja6, which requested 10 types of products, amounting to 4854 units. The vehicle employed by this store, visited 4 distinct stores and collected 2320 units, traveling 300.58 kilometers (table 4).

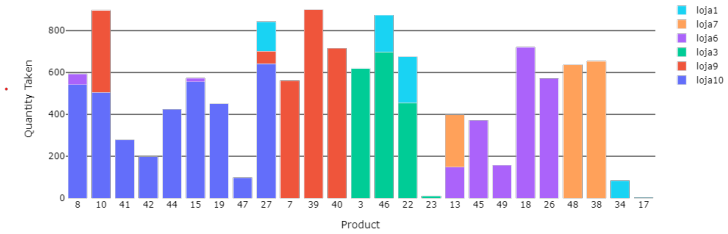
Table 4: Statistics for each request with the Dynamic2 solution.

Requesting Store	Number of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness
loja2	27	loja10, loja9, loja3, loja6, loja7, loja1	6	382.21	14742	12309	32.20
loja4	14	loja1, loja10, loja3, loja9	4	206.44	7119	4554	22.06
loja6	10	loja10, loja1, loja4, loja9	4	300.58	4854	2320	7.72
loja7	2	loja9, loja3	2	69.90	836	836	11.96
loja8	22	loja10, loja9, loja3, loja2, loja1, loja4, loja7, loja5, loja6	9	399.99	11125	6038	15.10
loja9	26	loja3, loja6, loja8, loja7, loja10, loja4, loja5	7	352.53	14952	5323	15.10
loja10	32	loja6, loja8, loja7, loja9, loja3, loja1, loja4, loja5	8	394.72	17111	6959	17.63

Representation of the Solution

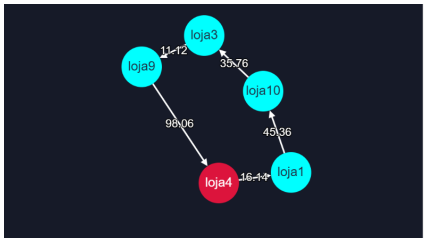


(a) Route for the store loja2's request.

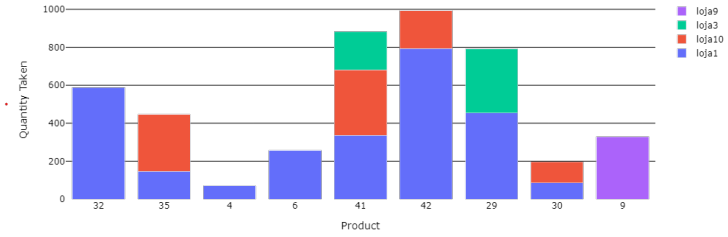


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 14: Requesting store's loja2 part of the solution.

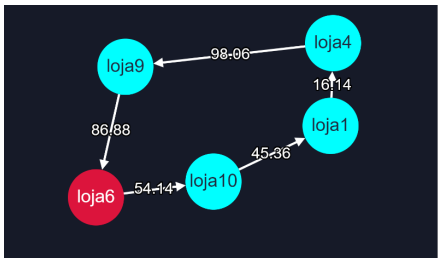


(a) Route for the store loja4's request.

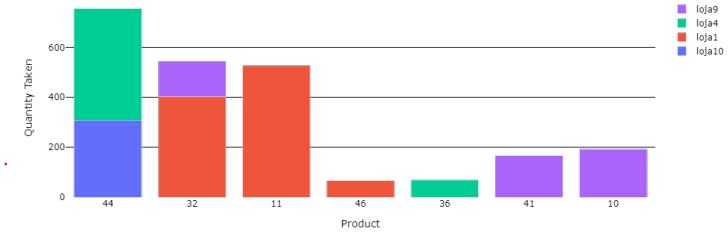


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 15: Requesting store's loja4 part of the solution.

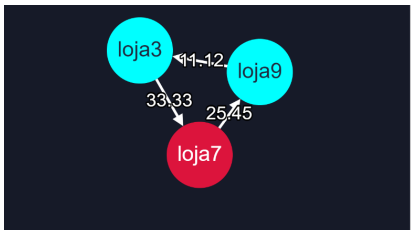


(a) Route for the store loja6's request.

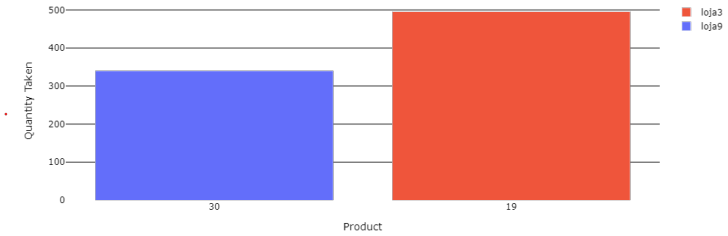


(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 16: Requesting store's loja6 part of the solution.

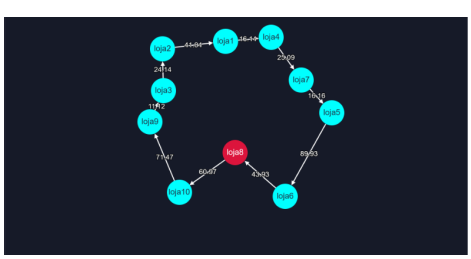


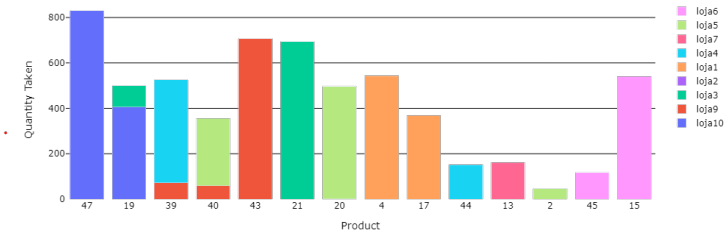
(a) Route for the store loja7's request.



(b) Products and respective quantities taken at each visited store for store loja7's request.

Figure 17: Requesting store's loja7 part of the solution.

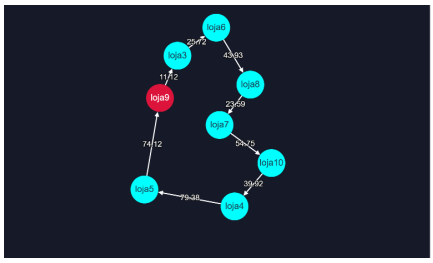


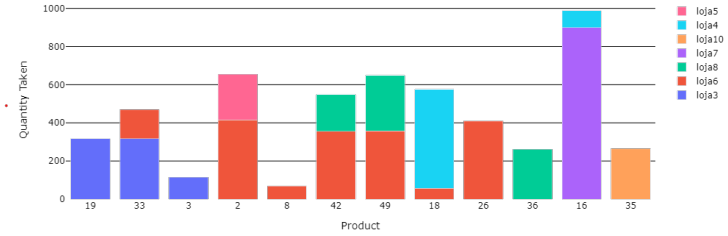


(a) Route for the store loja8's request.

(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 18: Requesting store's loja8 part of the solution.

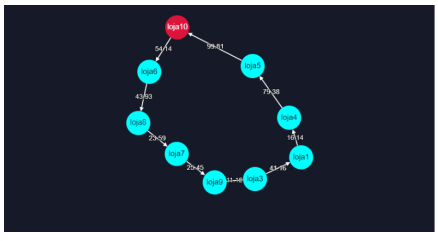


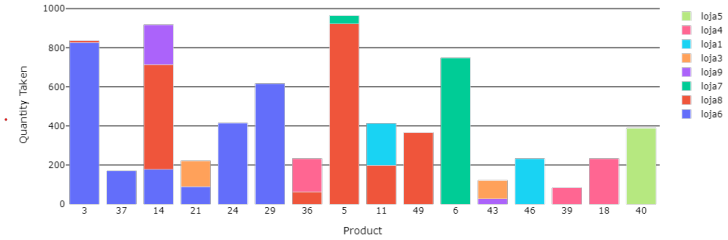


(a) Route for the store loja9's request.

(b) Products and respective quantities taken at each visited store for store loja9's request.

Figure 19: Requesting store's loja9 part of the solution.





(a) Route for the store loja10's request.

(b) Products and respective quantities taken at each visited store for store loja10's request.

Figure 20: Requesting store's loja10 part of the solution.

Simulated Annealing:

In this instance, the Simulated Annealing algorithm used the Dynamic2 solution mentioned in subsection 4.1.2 as initial solution, with a fitness of 121.77.

Statistics of the solution

In this fitness graph 21, it is represented the fitness through the iterations of the Simulated Annealing algorithm. The fitness fluctuates approximately between 20 to 142, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 40 and 60.

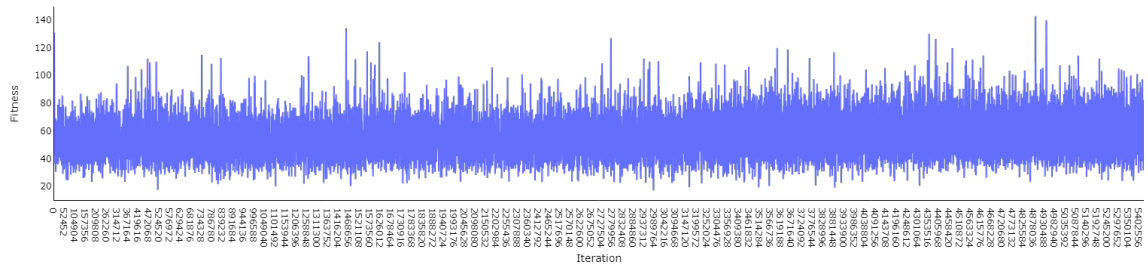


Figure 21: Fitness of the solutions found by the Simulated Annealing algorithm during its iterations.

The Simulated Annealing algorithm's solution has a fitness of 142.61. It was requested for 133 products across all requesting stores, of which 14 were completely fulfilled. In total, 12696 units were collected, accounting to an average fulfillment of 21%. There were 17 stores visited, accumulating a travel distance of 902.99 kilometers (table 5).

Table 5: Statistics of the Simulated Annealing solution.

Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	14	17	902.99	12696	0.21	142.61	70739

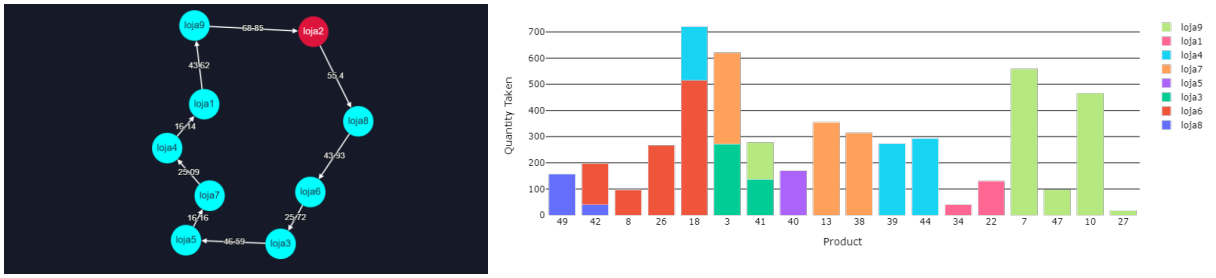
The requesting store that accounted for the highest value of the fitness function was the store loja4, which requested 14 products, amounting to 7119 units. The vehicle employed by loja4, visited 1 store and collected 1623 units, traveling 32.28 kilometers. The second store with the highest fitness value is the store loja9, which requested 26 types of products, amounting to 14952 units. The vehicle employed by this store, visited 1 store and collected 669 units, traveling 22.24 kilometers. The store with the lowest

fitness value is loja7, which requested 2 types of products, amounting to 836 units. The vehicle employed by this store, visited 1 store and collected 651 units, traveling 109.50 kilometers (table 6).

Table 6: Statistics for each request with the Simulated Annealing solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (km)	Quantity Requested	Quantity Taken	Fitness
loja2	27	loja8, loja6, loja3, loja5, loja7, loja4, loja1, loja9	8	341.50	14742	5052	14.79
loja4	14	loja1	1	32.28	7119	1623	50.28
loja6	10	loja1, loja4, loja3	3	128.09	4854	1190	9.29
loja7	2	loja10	1	109.50	836	651	5.95
loja8	22	loja3	1	79.84	11125	1889	23.66
loja9	26	loja3	1	22.24	14952	669	30.08
loja10	32	loja6, loja4	2	189.54	17111	1622	8.56

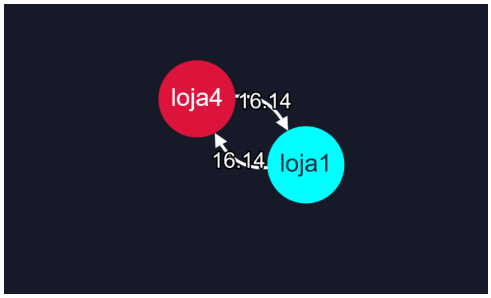
Representation of the solution

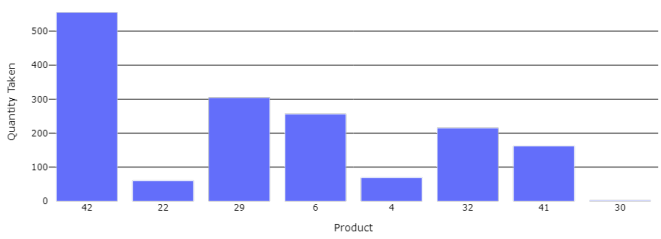


(a) Route for the store loja2's request.

(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 22: Requesting store's loja2 part of the solution.

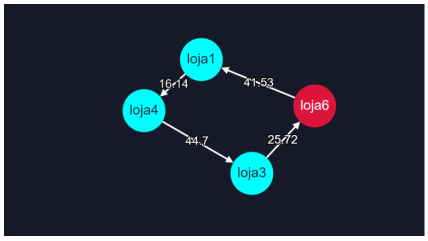


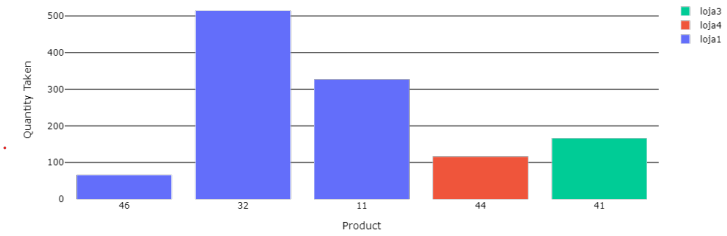


(a) Route for the store loja4's request.

(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 23: Requesting store's loja4 part of the solution.

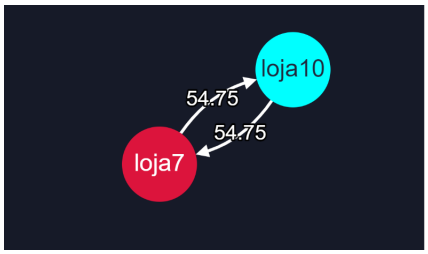


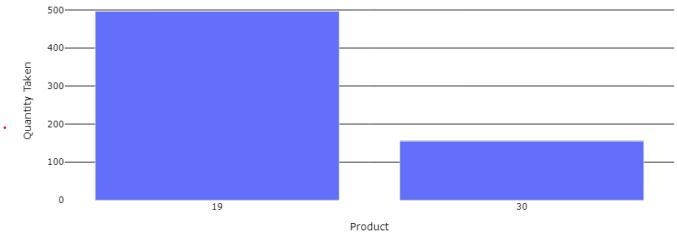


(a) Route for the store loja6's request.

(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 24: Requesting store's loja6 part of the solution.

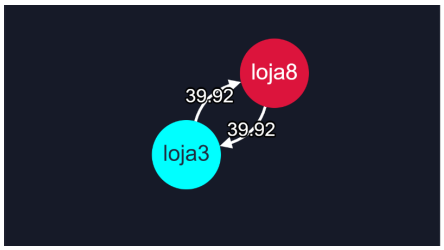


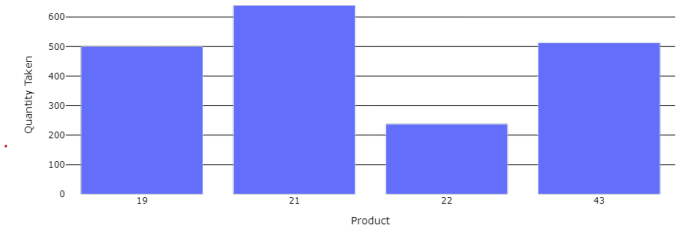


(a) Route for the store loja7's request.

(b) Products and respective quantities taken at each visited store for store loja7's request.

Figure 25: Requesting store's loja7 part of the solution.

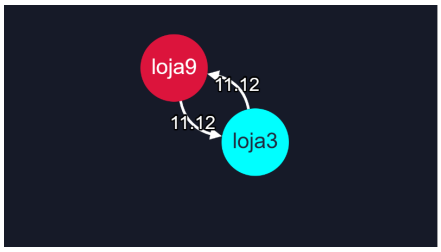


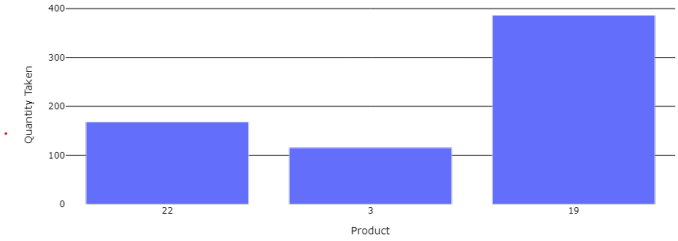


(a) Route for the store loja8's request.

(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 26: Requesting store's loja8 part of the solution.

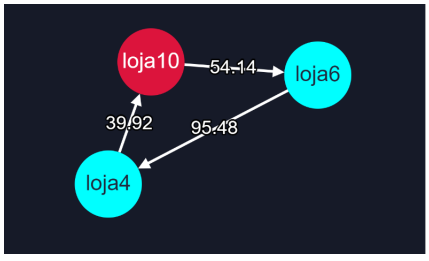


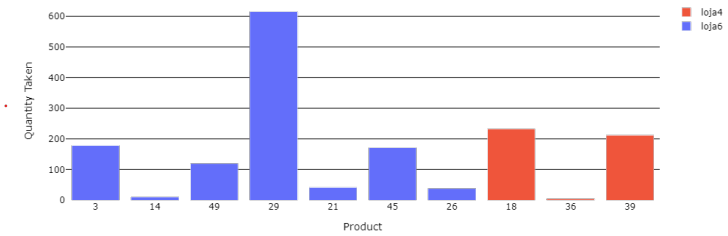


(a) Route for the store loja9's request.

(b) Products and respective quantities taken at each visited store for store loja9's request.

Figure 27: Requesting store's loja9 part of the solution.





(a) Route for the store loja10's request.

(b) Products and respective quantities taken at each visited store for store loja10's request.

Figure 28: Requesting store's loja10 part of the solution.

Simulated Annealing using Dynamic2 solutions

In this instance, the Simulated Annealing using Dynamic2 solutions algorithm used the Dynamic2 solution mentioned in subsection 3.4.1 as initial solution.

Statistics of the Solution

In this fitness graph 29, it is evident that the number of peaks is higher compared to the Simulated Annealing instance in subsection 4.1.2. This can be explained by the fact that the algorithm in question jumps in the solution search space to solutions generated by the Dynamic2 algorithm instead of jumping to random points in the search space. As mentioned before, these solutions generated by the Dynamic2 algorithms are generally better than solutions generated through random mechanisms. When these jumps happen, they can be further improved.

The fitness fluctuates approximately between 60 to 160, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 60 and 100.

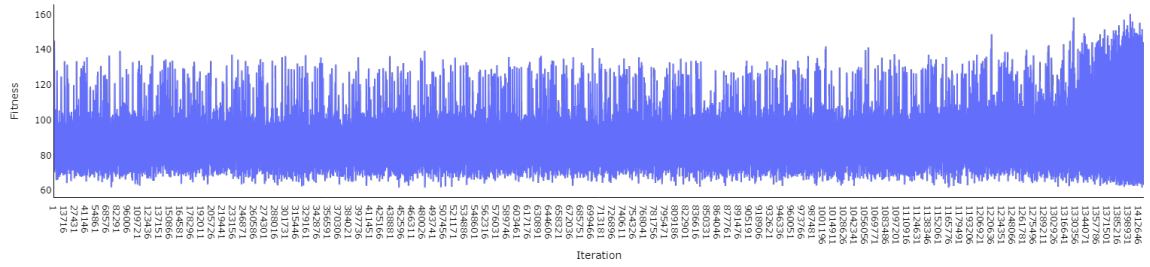


Figure 29: Fitness of the solutions found by the Simulated Annealing algorithm using Dynamic2 solutions during its iterations.

The Simulated Annealing using Dynamic2 solutions algorithm's solution has a fitness of 160.30. It was requested for 133 products across all requesting stores, of which 58 were completely fulfilled. In total, 38339 units were collected, accounting to an average fulfillment of 53%. There were 37 store visits, accumulating a travel distance of 1605.10 kilometers (table 7).

Table 7: Statistics of the Simulated Annealing Dynamic2 algorithm's solution.

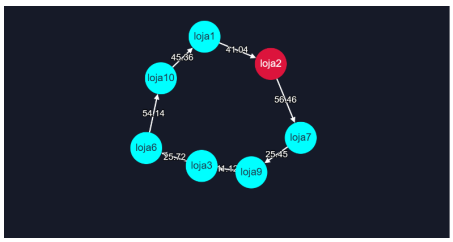
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	58	37	1605.10	38339	0.53	160.30	70739

The requesting store that accounted for the highest value of the fitness function was the store loja2, which requested 27 products, amounting to 14742 units. The vehicle employed by loja2, visited 6 distinct stores and collected 12309 units, traveling 259.29 kilometers. The second store with the highest fitness value is the store loja4, which requested 14 types of products, amounting to 7119 units. The vehicle employed by this store, visited 3 distinct stores and collected 4225 units, traveling 69.90 kilometers. The store with the lowest fitness value is the store loja6, which requested 10 types of products, amounting to 4854 units. The vehicle employed by this store, visited 4 distinct stores and collected 1832 units, traveling 281.34 kilometers (table 10).

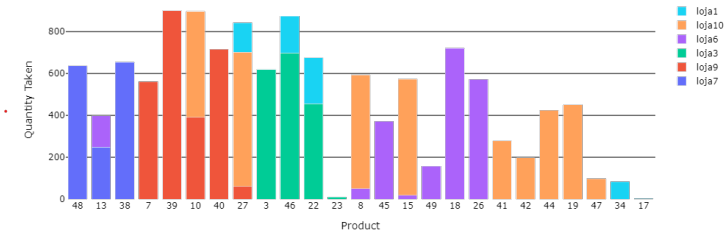
Table 8: Statistics for each request with the Simulated Annealing Dynamic2 algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness
loja2	27	loja7, loja9, loja3, loja6, loja10, loja1	6	259.29	14742	12309	47.47
loja10	32	loja2, loja3, loja9, loja7, loja8, loja5, loja6, loja1, loja4	9	363.19	17111	10409	28.66
loja4	14	loja1, loja3, loja10	3	132.98	7119	4225	31.77
loja7	2	loja9, loja3	2	69.90	836	836	11.96
loja6	10	loja1, loja4, loja9, loja10	4	281.34	4854	1832	6.51
loja9	26	loja3, loja6, loja8, loja5, loja7, loja10, loja4, loja1	7	274.20	14952	6159	22.46
loja8	22	loja4, loja10, loja3, loja9, loja5	5	224.20	11125	2569	11.46

Representation of the solution

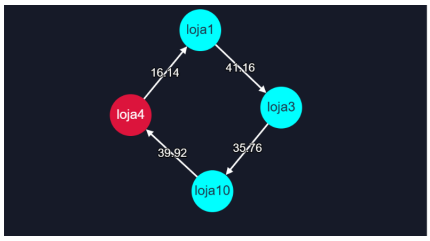


(a) Route for the store loja2's request.

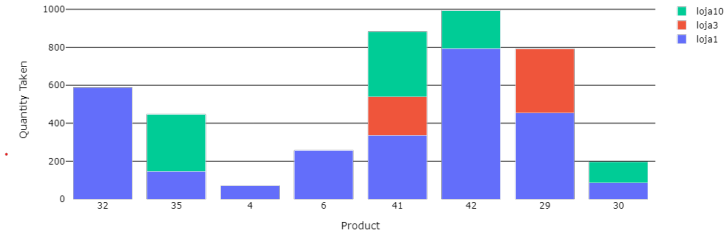


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 30: Requesting store's loja2 part of the solution.

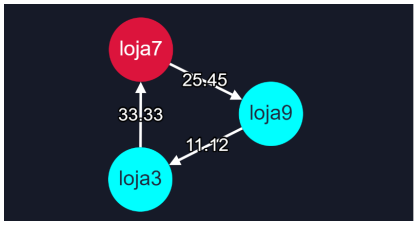


(a) Route for the store loja4's request.

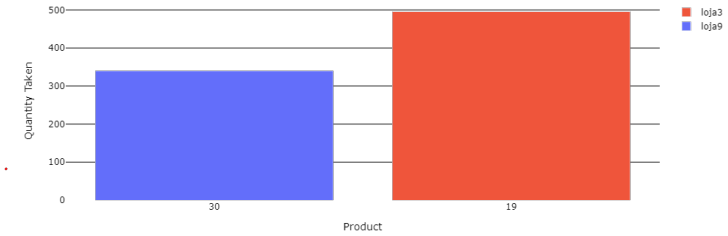


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 31: Requesting store's loja4 part of the solution.

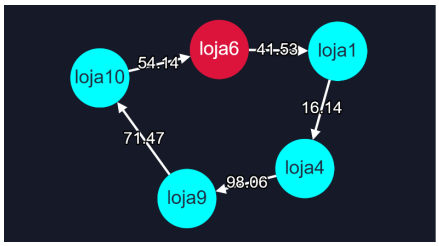


(a) Route for the store loja7's request.

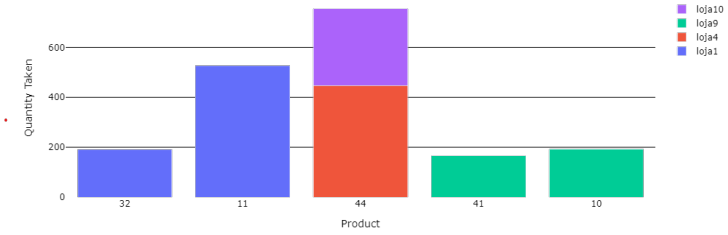


(b) Products and respective quantities taken at each visited store for store loja7's request.

Figure 32: Requesting store's loja7 part of the solution.

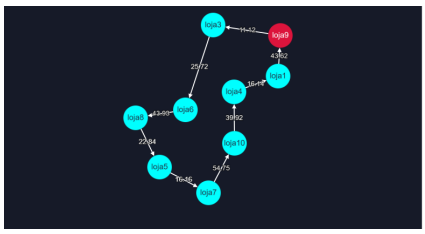


(a) Route for the store loja6's request.

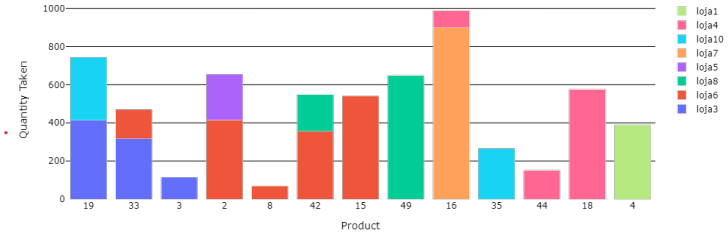


(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 33: Requesting store's loja6 part of the solution.

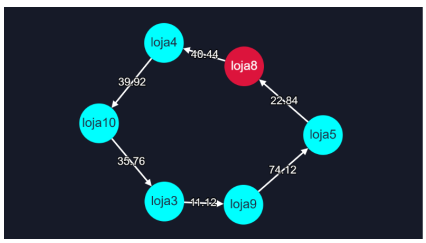


(a) Route for the store loja9's request.

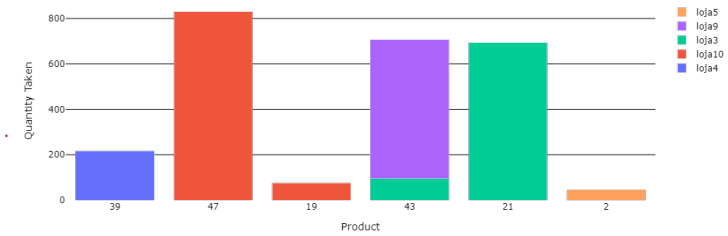


(b) Products and respective quantities taken at each visited store for store loja9's request.

Figure 34: Requesting store's loja9 part of the solution.



(a) Route for the store loja8's request.



(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 35: Requesting store's loja8 part of the solution.

Tabu Search

In this instance, the Tabu Search algorithm used the Dynamic2 solution mentioned in subsection 3.4.1 as initial solution.

Statistics of the solution

In this fitness graph 36, it is represented the fitness through the iterations of the Tabu Search algorithm. The fitness fluctuates approximately between 30 to 138, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 50 and 70.

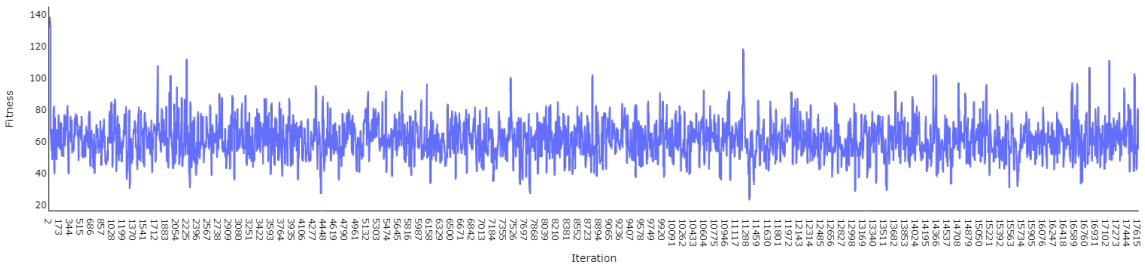


Figure 36: Fitness of the solutions found by the Tabu Search algorithm during its iterations.

The Tabu Search algorithm's solution has a fitness of 138.62. It was requested for 133 products across all requesting stores, of which 60 were completely fulfilled. In total, 38339 units were collected, accounting to an average fulfillment of 53%. There were 40 store visits, accumulating a travel distance of 2084.15 kilometers (table 9).

Table 9: Statistics of the Tabu Search algorithm's solution.

Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	60	40	2084.15	38339	0.56	138.62	70739

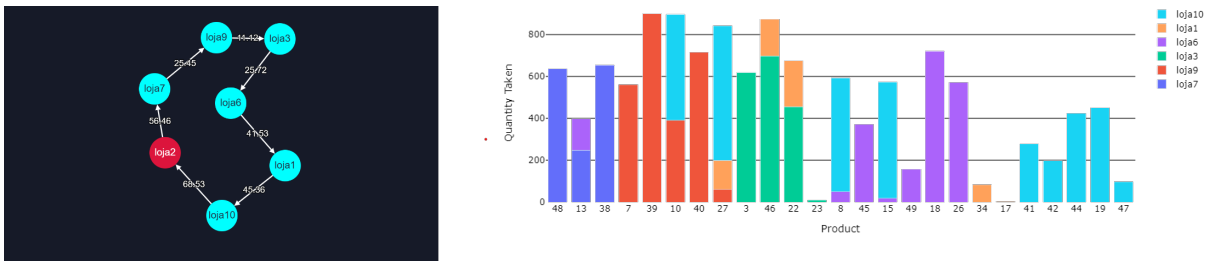
The requesting store that accounted for the highest value of the fitness function was the store loja2, which requested 27 products, amounting to 14742 units. The vehicle employed by loja2, visited 6 distinct stores and collected 12309 units, traveling 274.17 kilometers. The second store with the highest fitness value is the store loja4, which requested 14 types of products, amounting to 7119 units. The vehicle employed by this store, visited 4 distinct stores and collected 4554 units, traveling 146.56 kilometers. The store with

the lowest fitness value is the store loja6, which requested 10 types of products, amounting to 4854 units. The vehicle employed by this store, visited 4 distinct stores and collected 2320 units, traveling 280.85 kilometers (table 10).

Table 10: Statistics for each request for the Tabu Search's best found solution.

Requesting Store	Number of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness
loja2	27	loja7, loja9, loja3, loja6, loja1, loja10	6	274.17	14742	12309	44.90
loja4	14	loja10, loja3, loja9, loja1	4	146.56	7119	4554	31.07
loja6	10	loja10, loja9, loja1, loja4	4	280.85	4854	2320	8.26
loja7	2	loja9, loja3	2	69.9	836	836	11.96
loja8	22	loja6, loja3, loja1, loja10, loja7, loja4, loja5, loja9, loja2	9	513.76	11125	6038	11.75
loja9	26	loja3, loja8, loja5, loja4, loja10, loja6, loja7	7	353.42	14952	5323	15.06
loja10	32	loja1, loja4, loja5, loja9, loja6, loja3, loja7, loja8	8	445.49	17111	6959	15.62

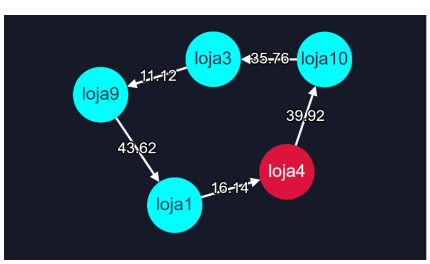
Representation of the solution



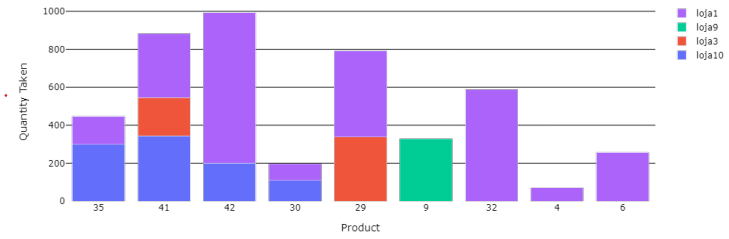
(a) Route for the store loja2's request.

(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 37: Requesting store's loja2 part of the solution.

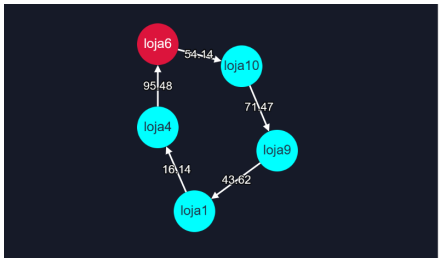


(a) Route for the store loja4's request.

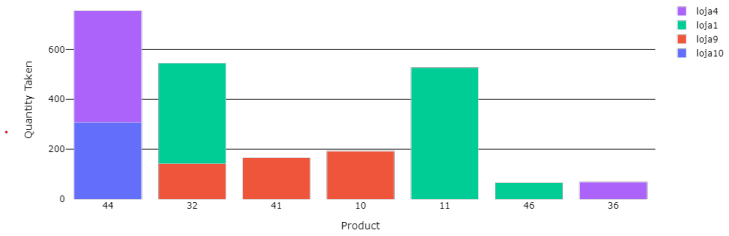


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 38: Requesting store's loja4 part of the solution.

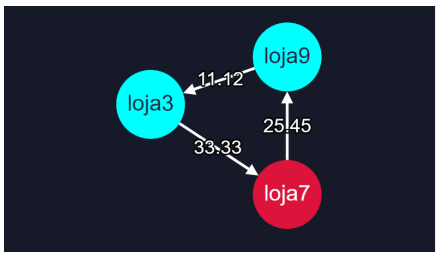


(a) Route for the store loja6's request.

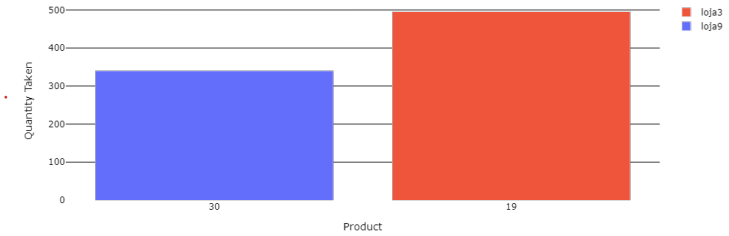


(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 39: Requesting store's loja6 part of the solution.

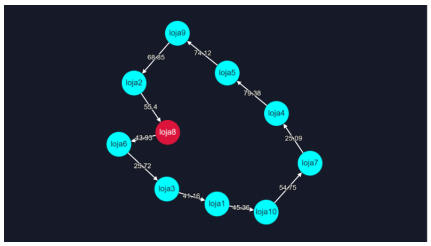


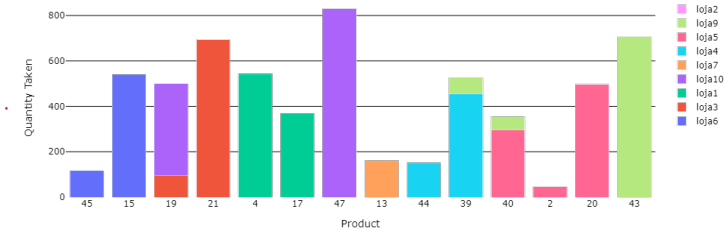
(a) Route for the store loja7's request.



(b) Products and respective quantities taken at each visited store for store loja7's request.

Figure 40: Requesting store's loja7 part of the solution.

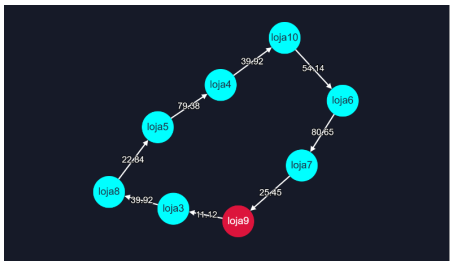


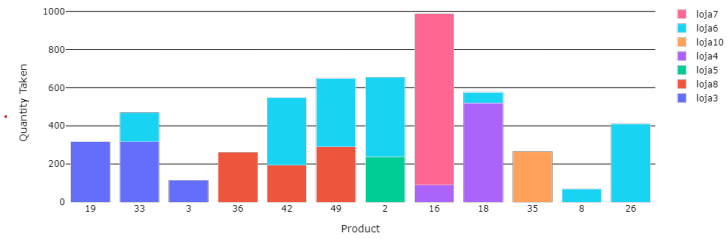


(a) Route for the store loja8's request.

(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 41: Requesting store's loja8 part of the solution.

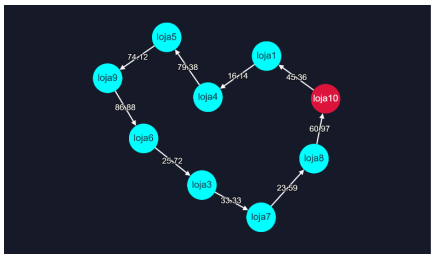


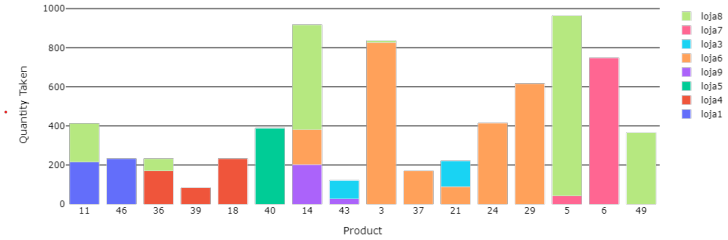


(a) Route for the store loja9's request.

(b) Products and respective quantities taken at each visited store for store loja9's request.

Figure 42: Requesting store's loja9 part of the solution.





(a) Route for the store loja10's request.

(b) Products and respective quantities taken at each visited store for store loja10's request.

Figure 43: Requesting store's loja10 part of the solution.

Tabu Search with Intensification and Diversification

In this instance, the Tabu Search with Intensification and Diversification algorithm used the Dynamic2 solution mentioned in subsection 3.4.1 as initial solution.

Statistics of the solution

In this fitness graph 44, it is evident that the number of peaks is higher compared to the Tabu Search instance in subsection 4.1.2. This might be due to the intensification mechanism by the way of elitism mentioned before. This consists in selecting one of the best found solutions in the medium-term memory as the current solution in order to intensify the search in promising areas of the solution search space.

The fitness fluctuates approximately between 30 to 153, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 60 and 140.

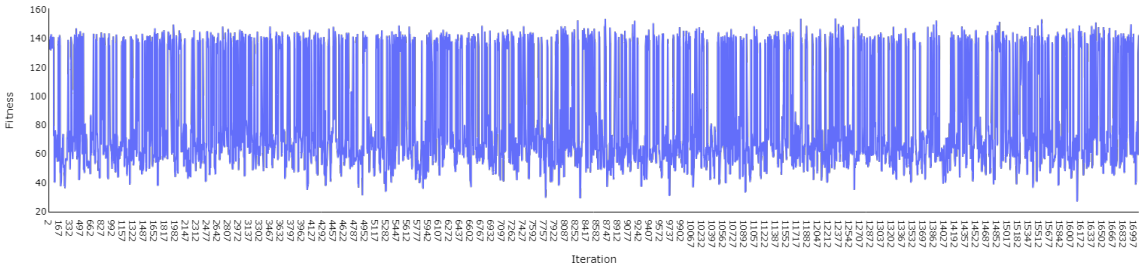


Figure 44: Fitness of the solutions found by the Tabu Search with Intensification and Diversification algorithm during its iterations.

The Tabu Search with Intensification and Diversification algorithm's solution has a fitness of 153.87. It was requested for 133 products across all requesting stores, of which 60 were completely fulfilled. In total, 38339 units were collected, accounting to an average fulfillment of 56%. There were 40 store visits, accumulating a travel distance of 1879.55 kilometers (table 11).

Table 11: Statistics of the Tabu Search with Intensification and Diversification algorithm's solution.

Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	60	40	1879.55	38339	0.56	153.87	70739

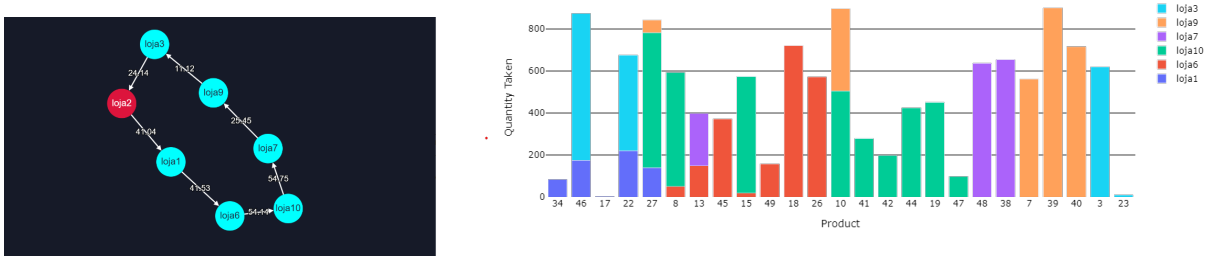
The requesting store that accounted for the highest value of the fitness function was the store loja2, which

requested 27 products, amounting to 14742 units. The vehicle employed by loja2, visited 6 distinct stores and collected 12309 units, traveling 252.17 kilometers. The second store with the highest fitness value is the store loja4, which requested 14 types of products, amounting to 7119 units. The vehicle employed by this store, visited 4 distinct stores and collected 4554 units, traveling 146.56 kilometers. The store with the lowest fitness value is the store loja6, which requested 10 types of products, amounting to 4854 units. The vehicle employed by this store, visited 4 distinct stores and collected 2320 units, traveling 240.70 kilometers (table 12).

Table 12: Statistics for each request of the Tabu Search with Intensification and Diversification algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness of the Requesting Store
loja2	27	loja1, loja6, loja10, loja7, loja9, loja3	6	252.17	14742	12309	48.81
loja4	14	loja1, loja9, loja3, loja10	4	146.56	7119	4554	31.07
loja6	10	loja10, loja4, loja1, loja9	4	240.70	4854	2320	9.64
loja7	2	loja3, loja9	2	69.90	836	836	11.96
loja8	22	loja5, loja10, loja6, loja2, loja7, loja3, loja4, loja9, loja1	9	587.84	11125	6038	10.27
loja9	26	loja3, loja6, loja8, loja5, loja7, loja4, loja10	7	256.25	14952	5323	20.77
loja10	32	loja5, loja7, loja9, loja3, loja6, loja8, loja1, loja4	8	326.13	17111	6959	21.34

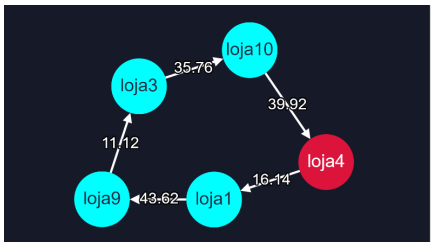
Representation of the solution



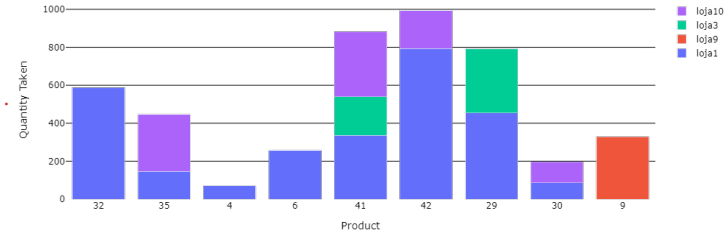
(a) Route for the store loja2's request.

(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 45: Requesting store's loja2 part of the solution.

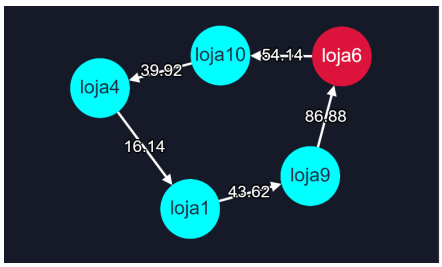


(a) Route for the store loja4's request.

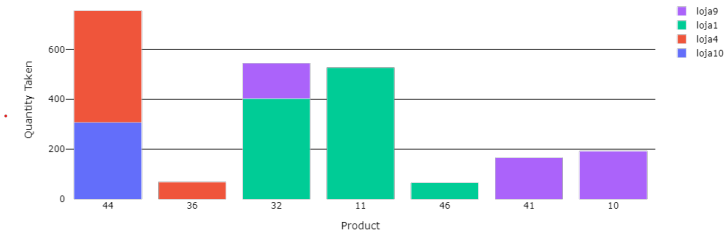


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 46: Requesting store's loja4 part of the solution.

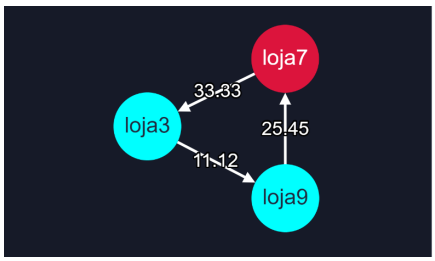


(a) Route for the store loja6's request.

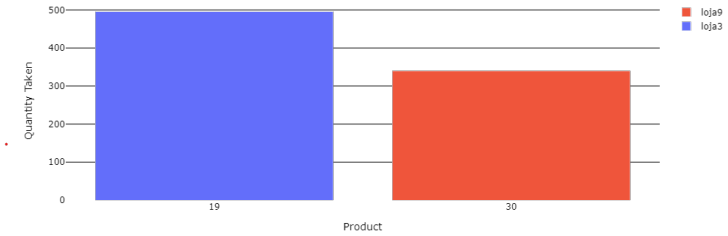


(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 47: Requesting store's loja6 part of the solution.

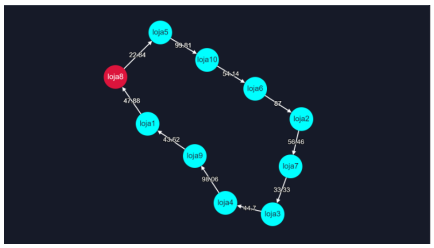


(a) Route for the store loja7's request.

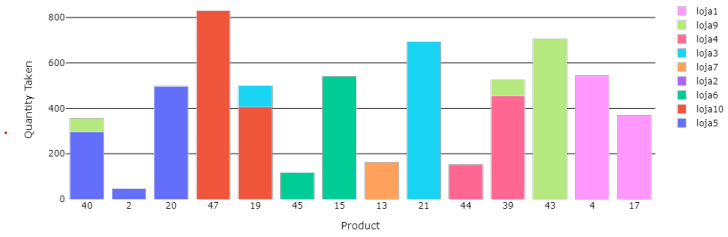


(b) Products and respective quantities taken at each visited store for store loja7's request.

Figure 48: Requesting store's loja7 part of the solution.

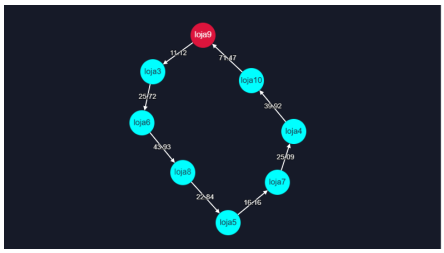


(a) Route for the store loja8's request.

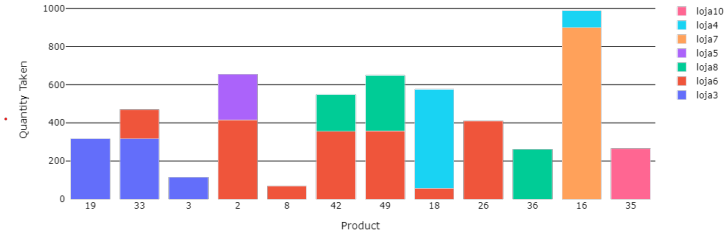


(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 49: Requesting store's loja8 part of the solution.

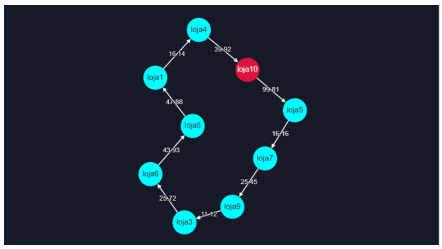


(a) Route for the store loja9's request.

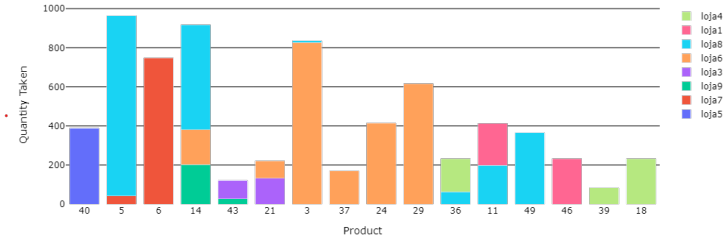


(b) Products and respective quantities taken at each visited store for store loja9's request.

Figure 50: Requesting store's loja9 part of the solution.



(a) Route for the store loja10's request.



(b) Products and respective quantities taken at each visited store for store loja10's request.

Figure 51: Requesting store's loja10 part of the solution.

Genetic Algorithm

In this instance, an initial population of 1000 solutions obtained through random procedures is used. Every solution in this initial population is feasible. The average fitness of the mentioned population is 51.98.

Statistics of the solution

In this fitness graph 52, it is represented the average fitness of the population through the generations of the Genetic algorithm. The fitness fluctuates approximately between 52 to 190, having values with an improved fitness over the initial population. In the first 30 generations, the average of the fitness values of the population increased consistently from around 52 to 188, slightly fluctuating around the latter until the end, reaching a maximum value of 190.15.

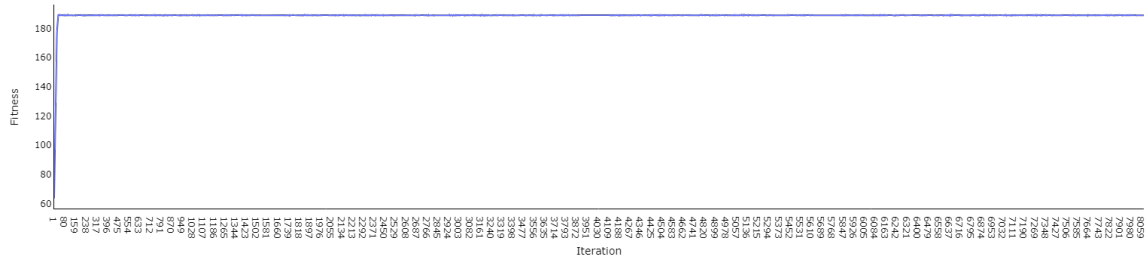


Figure 52: Fitness of the solutions found by the Genetic algorithm during its iterations.

The Genetic algorithm's solution has a fitness of 190.15. It was requested for 133 products across all requesting stores, of which 15 were completely fulfilled. In total, 11804 units were collected, accounting to an average fulfillment of 19%. There were 11 store visits, accumulating a travel distance of 604.60 kilometers (table 13).

Table 13: Statistics of the Solution.

Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	15	11	604.60	11804	0.19	190.15	70739

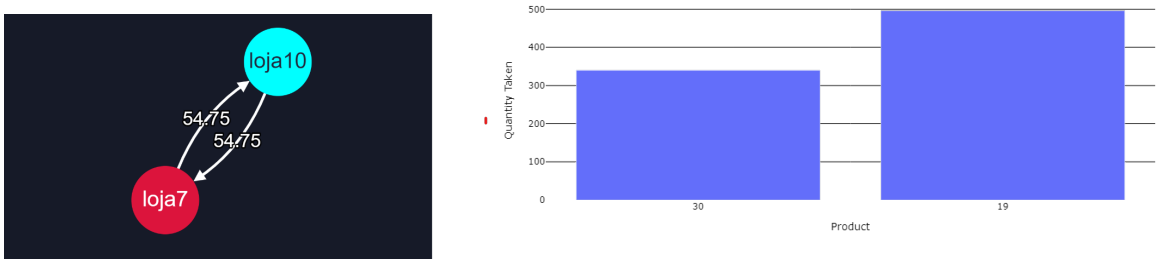
The requesting store that accounted for the highest value of the fitness function was the store loja4, which requested 14 products, amounting to 7119 units. The vehicle employed by loja4, visited 1 store and collected 1984 units, traveling 32.28 kilometers. The second store with the highest fitness value is the store loja9, which requested 26 types of products, amounting to 14952 units. The vehicle employed by this store, visited 1 store and collected 898 units, traveling 22.24 kilometers. The store with the lowest

fitness value is the store loja7, which requested 2 types of products, amounting to 836 units. The vehicle employed by this store, visited 1 store and collected 836 units, traveling 109.50 kilometers (table 14).

Table 14: Statistics for each request with the Genetic algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness
loja7	2	loja10	1	109.50	836	836	7.63
loja10	32	loja8, loja1	2	154.21	17111	2622	17.00
loja4	14	loja1	1	32.28	7119	1984	61.46
loja9	26	loja3	1	22.24	14952	898	40.38
loja8	22	loja7, loja9, loja3, loja6	4	129.81	11125	2891	22.27
loja6	10	loja10	1	108.28	4854	1036	9.57
loja2	27	loja3	1	48.28	14742	1537	31.84

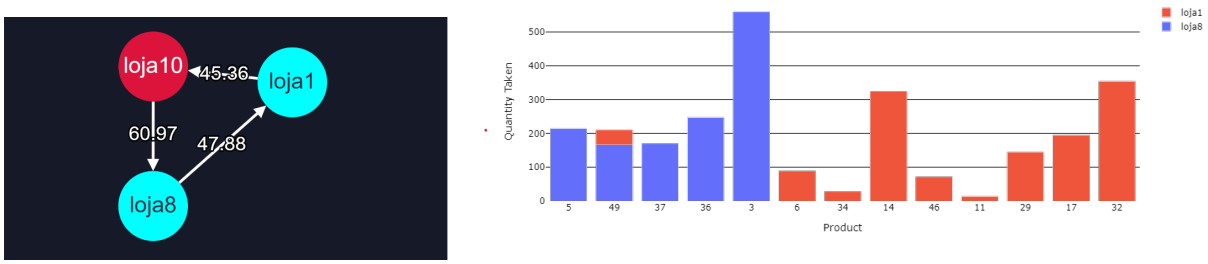
Representation of the solution



(a) Route for the store loja7's request.

(b) Products and respective quantities taken at each visited store for store loja7's request.

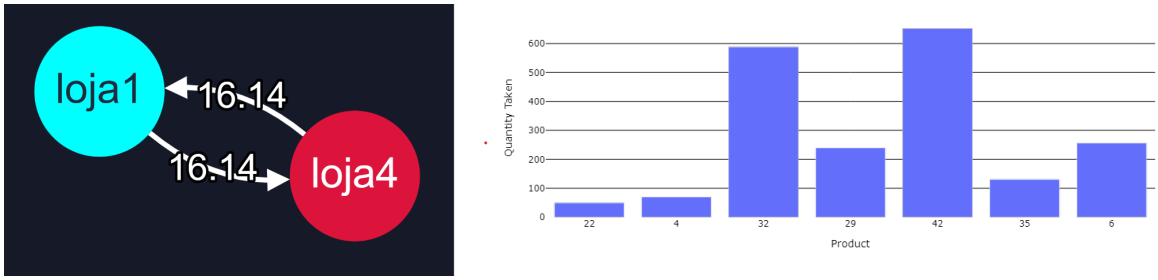
Figure 53: Requesting store's loja7 part of the solution.



(a) Route for the store loja10's request.

(b) Products and respective quantities taken at each visited store for store loja10's request.

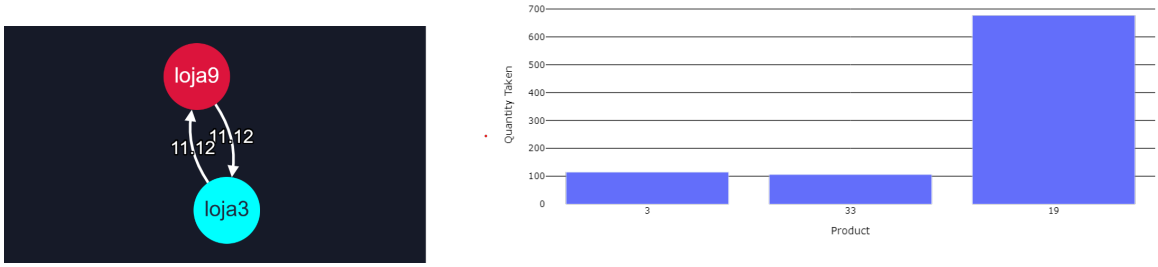
Figure 54: Requesting store's loja10 part of the solution.



(a) Route for the store loja4's request.

(b) Products and respective quantities taken at each visited store for store loja4's request.

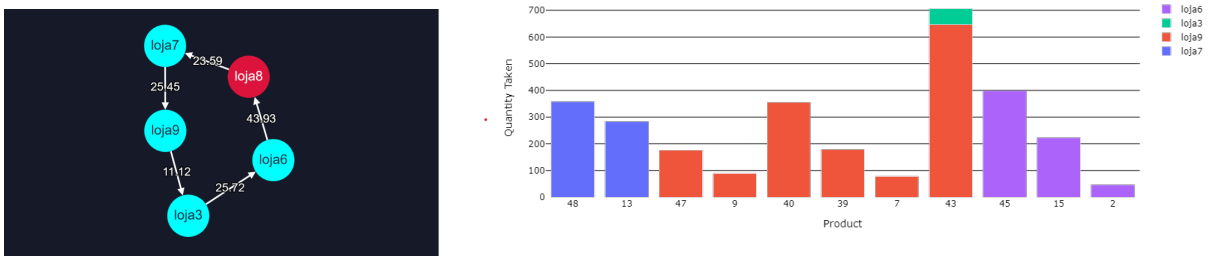
Figure 55: Requesting store's loja4 part of the solution.



(a) Route for the store loja9's request.

(b) Products and respective quantities taken at each visited store for store loja9's request.

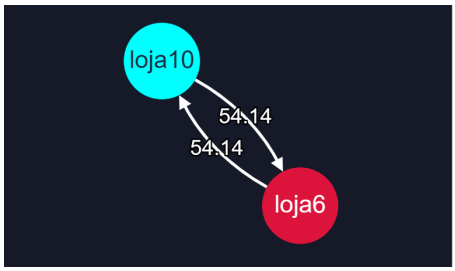
Figure 56: Requesting store's loja9 part of the solution.



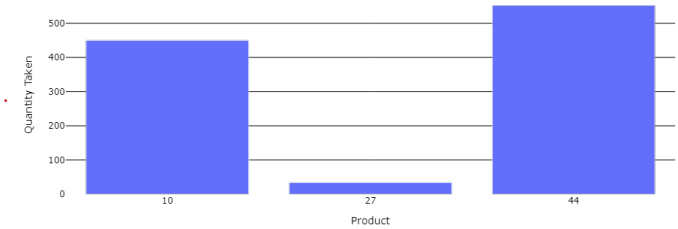
(a) Route for the store loja8's request.

(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 57: Requesting store's loja8 part of the solution.

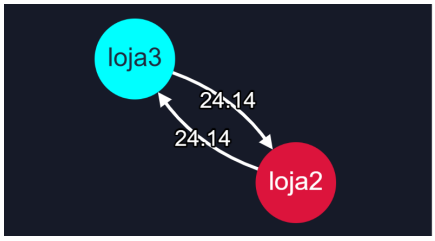


(a) Route for the store loja6's request.

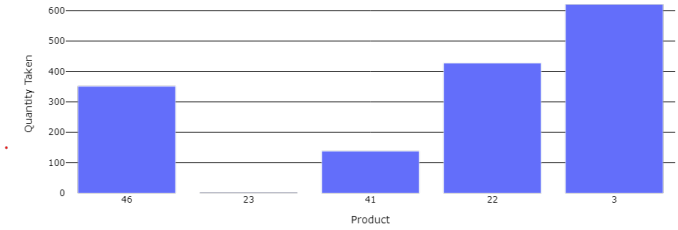


(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 58: Requesting store's loja6 part of the solution.



(a) Route for the store loja2's request.



(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 59: Requesting store's loja2 part of the solution.

Genetic Algorithm with Elitism

The initial population used in this instance of the algorithm is the same as the one used in the genetic algorithm instance 4.1.2.

Statistics of the solution

In this fitness graph 52, it is represented the average fitness of the population through the generations of the Genetic algorithm. The fitness fluctuates approximately between 52 to 183.56, having values with an improved fitness over the initial population. In the first 20 generations, the average of the fitness values of the population increased consistently from around 52 to 188, slightly fluctuating around the latter until the end, reaching a maximum value of 183.56.

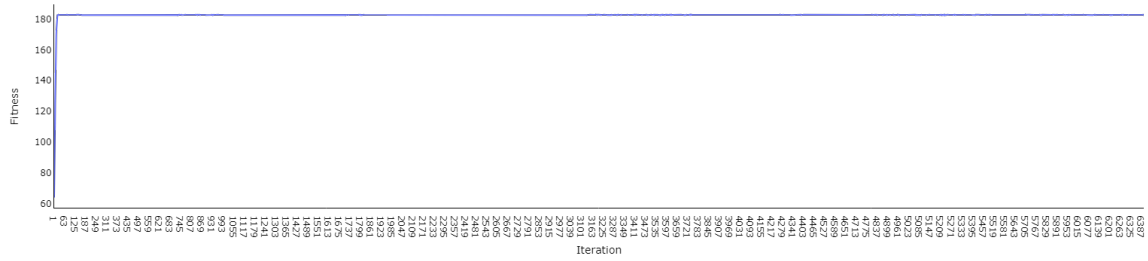


Figure 60: Fitness of the solutions found by the Genetic algorithm with Elitism during its iterations.

The Genetic algorithm with Elitism's solution has a fitness of 183.56. It was requested for 133 products across all requesting stores, of which 18 of these product requests were completely fulfilled. In total, 13362 units were collected, accounting to an average fulfillment of 23%. There were 15 store visits, accumulating a travel distance of 753.56 kilometers (table 15).

Table 15: Statistics of the Solution.

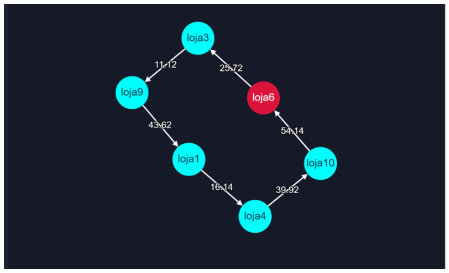
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
133	18	15	753.56	13362	0.23	183.56	70739

The requesting store that accounted for the highest value of the fitness function was the store loja4, which requested 14 products, amounting to 7119 units. The vehicle employed by loja4, visited 1 store and collected 1984 units, traveling 32.28 kilometers. The second store with the highest fitness value is the store loja9, which requested 26 types of products, amounting to 14952 units. The vehicle employed by this store, visited 1 store and collected 1159 units, traveling 22.24 kilometers. The store with the lowest fitness value is the store loja7, which requested 2 types of products, amounting to 836 units. The vehicle employed by this store, visited 1 store and collected 771 units, traveling 109.50 kilometers (table 42).

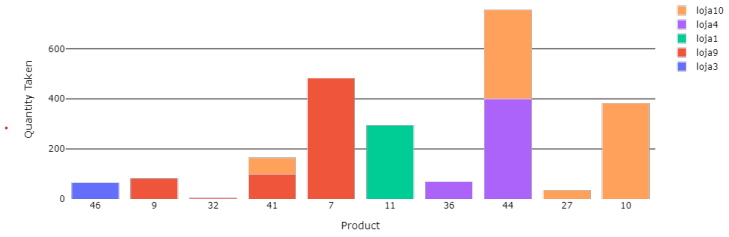
Table 16: Statistics for each request for the Genetic algorithm with Elitism's solution.

Requesting Store	Number of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness
loja6	10	loja3, loja9, loja1, loja4, loja10	5	190.66	4854	2332	12.23
loja2	27	loja6, loja10, loja4, loja1	4	238.24	14742	4767	20.01
loja8	22	loja4, loja7	2	89.12	11125	775	8.70
loja7	2	loja10	1	109.50	836	771	7.04
loja10	32	loja3	1	71.52	17111	1574	22.01
loja9	26	loja3	1	22.24	14952	1159	52.11
loja4	14	loja1	1	32.28	7119	1984	61.46

Representation of the solution

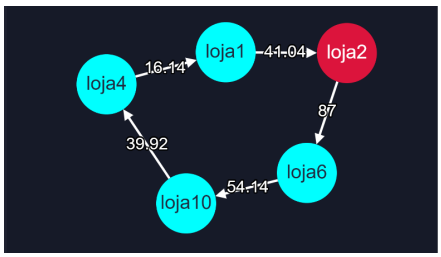


(a) Route for the store loja6's request.

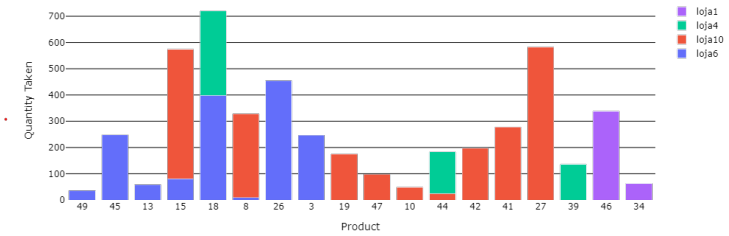


(b) Products and respective quantities taken at each visited store for store loja6's request.

Figure 61: Requesting store's loja6 part of the solution.

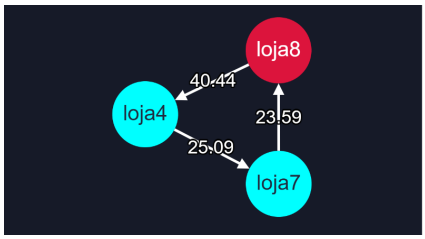


(a) Route for the store loja2's request.

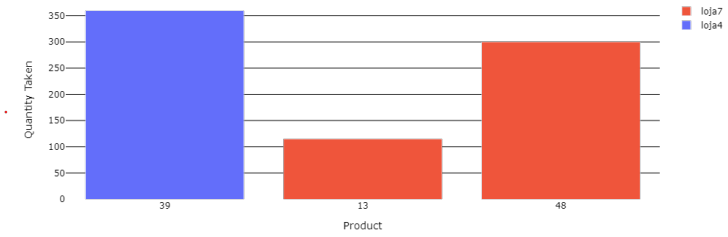


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 62: Requesting store's loja2 part of the solution.

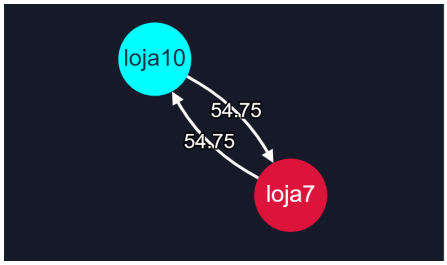


(a) Route for the store loja8's request.

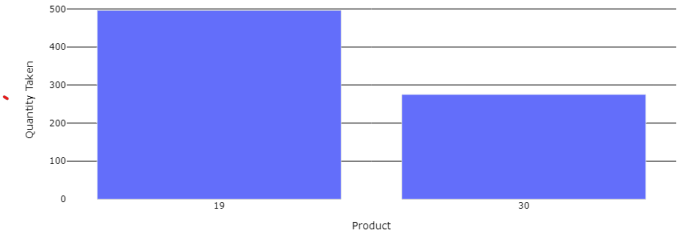


(b) Products and respective quantities taken at each visited store for store loja8's request.

Figure 63: Requesting store's loja8 part of the solution.

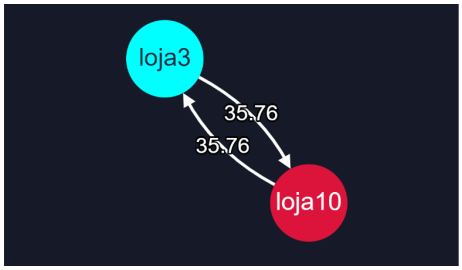


(a) Route for the store loja7's request.

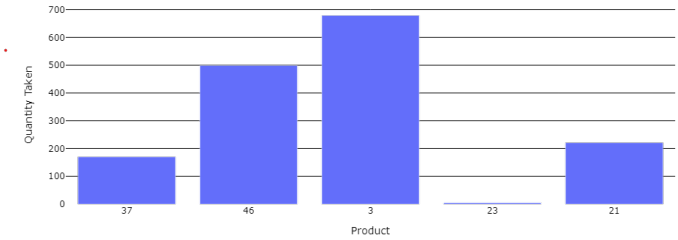


(b) Products and respective quantities taken at each visited store for store loja7's request.

Figure 64: Requesting store's loja7 part of the solution.

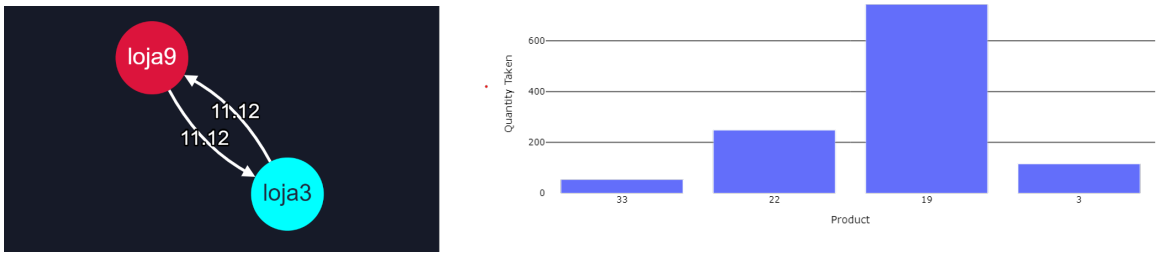


(a) Route for the store loja10's request.



(b) Products and respective quantities taken at each visited store for store loja10's request.

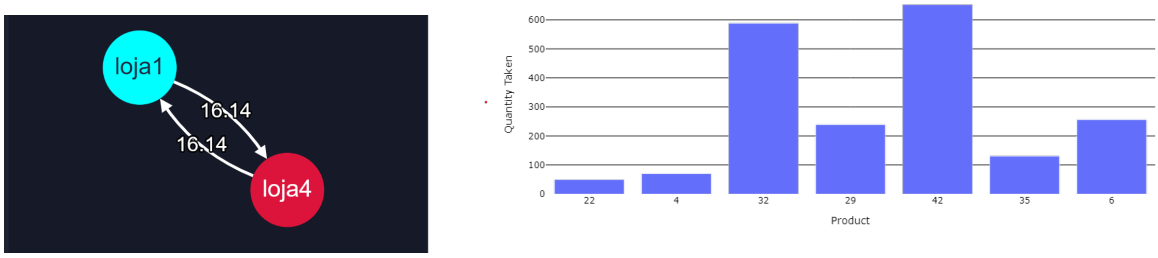
Figure 65: Requesting store's loja10 part of the solution.



(a) Route for the store loja9's request.

(b) Products and respective quantities taken at each visited store for store loja9's request.

Figure 66: Requesting store's loja9 part of the solution.



(a) Route for the store loja4's request.

(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 67: Requesting store's loja4 part of the solution.

4.1.3 Comparison of Solutions

As is observable from the comparison table 17, the algorithm that generated the solution with the best fitness was the genetic algorithm with random initial population. Additionally, both the Genetic Algorithm and the Genetic Algorithm with Elitism were tested with another initial population. This initial population is composed by 1000 solutions generated by the Dynamic2 algorithm with an average fitness of 125.05. Still, the instances of the genetic algorithms with the random initial population, found solutions with higher fitness values than the ones with a initial population of solutions generated by the Dynamic2 algorithm. This might be because having an initial population compromised of just random solutions can allow the genetic algorithm to explore the solution search space more effectively. Even though the Dynamic2 solutions are usually significantly better solutions than the solutions generated randomly, it makes the exploration of the solution search space much more limited since the logic for building the different Dynamic2 solu-

tions is similar. Despite this, the genetic algorithms with or without elitism manages to improve the initial population consisting of solutions generated by the Dynamic2 algorithm. Comparing the genetic algorithm to the genetic algorithm with elitism, the latter has a faster convergence speed. It is also observable by the table that the genetic algorithm with elitism found slightly better solutions than the genetic algorithm, excluding the instances where the genetic algorithm had a random initial population. In those instances, for this scenario, the genetic algorithm's best solution found was better than the one found by the genetic algorithm with elitism. The best solution found by the genetic algorithm with random initial population has a fitness value 56.14% greater than the baseline Dynamic2 solution.

Regarding the single solution algorithms, the Simulated Annealing algorithm with jumps in the solution search space to Dynamic2 solutions, or simulated_annealing_d_2 in the comparison table 17, found the solution with the highest fitness value out of all the other single solution algorithms, with a value 31.61% greater than the baseline solution. This solution proved having a greater fitness value than the best one found by the Simulated Annealing algorithm, or simulated_annealing in the comparison table, which, in turn, had a fitness value 17.10% greater than the baseline solution. The Tabu Search algorithm with Intensification and Diversification, or tabu_search2 in the table 17, was the single solution algorithm that found the second best solution with a fitness 26.32% greater than the baseline solution, with the intensification mechanism proving useful when comparing to the tabu_search algorithm's instance in the comparison table, that found its best solution with a fitness value of 138.62, which translates to being 13.83% greater than the baseline solution.

Table 17: Table with the instances of the algorithms previously presented for Scenario 1

Algorithm	Instance	Fitness	Nº of Solutions Generated	Parameters	Iterations	Initial Solution/Population
dynamic2	1	121.77	1	Rank: 1	1	None
simulated_annealing	1	142.61	8071611	Initial Temp: 1000, Cooling Rate: 0.999, Stop Temp: 0.0001, Temp Iterations: 500	16111	dynamic2
simulated_annealing_d_2	1	160.30	5434347	Initial Temp: 1000, Cooling Rate: 0.999, Stop Temp: 0.0001, Temp Iterations: 500, max_iter_without_new_sol: 1000	10847	dynamic2
tabu_search	1	138.62	8071661	Tabu List: 50, Max Iterations without Improvement: 10, subset_size: 500	17623	dynamic2
tabu_search2	1	153.87	8071684	Tabu List: 50, Elite List: 20, Max Iterations without Improvement: 10, subset_size: 500	17063	dynamic2
genetic_algorithm	1	169.41	8072210	Pop. Size: 1000, Crossover Rate: 0.6, Mutation Rate: 0.4, Selection: tourney_selection(size=5)	8074	d2: 1000 ; random: 0
genetic_algorithm_elitism	1	170.13	8072106	Pop. Size: 1000, Crossover Rate: 0.6, Mutation Rate: 0.4, Selection: tourney_selection(size=5), Elitism Size: 100	6409	d2: 1000 ; random: 0
genetic_algorithm	2	190.15	8071638	Pop. Size: 1000, Crossover Rate: 0.6, Mutation Rate: 0.4, Selection: tourney_selection(size=5)	8074	random: 1000
genetic_algorithm_elitism	2	183.56	8072250	Pop. Size: 1000, Crossover Rate: 0.6, Mutation Rate: 0.4, Selection: tourney_selection(size=5), Elitism Size: 100	6400	random: 1000

4.2 Scenario 2

4.2.1 Characteristics of the Problem

Products and Quantity Needed for each Requesting Store

In the network of stores, 113 stores are in need of at least one product. There is a necessity for 50 types of products, all of them requested by more than one of the 113 requesting stores. The accumulated requested quantities range from approximately 7000 to more than 13000 units of a product (figure 68).

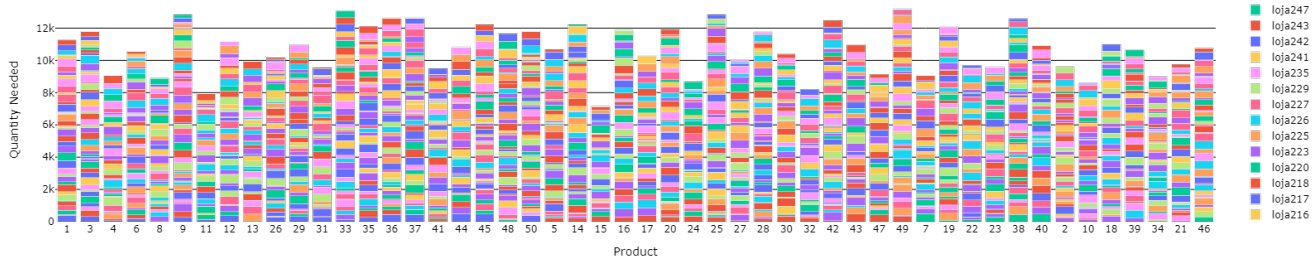


Figure 68: Bar chart representing the needs of the requesting stores.

Stock of the Requested Products

In the network of stores, there are 50 types of products in stock, and the accumulated quantities of a product in at least one store vary approximately from 18000 to 30000 units. It is important to mention that the distribution of products and their quantities is not uniform across all stores in the network (figure 69).

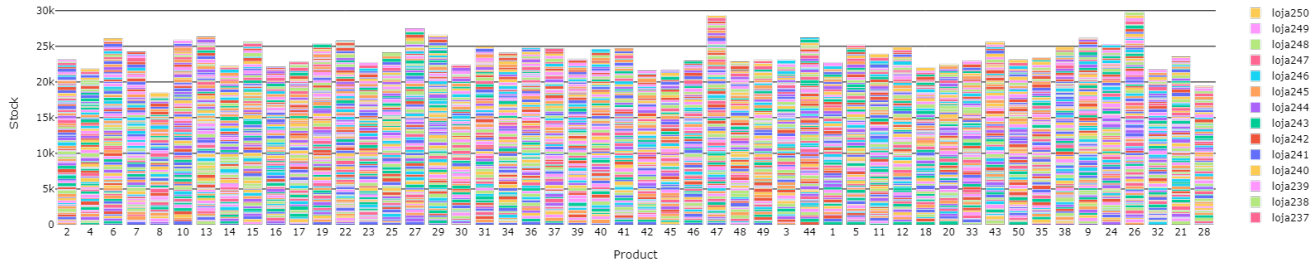


Figure 69: Bar chart representing the stock of the requested products.

Distances between the Stores

The distance between two stores is distributed randomly from a minimum of 250 kilometers to a maximum of 500 kilometers (figure 70).

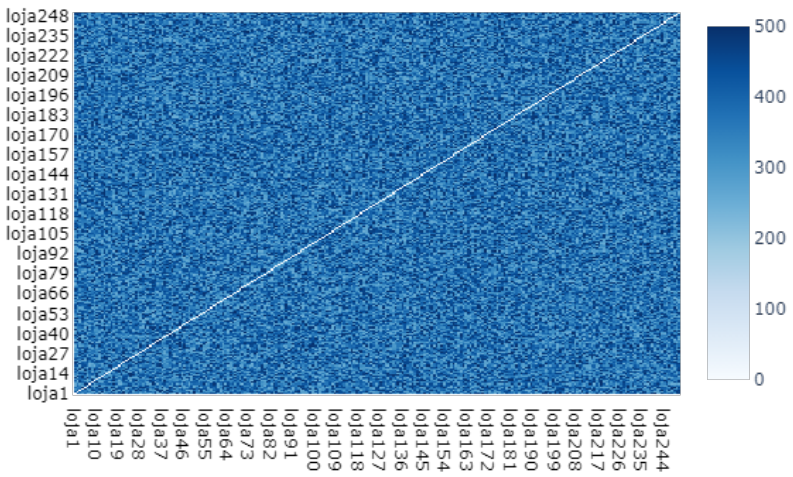


Figure 70: Heatmap representing the distances between the stores in the network.

4.2.2 Solutions and Results:

Due to the scenario's size, which has 250 stores in the network and 113 requests, for the best solutions found by the developed algorithms, only the fitness graph, table with statistics of the solution, and statistics table of the request with the highest fitness and its representation will be presented. It is relevant to mention that for a given instance of an algorithm, if no improvement over the initial solution is found, only the fitness graph is shown. This is because the initial solution would be the best solution found, which at that point had already been presented along with its statistics.

Furthermore, since the number of stores and requests is significantly higher than in the other scenarios, so is the solution search space. Consequently, the search becomes more computationally expensive than other smaller scenarios. Due to time restrictions, the instance of the algorithms tested for this scenario had fewer iterations to search for solutions. Given all these factors, finding promising solutions in this case becomes more challenging for the developed algorithms than in smaller scenarios.

In this scenario, the single solution-based algorithms were tested using a Dynamic2 rank 1 solution as the initial solution. These algorithms did not manage to find any solution with a higher fitness value than the one the initial solution had. So, in addition to testing these algorithms using a Dynamic2 solution as an initial solution, the algorithms were also tested using a feasible solution generated through random

mechanisms.

Dynamic2

Statistics of the Solution

The Dynamic2 algorithm's solution has a fitness of 151.75. It was requested for 2118 products across all requesting stores, of which 2118 were completely fulfilled. In total, 535393 units were collected, accounting to an average fulfillment of 100%. There were 1278 store visits, accumulating a travel distance of 443883.89 kilometers (table 18).

Table 18: Statistics for the solution.

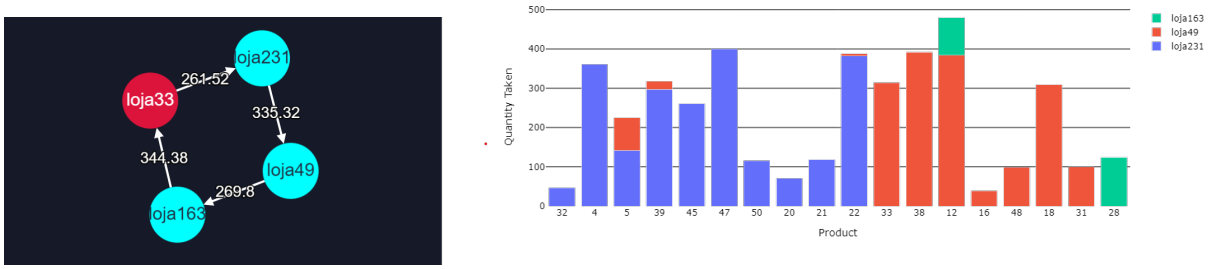
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	2118	1278	443883.89	535393	1	151.75	535393

The requesting store that accounted for the highest value of the fitness function was the store loja33, which requested 18 products, amounting to 4161 units. The vehicle employed by loja33, visited 3 distinct stores and collected 4161 units, traveling 1211.02 kilometers.

Table 19: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja33	18	loja231, loja49, loja163	3	1211.02	4161	3.44	4161

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja33's request. (b) Products and respective quantities taken at each visited store for store loja33's request.

Figure 71: Requesting store's loja33 part of the solution.

Random solution

This feasible solution generated through random mechanisms is used as an initial solution for some of the developed algorithms.

Statistics of the Solution

The random solution has a fitness of 48.53. It was requested for 2118 products across all requesting stores, of which 1898 were completely fulfilled. In total, 492781 units were collected, accounting to an average fulfillment of 93%. There were 2960 store visits, accumulating a travel distance of 1152867.58 kilometers (table 20).

Table 20: Statistics for the solution.

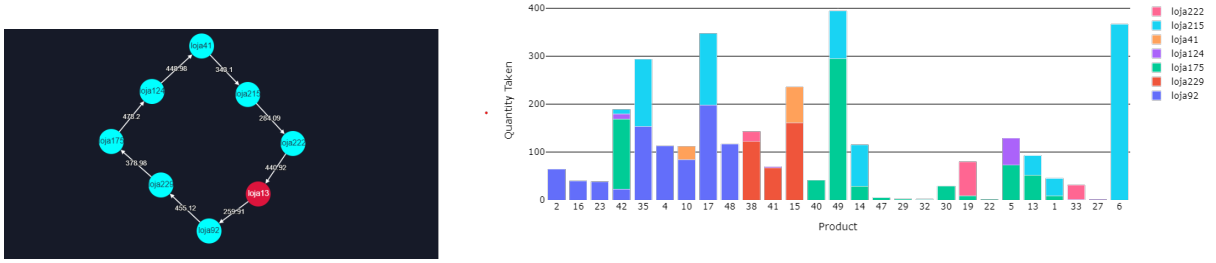
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	1898	2960	1152867.58	492781	0.93	48.53	535393

The requesting store that accounted for the highest value of the fitness function was the store loja13, which requested 32 products, amounting to 7005 units. The vehicle employed by loja13, visited 7 distinct stores and collected 3100 units, traveling 3089.30 kilometers.

Table 21: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja13	32	loja92, loja229, loja175, loja124, loja41, loja215, loja222	7	3089.30	3100	1.00	7005

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja13's request.

(b) Products and respective quantities taken at each visited store for store loja13's request.

Figure 72: Requesting store's loja13 part of the solution.

Simulated Annealing

Statistics of the Solution using the Dynamic2 algorithm's solution as initial solution

In this fitness graph 73, in approximately the first 1000 iterations the fitness decreases from 151.75, which is the value of the initial Dynamic2 solution, to approximately 46. From then, the fitness values fluctuates approximately from 46 to 55. No solution with a fitness value greater than the initial solution was found.

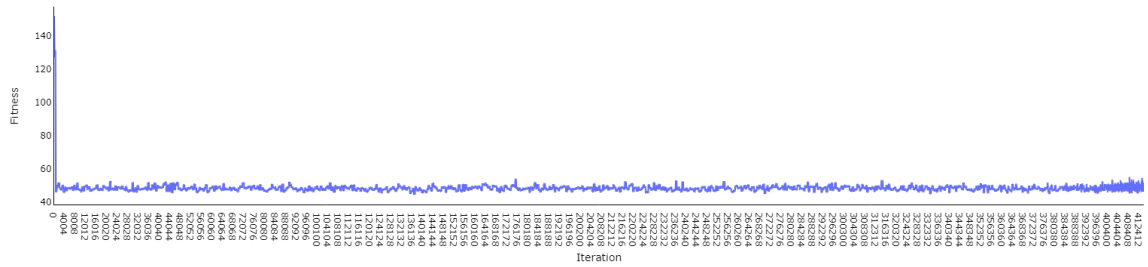


Figure 73: Fitness of the solutions found by the Simulated Annealing algorithm with Dynamic2 initial solution during its iterations.

Statistics of the Solution using the random solution as initial solution

In this fitness graph 21, it is represented the fitness through the iterations of the Simulated Annealing algorithm. The fitness values range from 43.15 to 53.51, having values with an improved fitness over the initial random solution throughout the iterations. The majority of iterations have fitness values varying approximately between 47 and 50.

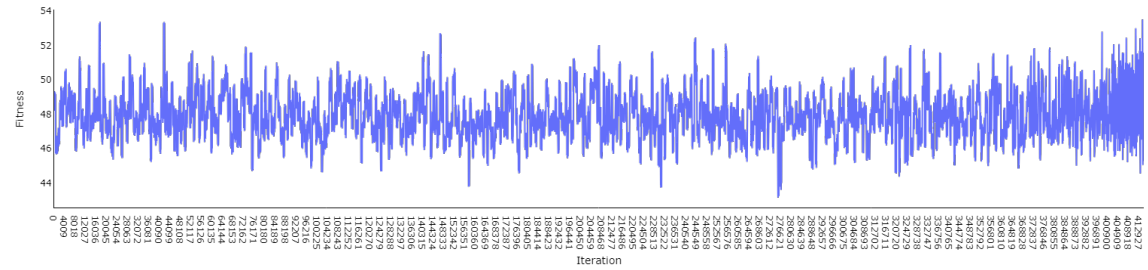


Figure 74: Fitness of the solutions found by the Simulated Annealing algorithm with random initial solution during its iterations.

The Simulated Annealing algorithm's solution has a fitness of 53.51. It was requested for 2118 products across all requesting stores, of which 1884 were completely fulfilled. In total, 490625 units were collected,

accounting to an average fulfillment of 93%. There were 2840 store visits, accumulating a travel distance of 1085240.10 kilometers (table 3).

Table 22: Statistics for the solution.

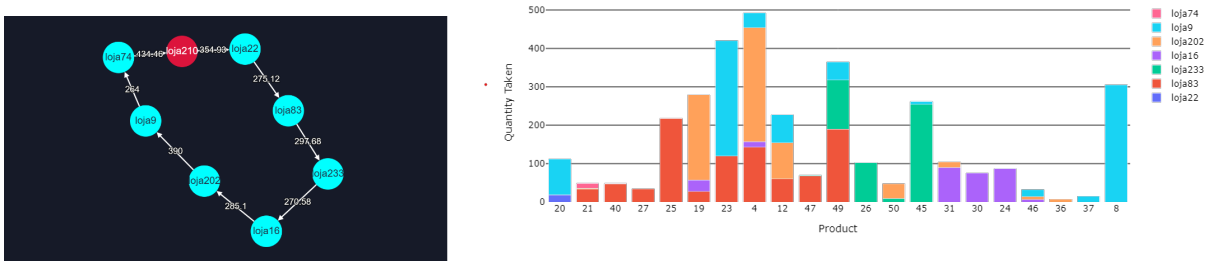
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	1884	2840	1085240.1	490625	0.93	53.51	535393

The requesting store that accounted for the highest value of the fitness function was the store loja210, which requested 22 products, amounting to 6482 units. The vehicle employed by loja210, visited 7 distinct stores and collected 3353 units, traveling 2571.87 kilometers.

Table 23: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja210	22	loja22, loja83, loja233, loja16, loja202, loja9, loja74	7	2571.87	3353	1.30	6482

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja210's request.

(b) Products and respective quantities taken at each visited store for store loja210's request.

Figure 75: Requesting store's loja210 part of the solution.

Simulated Annealing using Dynamic2 solutions for jumps in the solution search space

In this fitness graph 73, it is evident that the number of peaks is higher compared to the Simulated Annealing instance in subsection 4.2.2. This can be explained by the fact that the algorithm in question jumps in the solution search space to solutions generated by the Dynamic2 algorithm instead of jumping to random points in the search space. As mentioned before, these solutions generated by the Dynamic2 algorithms are generally better than solutions generated through random mechanisms. When these jumps happen,

they can be further improved. However, no solution with an improved fitness over the initial Dynamic2 rank 1 solution is found.

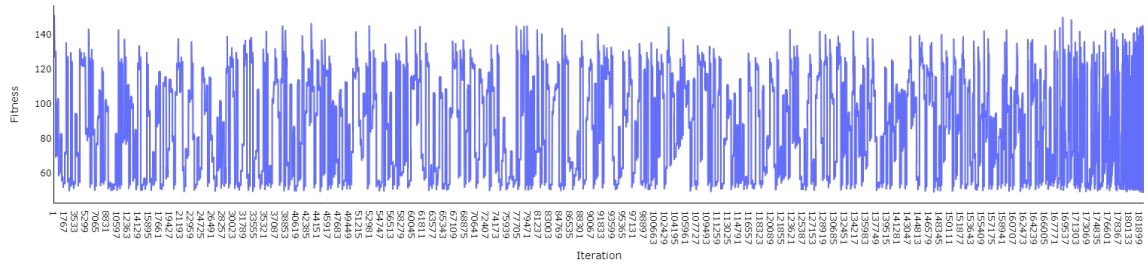


Figure 76: Fitness of the solutions found by the Simulated Annealing algorithm using Dynamic2 solutions during its iterations.

Tabu Search

Statistics of the Solution using the Dynamic2 algorithm's solution as initial solution

In this fitness graph 77, in the first 11 iterations the fitness decreases from 151.75, which is the value of the initial Dynamic2 solution, to approximately 50. From then, the fitness values fluctuates approximately from 46 to 52. No solution with a fitness value greater than the initial solution was found.

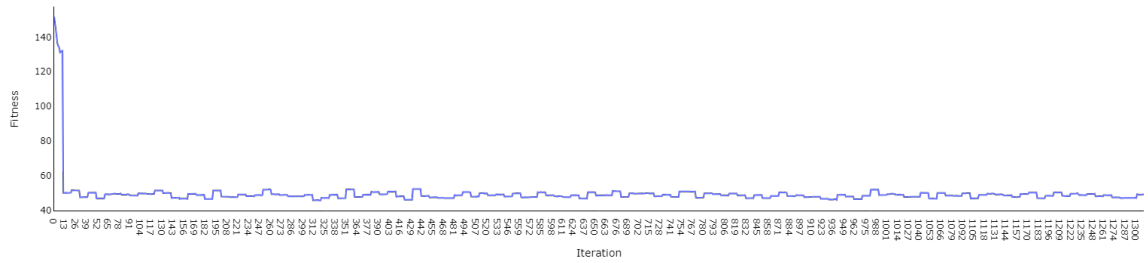


Figure 77: Fitness of the solutions found by the Tabu Search algorithm with Dynamic2 initial solution during its iterations.

Statistics of the Solution using the random solution as initial solution

In this fitness graph 78, it is represented the fitness through the iterations of the Tabu Search algorithm. The fitness fluctuates approximately between 45.31 to 53.59, having values with an improved fitness over the initial random solution. The majority of iterations have fitness values varying approximately between 46 and 50.

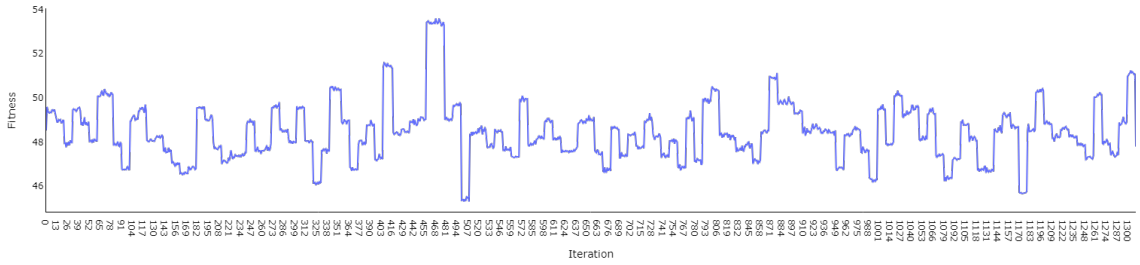


Figure 78: Fitness of the solutions found by the Tabu Search algorithm with random solution as initial solution during its iterations.

The Tabu Search algorithm's solution has a fitness of 53.59. It was requested for 2118 products across all requesting stores, of which 1788 were completely fulfilled. In total, 473205 units were collected, accounting to an average fulfillment of 90%. There were 2696 store visits, accumulating a travel distance of 1041973.18 kilometers (table 24).

Table 24: Statistics for the solution.

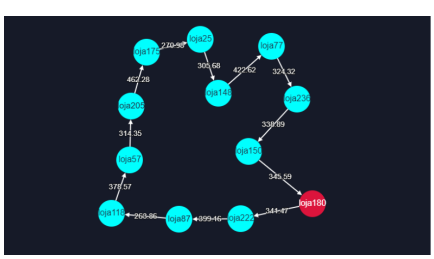
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	1788	2696	1041973.18	473205	0.90	53.59	535393

The requesting store that accounted for the highest value of the fitness function was the store loja180, which requested 34 products, amounting to 9131 units. The vehicle employed by loja180, visited 21 distinct stores and collected 5510 units, traveling 4167.77 kilometers.

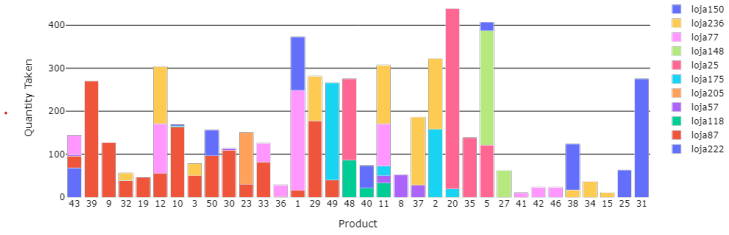
Table 25: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja180	34	loja222, loja87, loja118, loja57, loja205, loja175, loja25, loja148, loja77, loja236, loja150	11	4167.77	5510	1.32	9131

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja180's request.



(b) Products and respective quantities taken at each visited store for store loja180's request.

Figure 79: Requesting store's loja180 part of the solution.

Tabu Search with Intensification and Diversification

Statistics of the Solution using the Dynamic2 algorithm's solution as initial solution

In this fitness graph 80, it is evident that the number of peaks is higher compared to the Tabu Search instance in subsection 4.2.2. This might be due to the intensification mechanism by the way of elitism mentioned before. This consists in selecting one of the best found solutions in the medium-term memory as the current solution in order to intensify the search in promising areas of the solution search space. However, no solution was found with a greater fitness value than the initial Dynamic2 rank 1 solution.

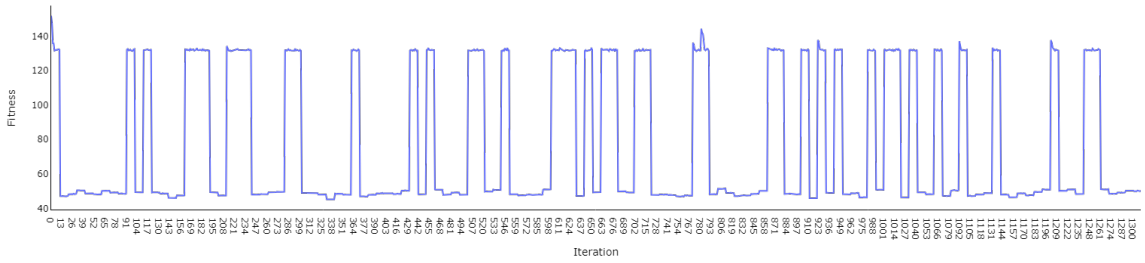


Figure 80: Fitness of the solutions found by the Tabu Search with Intensification and Diversification algorithm with Dynamic2 initial solution during its iterations.

Statistics of the Solution using the random solution as initial solution

Like in the fitness graph 77, in this graph 81 it is observable that the number of peaks is higher compared to the Tabu Search instance in subsection 4.2.2, which might be due to the mentioned intensification mechanism.

The fitness fluctuates approximately between 45 to 51.72, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 48 and 51.

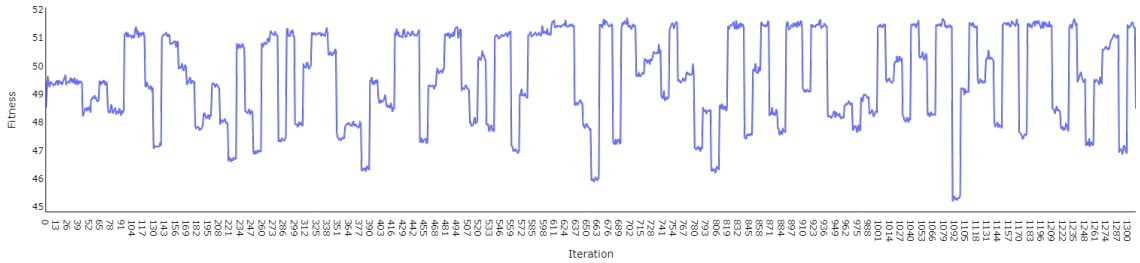


Figure 81: Fitness of the solutions found by the Tabu Search with Intensification and Diversification algorithm with random solution as initial solution during its iterations.

The Tabu Search with Intensification and Diversification algorithm's solution has a fitness of 51.72. It was requested for 2118 products across all requesting stores, of which 1788 were completely fulfilled. In total, 470154 units were collected, accounting to an average fulfillment of 89%. There were 2695 store visits, accumulating a travel distance of 1035812.54 kilometers (table 26).

Table 26: Statistics for the solution.

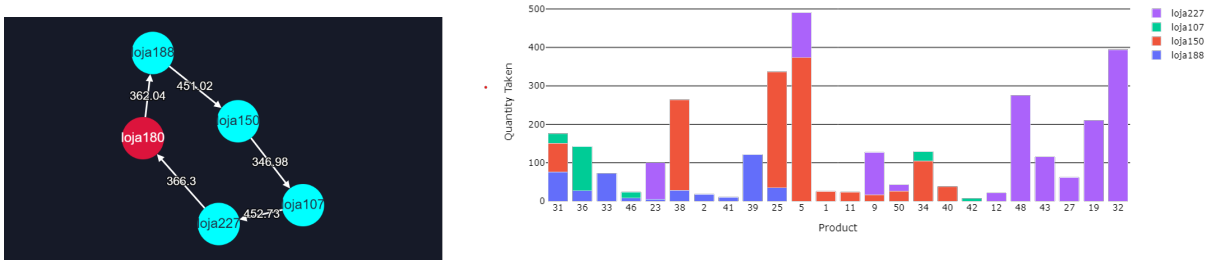
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	1788	2695	1035812.54	470154	0.89	51.72	535393

The requesting store that accounted for the highest value of the fitness function was the store loja180, which requested 34 products, amounting to 9131 units. The vehicle employed by loja180, visited 4 distinct stores and collected 3223 units, traveling 1979.07 kilometers.

Table 27: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja180	34	loja188, loja150, loja107, loja227	4	1979.07	3223	1.63	9131

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja180's request. (b) Products and respective quantities taken at each visited store for store loja180's request.

Figure 82: Requesting store's loja180 part of the solution.

Genetic Algorithm

Statistics of the Solution using a random population as initial population

In this instance, an initial population of 1000 solutions obtained through random procedures is used. Every solution in this initial population is feasible. The average fitness of the mentioned population is 48.16.

In this fitness graph 83, it is represented the average fitness of the population through the generations of the Genetic algorithm. In the second generation, the average of the fitness of the population decreases to negative values. From then it increases to an average of 54 in the ninth generation. From this generation, the fitness fluctuates approximately between 54 to 59.33, having values with an improved fitness over the initial population.

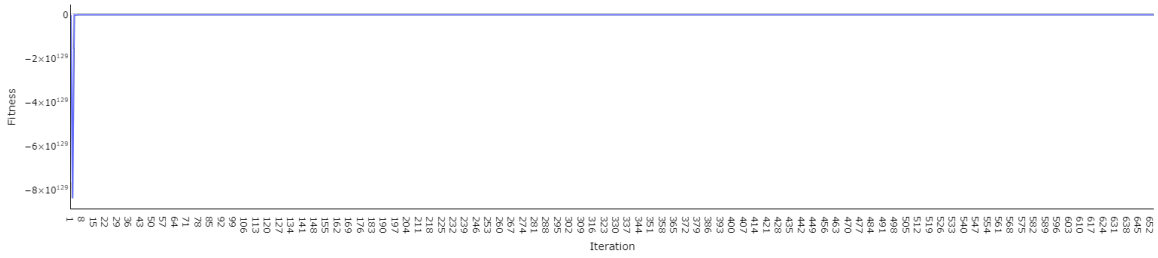


Figure 83: Fitness of the solutions found by the Genetic Algorithm during its iterations.

The Genetic algorithm's solution has a fitness of 59.33. It was requested for 2118 products across all requesting stores, of which 1780 were completely fulfilled. In total, 464164 units were collected, accounting to an average fulfillment of 89%. There were 2629 store visits, accumulating a travel distance of 938043.93

kilometers (table 28).

Table 28: Statistics for the solution.

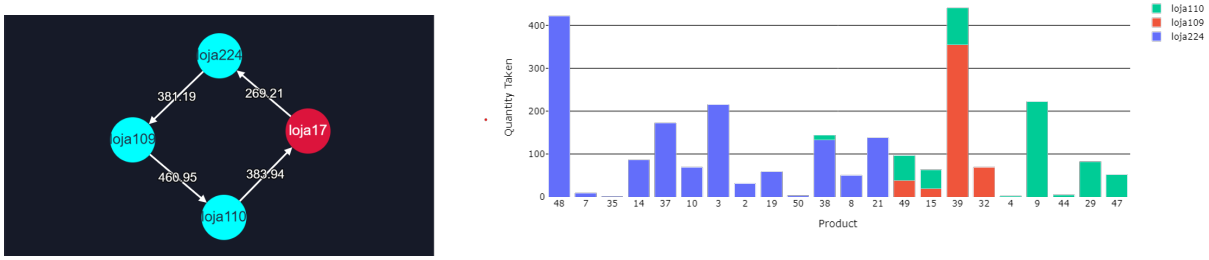
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	1780	2629	938043.93	464164	0.89	59.33	535393

The requesting store that accounted for the highest value of the fitness function was the store loja17, which requested 31 products, amounting to 6375 units. The vehicle employed by loja17, visited 3 distinct stores and collected 2432 units, traveling 1495.29 kilometers.

Table 29: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja17	31	loja224, loja109, loja110	3	1495.29	2432	1.63	6375

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja17's request.

(b) Products and respective quantities taken at each visited store for store loja17's request.

Figure 84: Requesting store's loja17 part of the solution.

Statistics of the Solution using a Dynamic2 population as the initial population

In this instance, an initial population of 1000 Dynamic2 solutions is used. Every solution in this initial population is feasible. The average fitness of the mentioned population is 151.66.

In this fitness graph 85, it is represented the average fitness of the population through the generations of the Genetic algorithm. The average of the fitness of the population increases consistently throughout the generations, reaching a maximum of 152.93, having values with an improved fitness of the initial population.

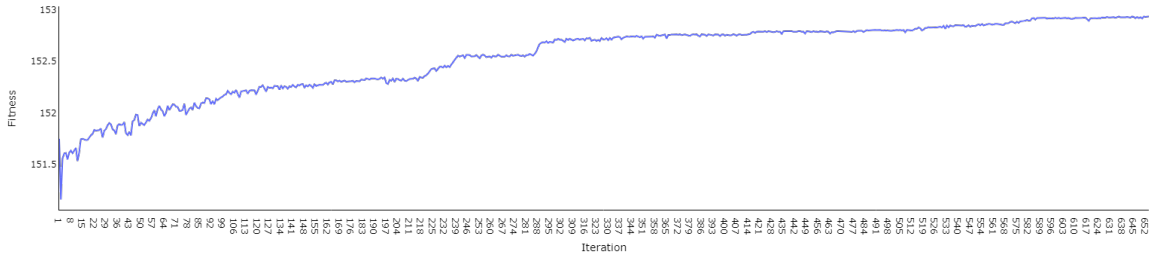


Figure 85: Fitness of the solutions found by the Genetic Algorithm during its generations.

The Genetic algorithm's solution has a fitness of 152.93. It was requested for 2118 products across all requesting stores, of which 2118 were completely fulfilled. In total, 535393 units were collected, accounting to an average fulfillment of 100%. There were 1275 store visits, accumulating a travel distance of 438128.78 kilometers (table 34).

Table 30: Statistics of the Solution.

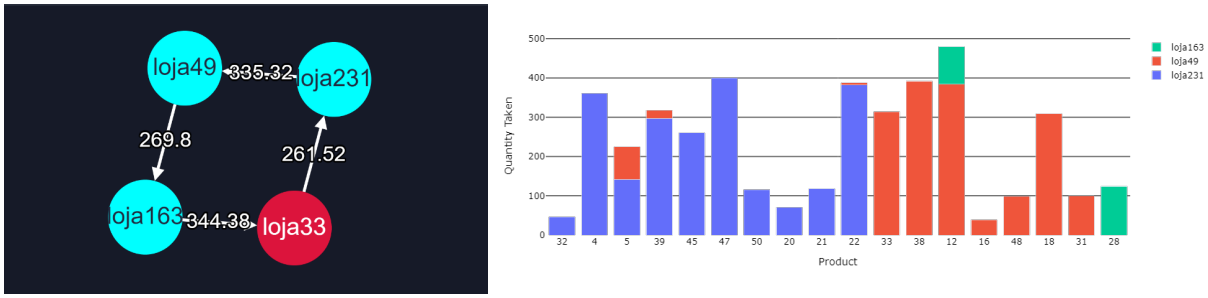
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	2118	1275	438128.78	535393	1	152.93	535393

The requesting store that accounted for the highest value of the fitness function was the store loja33, which requested 18 products, amounting to 4161 units. The vehicle employed by loja33, visited 3 distinct stores and collected 4161 units, traveling 1211.02 kilometers.

Table 31: Statistics for the Requesting Store with the highest fitness value of the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja33	18	loja231, loja49, loja163	3	1211.02	4161	3.44	4161

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja33's request. (b) Products and respective quantities taken at each visited store for store loja33's request.

Figure 86: Requesting store's loja33 part of the solution.

Genetic Algorith with Elitism

Statistics of the Solution using a random population as initial population

The initial population used in this instance of the algorithm is the same as the one used in the genetic algorithm instance 4.2.2.

In this fitness graph 87, it is represented the average fitness of the population through the generations of the Genetic algorithm. In the second generation, the average of the fitness of the population decreases to negative values. From then it increases to an average of 54 in the seventh generation. From this generation, the fitness fluctuates approximately between 54 to 59.61, having values with an improved fitness over the initial population.

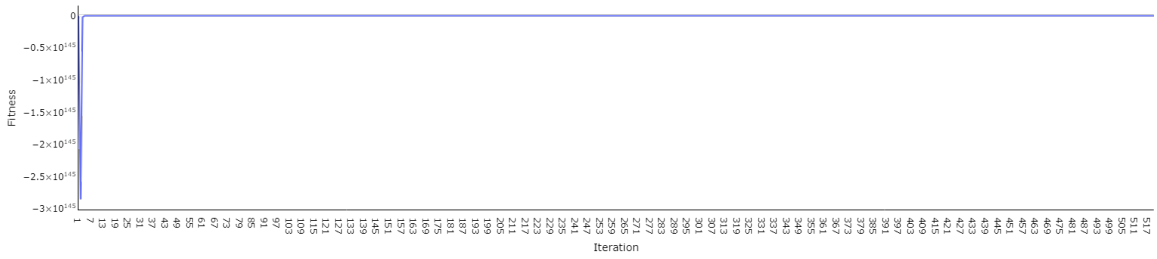


Figure 87: Fitness of the solutions found by the Genetic Algorithm with Elitism during its iterations.

The Genetic algorithm with Elitism's solution has a fitness of 59.61. It was requested for 2118 products across all requesting stores, of which 1780 were completely fulfilled. In total, 535393 units were collected, accounting to an average fulfillment of 89%. There were 2629 store visits, accumulating a travel distance

of 937193.30 kilometers (table 32).

Table 32: Statistics for the solution.

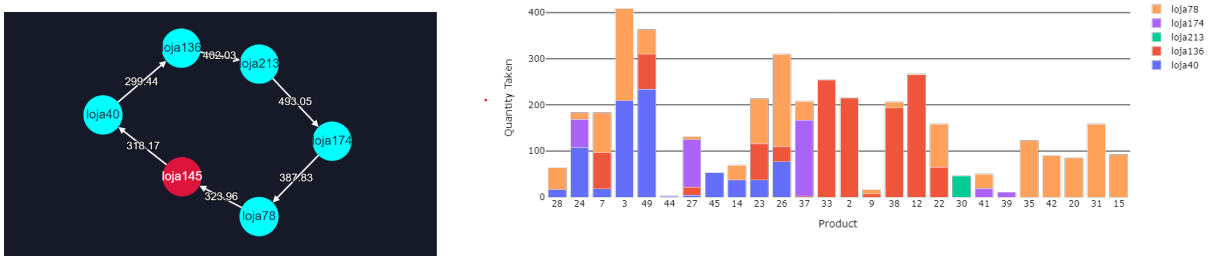
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	1780	2629	937193.30	464164	0.89	59.61	535393

The requesting store that accounted for the highest value of the fitness function was the store loja145 which requested 28 products, amounting to 7392 units. The vehicle employed by loja145 visited 5 distinct stores and collected 3958 units, traveling 2224.48 kilometers.

Table 33: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja145	28	loja40, loja136, loja213, loja174, loja78	5	2224.48	3958	1.78	7392

Representation of the request with the highest fitness value of the solution



(a) Route for the store loja145's request. (b) Products and respective quantities taken at each visited store for store loja145's request.

Figure 88: Requesting store's loja145 part of the solution.

Statistics of the Solution using a Dynamic2 population as the initial population

In this fitness graph 89, it is represented the average fitness of the population through the generations of the Genetic algorithm with Elitism. The average of the fitness of the population increases consistently throughout the generations, reaching a maximum of 153.27, having values with an improved fitness of the initial population.

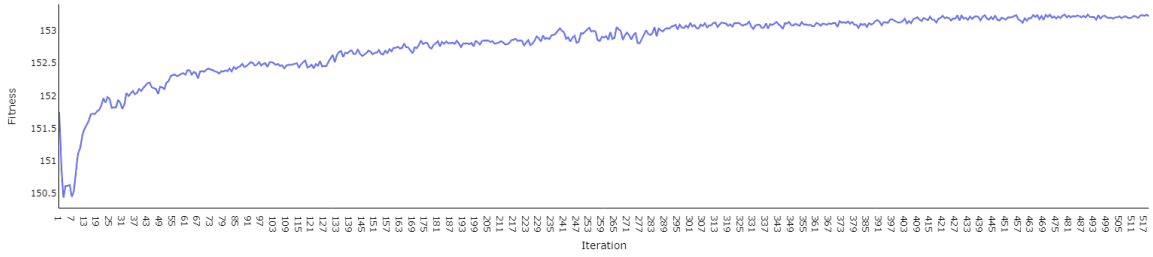


Figure 89: Fitness of the solutions found by the Genetic Algorithm with Elitism during its iterations.

The Genetic algorithm with Elitism's solution has a fitness of 153.27. It was requested for 2118 products across all requesting stores, of which 2118 were completely fulfilled. In total, 535393 units were collected, accounting to an average fulfillment of 100%. There were 1275 store visits, accumulating a travel distance of 437013.62 kilometers (table 34).

Table 34: Statistics for the Solution.

Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
2118	2118	1275	437013.62	535393	1	153.27	535393

The requesting store that accounted for the highest value of the fitness function was the store loja33 which requested 18 products, amounting to 4161 units. The vehicle employed by loja33 visited 53distinct stores and collected 4161 units, traveling 1211.02 kilometers.

Table 35: Statistics for the request with the highest fitness value in the solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja33	18	loja231, loja49, loja163	3	1211.02	4161	3.44	4161

4.2.3 Comparison of Solutions

As is observable from the comparison table 36, the algorithm that generated the solution with the best fitness was the genetic algorithm with elitism starting with a population consisting of solutions generated by the Dynamic2 algorithm. Both the genetic algorithm and genetic algorithm with elitism were the only algorithms in 36 that managed to improve the baseline solution dynamic2, which has a fitness of 151.75. The instances that found a better solution than the baseline one, used a Dynamic2 solution population as

initial population. The genetic algorithm found a solution with fitness greater than the baseline solution by 0.78%, and the genetic algorithm with elitism's best solution is approximately 1.00% even higher. As mentioned before, considering the significantly higher number of stores and requests compared to the other scenarios, which consequently results in a larger solution space, the complexity of the search increases. Due to time constraints and the computational cost of this scenario, the tested algorithm instances have fewer iterations to search for solutions than the other scenarios. These factors make finding promising solutions in the algorithms' run time more challenging. So, the reason the genetic algorithms' instances with initial population consisting of dynamic2 solutions are the only instances finding better solutions than the baseline dynamic2 solution, might be because starting with a population of solutions that have fitness values close to the baseline solution and recombining and mutating these solutions can lead to finding a better solution than the baseline in fewer generations than starting, for example, with a random population.

In addition to testing these algorithms using a Dynamic2 solutions as initial solutions, the algorithms were also tested using a feasible solution generated through random mechanisms. All of these instances found solutions with fitness values greater than their initial solution or population. The genetic algorithm with elitism found a better solution than the genetic algorithm, with its solution having a fitness 23.77% greater than the average of its initial random population, 48.16. The genetic algorithm's best solution has a fitness 23.28% greater than the random initial population. Regarding the single solution algorithms, the Tabu Search algorithm found the best of all the single solution algorithms developed in this scenario, with its fitness value being 10.43% greater than the initial random solution, which has a fitness value of 48.53. Its then followed by the simulated annealing algorithm, which found a solution with a fitness value 10.29% greater than the random initial solution. Finally, the Tabu Search with Intensification and Diversification, tabu_search2 in the comparison table 36, found a solution with a fitness value 6.58% greater than the random initial solution.

Table 36: Table with the instances of the algorithms for Scenario 2

Algorithm	Instance	Fitness	Solutions Generated	Parameters	Iterations	Initial Solution/Population
dynamic2	1	151.75	1	Rank: 1	1	None
random	1	48.53	1	None	1	None
simulated_annealing	1	151.75	654306	Initial Temp: 500, Cooling Rate: 0.99, Stop Temp: 0.001, Temp Iter: 500	1306	dynamic2
simulated_annealing	2	53.51	654306	Initial Temp: 500, Cooling Rate: 0.99, Stop Temp: 0.001, Temp Iter: 500	1306	random
simulated_annealing_d_2	1	151.75	654306	Initial Temp: 500, Cooling Rate: 0.99, Stop Temp: 0.001, Temp Iter: 500, max_iter_without_new_sol: 1000	1306	dynamic2
tabu_search	1	151.75	654630	Tabu List: 50, Max Iter: 10, subset_size: 500	1309	dynamic2
tabu_search	2	53.59	654628	Tabu List: 50, Max Iter: 10, subset_size: 500	1309	random
tabu_search2	1	151.75	654589	Tabu List: 50, Elite: 20, Max Iter: 10, subset_size: 500	1309	dynamic2
tabu_search2	2	51.72	654587	Tabu List: 50, Elite: 20, Max Iter: 10, subset_size: 500	1309	random
genetic_algorithm	1	59.33	654333	Pop Size: 1000, Crossover_p: 0.6, Mutation_p: 0.4, Selection: tourney_sel(size=5)	654	random: 1000
genetic_algorithm_elitism	1	59.61	655300	Pop Size: 1000, Crossover_p: 0.6, Mutation_p: 0.4, Selection: tourney_sel(size=5), Elitism: 100	520	random: 1000
genetic_algorithm	2	152.93	654319	Pop Size: 1000, Crossover_p: 0.6, Mutation_p: 0.4, Selection: tourney_sel(size=5)	654	d2: 1000
genetic_algorithm_elitism	2	153.27	655614	Pop Size: 1000, Crossover_p: 0.6, Mutation_p: 0.4, Selection: tourney_sel(size=5), Elitism: 100	519	d2: 1000

4.3 Scenario 3

4.3.1 Characteristics of the Problem

Products and Quantity Needed for each Requesting Store

In the network of stores, 4 stores are in need of at least one product. There is a necessity for 245 types of products, all of them requested by more than one of the 4 requesting stores. The accumulated requested quantities range from 32 to 4276 units of a product (figure 90).

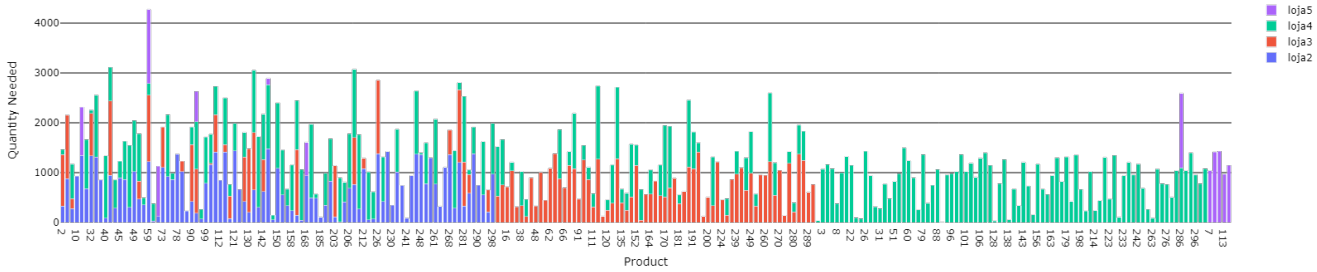


Figure 90: Bar chart representing the needs of the requesting stores.

Stock of the Requested Products

In the network of stores, there are 250 types of products in stock, and the accumulated quantities of a product in at least one store vary from 506 to 8192 units. It is important to mention that the distribution of products and their quantities is not uniform across all stores in the network (figure 91).

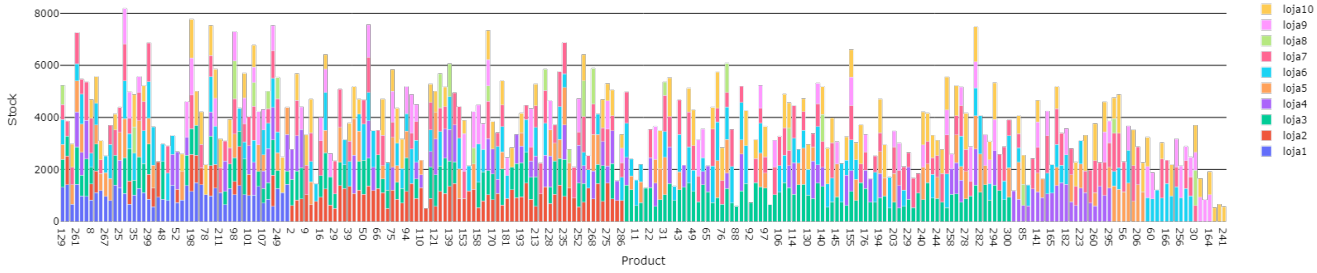


Figure 91: Bar chart representing the stock of the requested products.

Distances between the Stores

The distance between two stores is distributed randomly from a minimum of 10 kilometers to a maximum of 100 kilometers (figure 92).

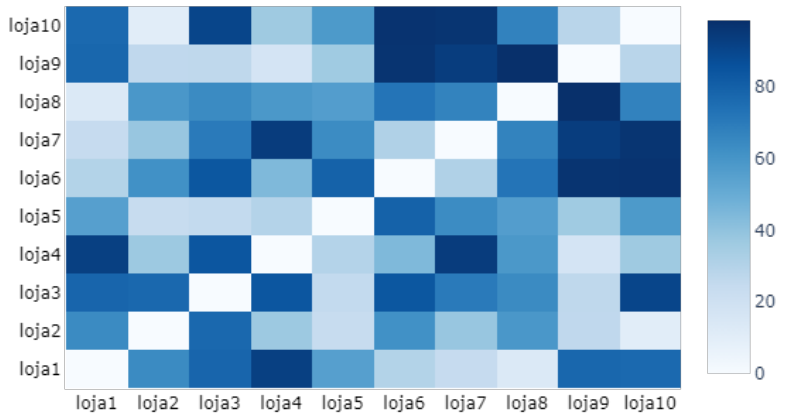


Figure 92: Heatmap representing the distances between the stores in the network.

4.3.2 Solutions and Results:

Dynamic2

Statistics of the Solution

The Dynamic2 algorithm's solution has a fitness of 804.03. It was requested for 385 products across all requesting stores, of which 337 were completely fulfilled. In total, 2641121 units were collected, accounting to an average fulfillment of 92%. There were 30 store visits, accumulating a travel distance of 1242.42 kilometers (table 37).

Table 37: Statistics of the Solution.

Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	337	30	1242.42	264121	0.92	804.03	288690

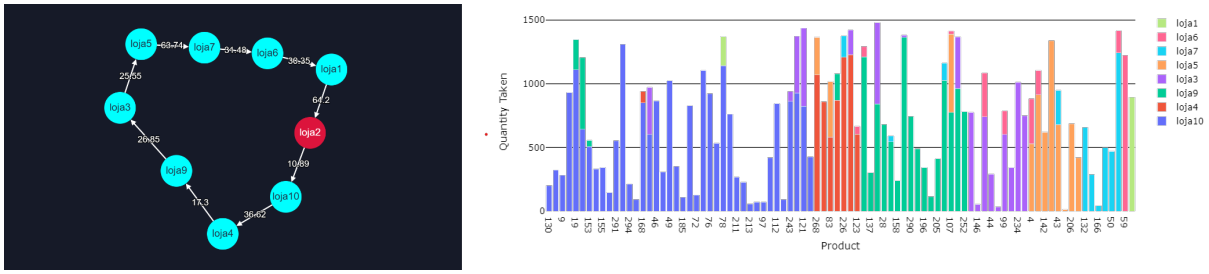
The requesting store that accounted to the highest value of the fitness function was the store loja4, which requested 186 products, amounting to 142006 units. The vehicle employed by loja4, visited 8 distinct

stores and collected 128384 units, traveling 355.64 kilometers. The second store with the highest fitness value is the store loja3, which requested 96 types of products, amounting to 71977 unities. The vehicle employed by this stores, visited 7 distinct stores and collected 64190 units, traveling 313.13 kilometers. The store with the lowest fitness value is the store loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store, visited 12 distinct stores and collected 10102 units, traveling 266.67 kilometers (table 38).

Table 38: Statistics for each request for the Dynamic2 algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja2	91	loja10, loja4, loja9, loja3, loja5, loja7, loja6, loja1	8	306.98	61445	200.16	62360
loja3	96	loja5, loja2, loja10, loja4, loja9, loja6, loja7	7	313.13	64190	204.99	71977
loja4	186	loja9, loja3, loja5, loja2, loja10, loja7, loja6, loja1	8	355.64	128384	360.99	142006
loja5	12	loja3, loja9, loja4, loja10, loja2, loja7, loja6	7	266.67	10102	37.88	12347

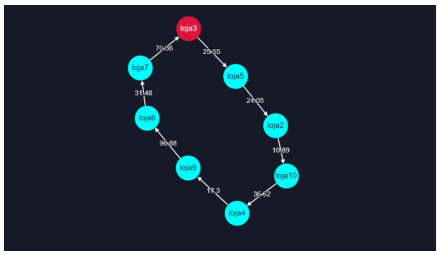
Representation of the Solution



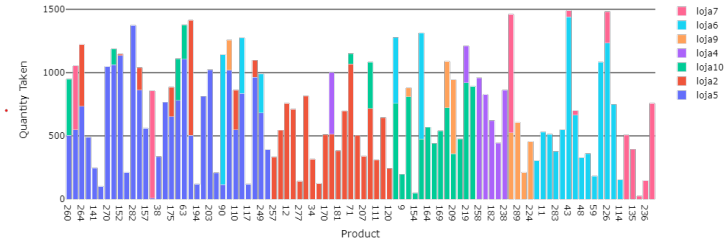
(a) Route for the store loja2's request.

(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 93: Requesting store's loja2 part of the solution.

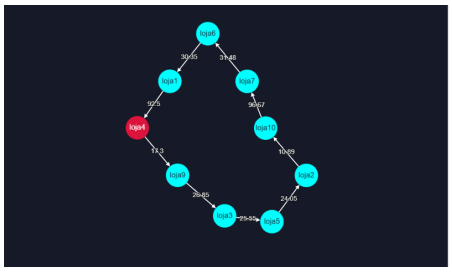


(a) Route for the store loja3's request.

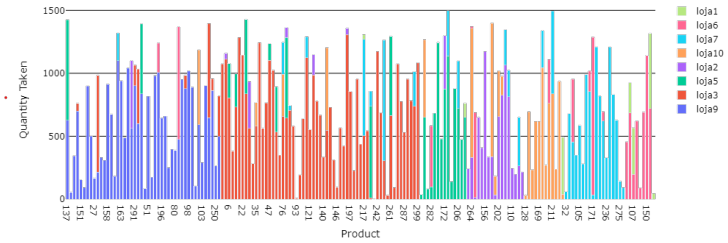


(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 94: Requesting store's loja3 part of the solution.

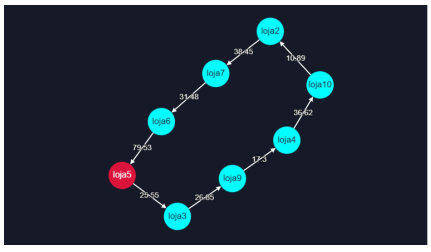


(a) Route for the store loja4's request.

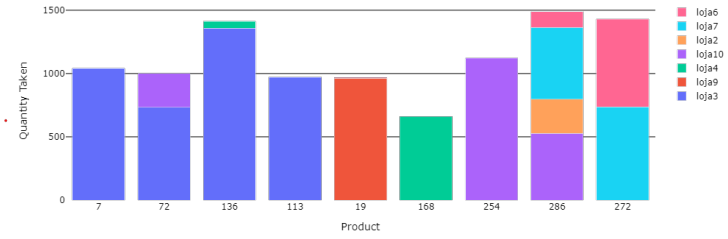


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 95: Requesting store's loja4 part of the solution.



(a) Route for the store loja5's request.



(b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 96: Requesting store's loja5 part of the solution.

Simulated Annealing

Statistics of the Solution

In this fitness graph 97, it is represented the fitness through the iterations of the Simulated Annealing algorithm. The fitness fluctuates approximately between 52 to 1088.50, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 200 and 40.

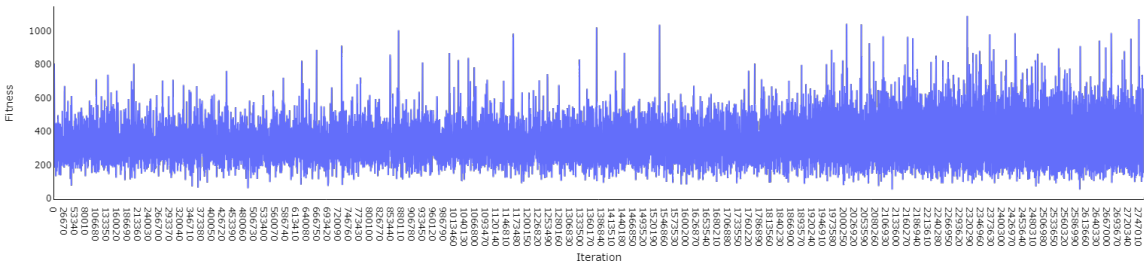


Figure 97: Fitness of the solutions found by the Simulated Annealing algorithm during its iterations.

The Simulated Annealing algorithm's solution has a fitness of 1088.50. It was requested for 385 products across all requesting stores, of which 111 were completely fulfilled. In total, 102865 units were collected, accounting to an average fulfillment of 41%. There were 12 store visits, accumulating a travel distance of 550.32 kilometers (table 39).

Table 39: Statistics of the Solution.

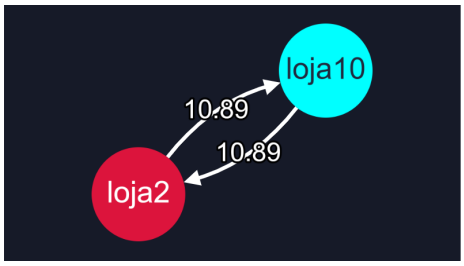
Nº of Requested Items	Nº of Fulfilled Items	Number of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	111	12	550.32	102865	0.41	1088.50	288690

The requesting store that accounted for the highest value of the fitness function was the store loja2 which requested 91 products, amounting to 62360 units. The vehicle employed by loja2 visited 1 store and collected 14238 units, traveling 21.78 kilometers. The second store with the highest fitness value is the store loja4, which requested 186 types of products, amounting to 142006 units. The vehicle employed by this store visited 6 distinct stores and collected 83044 units, traveling 208.10 kilometers. The store with the lowest fitness value is loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store visited 3 distinct stores and collected 2603 units, traveling 180.54 kilometers (table 40).

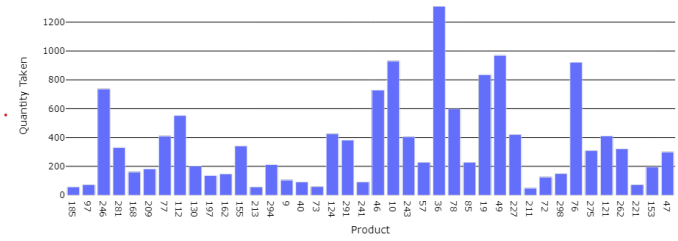
Table 40: Statistics for each request of the Simulated Annealing algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja2	91	loja10	1	21.78	14238	653.72	62360
loja3	96	loja4, loja5	2	139.90	2980	21.30	71977
loja4	186	loja9, loja3, loja5, loja2, loja7, loja6	6	208.10	83044	399.06	142006
loja5	12	loja1, loja9, loja4	3	180.54	2603	14.42	12347

Representation of the Solution

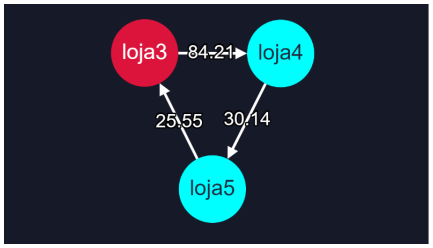


(a) Route for the store loja2's request.

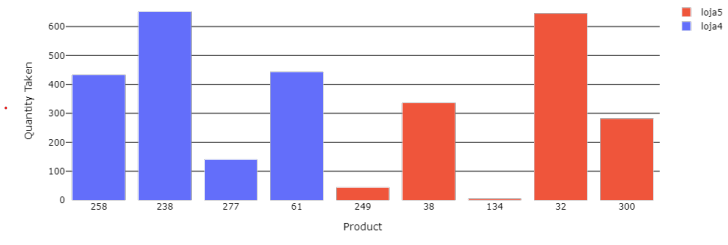


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 98: Requesting store's loja2 part of the solution.

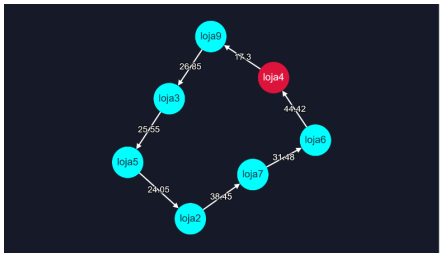


(a) Route for the store loja3's request.

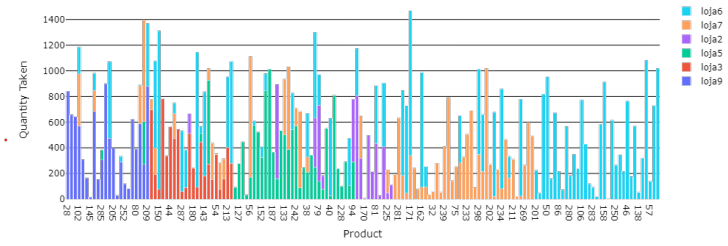


(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 99: Requesting store's loja3 part of the solution.

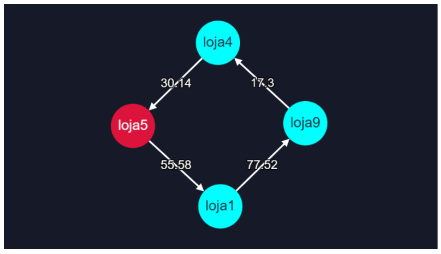


(a) Route for the store loja4's request.

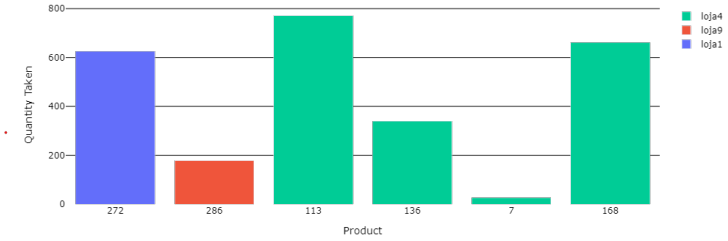


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 100: Requesting store's loja4 part of the solution.



(a) Route for the store loja5's request.



(b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 101: Requesting store's loja5 part of the solution.

Simulated Annealing using Dynamic2 solutions

Statistics of the Solution

In this fitness graph 102, it is evident that the number of peaks is higher compared to the Simulated Annealing instance in subsection 4.3.2. This can be explained by the fact that the algorithm in question jumps in the solution search space to solutions generated by the Dynamic2 algorithm instead of jumping to random points in the search space. As mentioned before, these solutions generated by the Dynamic2 algorithms are generally better than solutions generated through random mechanisms. When these jumps happen, they can be further improved.

The fitness fluctuates approximately between 342 to 832.41, having values with an improved fitness over

the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 450 and 620.

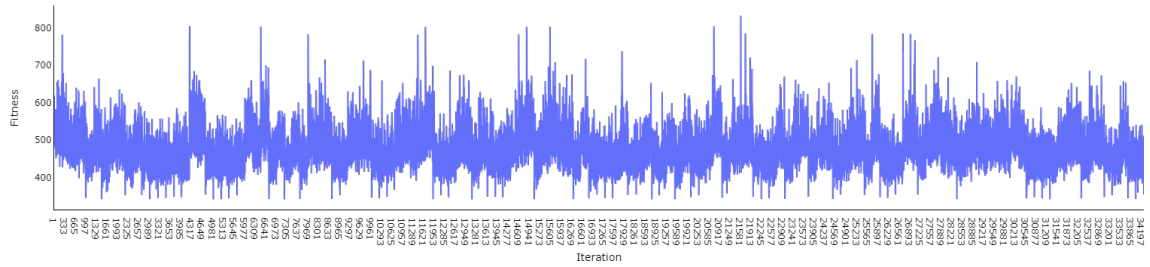


Figure 102: Fitness of the solutions found by the Simulated Annealing algorithm using Dynamic2 solutions during its iterations.

The Simulated Annealing algorithm using jumps to Dynamic2 solutions, best found solution has a fitness of 832.41. It was requested for 385 products across all requesting stores, of which 337 were completely fulfilled. In total, 264121 units were collected, accounting to an average fulfillment of 92%. There were 30 store visits, accumulating a travel distance of 1518.17 kilometers (table 41).

Table 41: Statistics of the Solution.

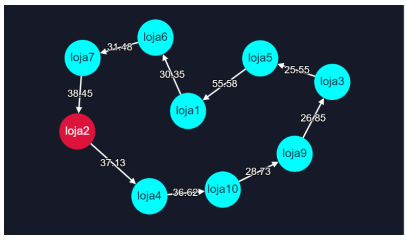
Number of Requested Items	Number of Fulfilled Items	Number of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	337	30	1518.17	264121	0.918	832.41	288690

The requesting store that accounted for the highest value of the fitness function was the store Loja4 which requested 186 products, amounting to 142006 units. The vehicle employed by Loja4 visited 8 distinct stores and collected 128384 units, traveling 260.05 kilometers. The second store with the highest fitness value is the store Loja2, which requested 91 types of products, amounting to 62360 units. The vehicle employed by this store visited 8 distinct stores and collected 61445 units, traveling 310.74 kilometers. The store with the lowest fitness value is loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store visited 5 distinct stores and collected 10102 units, traveling 389.41 kilometers (table 39).

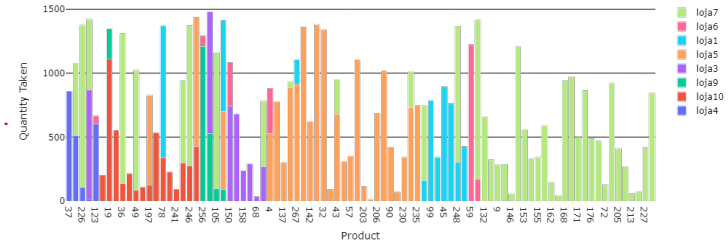
Table 42: Statistics for each request for the Simulated Annealing algorithm using jumps to Dynamic2 solutions' best found solution.

Requesting Store	Number of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Requested	Quantity Taken	Fitness
loja2	91	loja4, loja10, loja9, loja3, loja5, loja1, loja6, loja7	8	310.74	62360	61445	197.74
loja3	96	loja7, loja4, loja5, loja9, loja1, loja8, loja6, loja2, loja10	9	557.97	71977	64190	115.04
loja4	186	loja6, loja1, loja7, loja2, loja10, loja9, loja3, loja5	8	260.05	142006	128384	493.69
loja5	12	loja2, loja7, loja10, loja6, loja9	5	389.41	12347	10102	25.94

Representation of the Solution

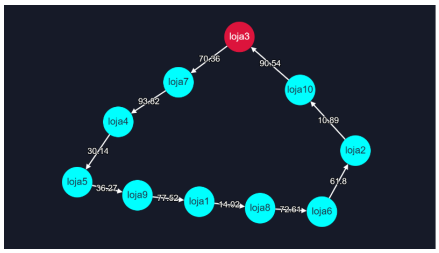


(a) Route for the store loja2's request.

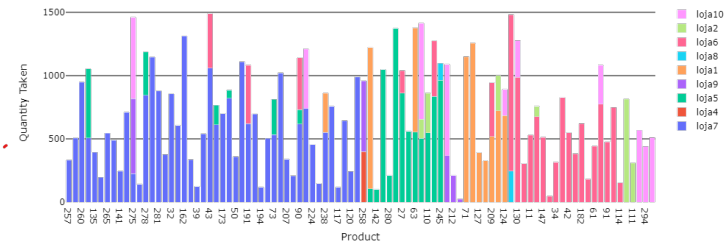


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 103: Requesting store's loja2 part of the solution.

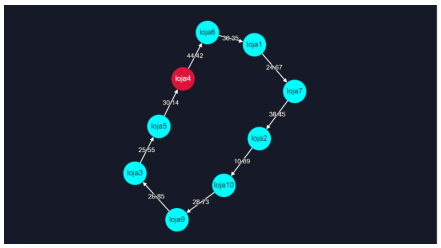


(a) Route for the store loja3's request.

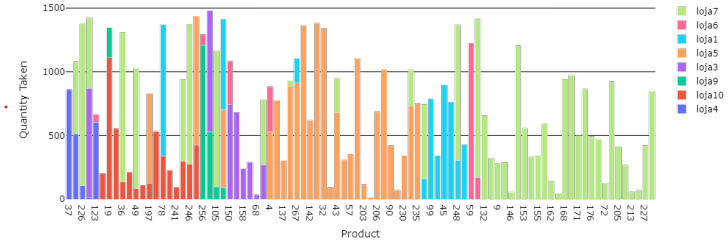


(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 104: Requesting store's loja3 part of the solution.

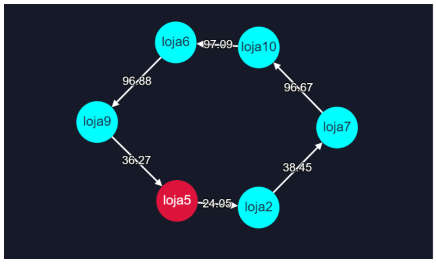


(a) Route for the store loja4's request.

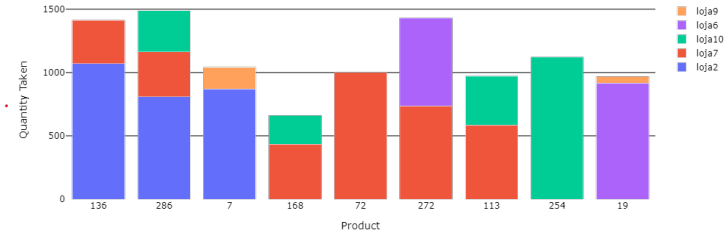


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 105: Requesting store's loja4 part of the solution.



(a) Route for the store loja5's request.



(b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 106: Requesting store's loja5 part of the solution.

Tabu Search

Statistics of the Solution

In this fitness graph 107, it is represented the fitness through the iterations of the Simulated Annealing algorithm. The fitness fluctuates approximately between 54 to 1067.93, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 250 and 500.

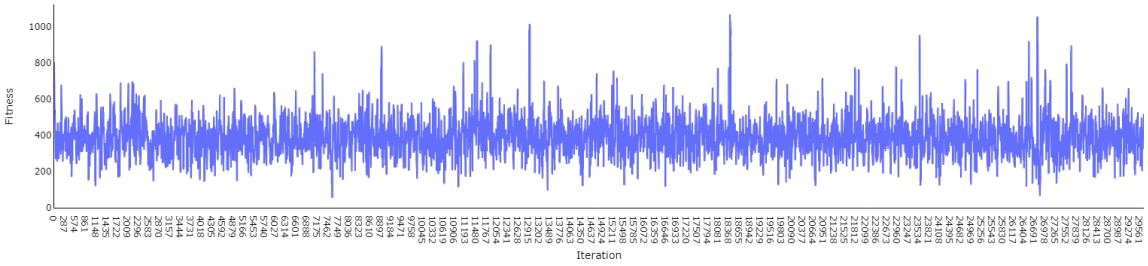


Figure 107: Fitness of the solutions found by the Tabu Search algorithm during its iterations.

The Tabu Search algorithm's solution has a fitness of 1067.93. It was requested for 385 products across all requesting stores, of which 110 were completely fulfilled. In total, 113123 units were collected, accounting to an average fulfillment of 46%. There were 16 store visits, accumulating a travel distance of 678.43 kilometers (table 43).

Table 43: Statistics of the Solution.

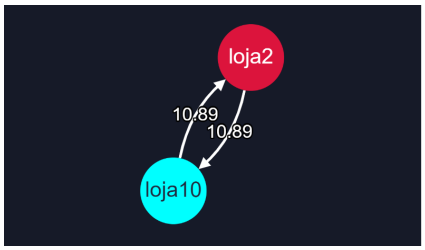
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	110	16	678.43	113123	0.46	1067.93	288690

The requesting store that accounted for the highest value of the fitness function was the store loja2 which requested 91 products, amounting to 62360 units. The vehicle employed by loja2 visited 1 store and collected 14598 units, traveling 21.78 kilometers. The second store with the highest fitness value is the store loja4, which requested 186 types of products, amounting to 142006 units. The vehicle employed by this store visited 7 distinct stores and collected 65472 units, traveling 246.87 kilometers. The store with the lowest fitness value is loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store visited 2 distinct stores and collected 2905 units, traveling 139.90 kilometers (table 44).

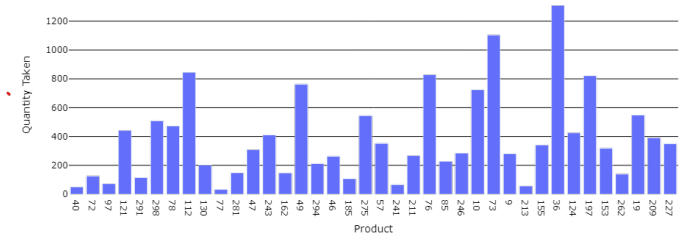
Table 44: Statistics for each request of the Tabu Search algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja2	91	loja10	1	21.78	14598	670.25	62360
loja3	96	loja9, loja4, loja6, loja8, loja2, loja5	6	269.88	30148	111.71	71977
loja4	186	loja6, loja1, loja7, loja2, loja9, loja3, loja5	7	246.87	65472	265.21	142006
loja5	12	loja3, loja4	2	139.90	2905	20.76	12347

Representation of the Solution

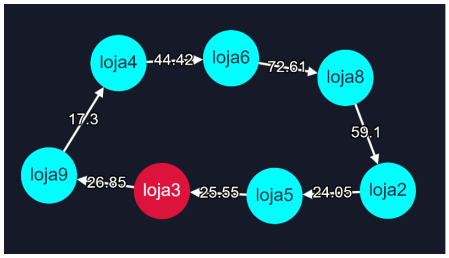


(a) Route for the store loja2's request.

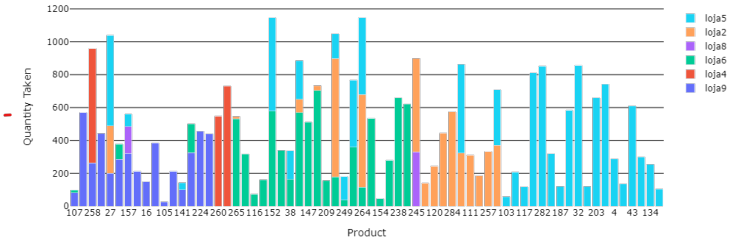


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 108: Requesting store's loja2 part of the solution.

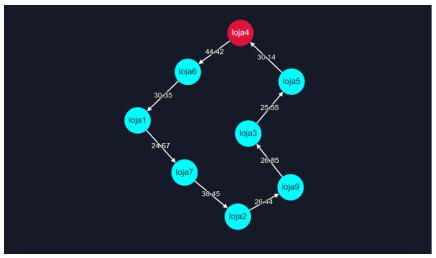


(a) Route for the store loja3's request.

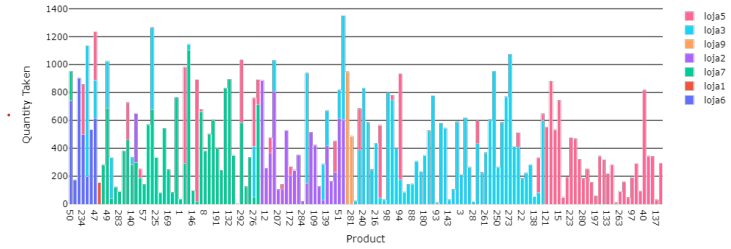


(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 109: Requesting store's loja3 part of the solution.

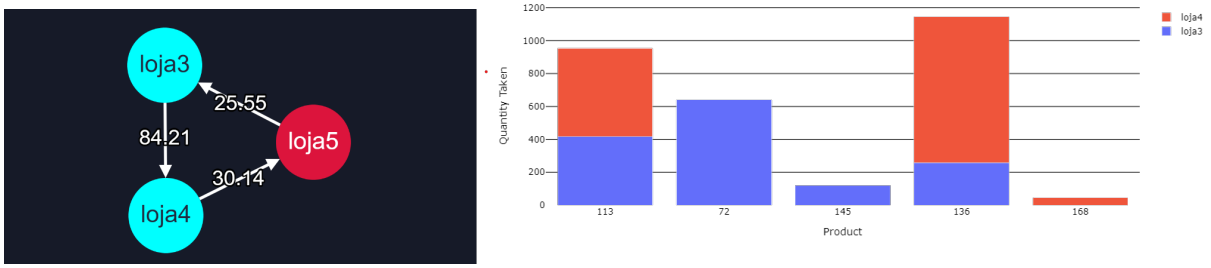


(a) Route for the store loja4's request.



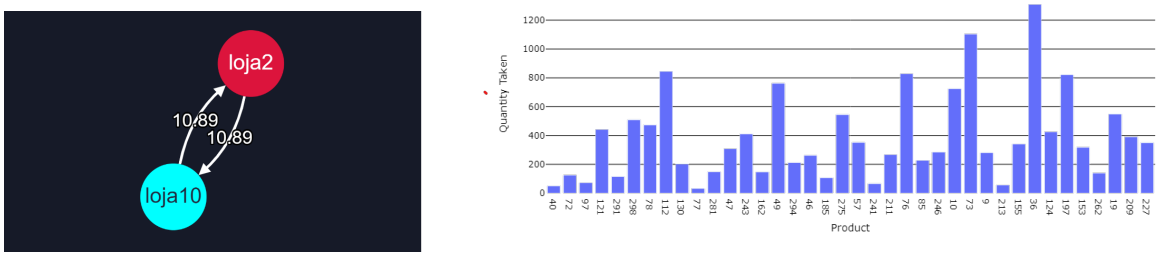
(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 110: Requesting store's loja4 part of the solution.



(a) Route for the store loja5's request. (b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 111: Requesting store's loja5 part of the solution.



(a) Route for the store loja2's request. (b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 112: Requesting store's loja2 part of the solution.

Tabu Search with Intensification and Diversification

Statistics of the Solution

In this fitness graph 113, it is evident that the number of peaks is higher compared to the Tabu Search instance in subsection 4.3.2. This might be due to the intensification mechanism by the way of elitism mentioned before. This consists in selecting one of the best found solutions in the medium-term memory as the current solution in order to intensify the search in promising areas of the solution search space.

The fitness fluctuates approximately between 95 to 1158.87, having values with an improved fitness over the initial solution generated by the Dynamic2 algorithm. The majority of iterations have fitness values varying approximately between 300 and 800.

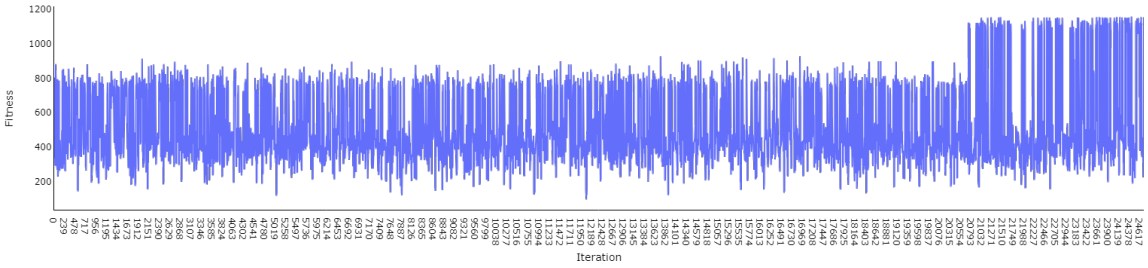


Figure 113: Fitness of the solutions found by the Tabu Search with Intensification and Diversification algorithm during its iterations.

The Tabu Search with Intensification and Diversification algorithm's solution has a fitness of 1158.87. It was requested for 385 products across all requesting stores, of which 127 were completely fulfilled. In total, 111497 units were collected, accounting to an average fulfillment of 43%. There were 16 store visits, accumulating a travel distance of 664.43 kilometers (table 45).

Table 45: Statistics of the Solution.

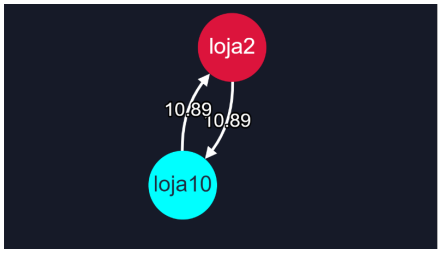
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	127	16	664.43	111497	0.43	1158.87	288690

The requesting store that accounted for the highest value of the fitness function was the store loja2 which requested 91 products, amounting to 62360 units. The vehicle employed by loja2 visited 1 store and collected 11980 units, traveling 21.78 kilometers. The second store with the highest fitness value is the store loja4, which requested 186 types of products, amounting to 142006 units. The vehicle employed by this store visited 1 store and collected 28370 units, traveling 88.84 kilometers. The store with the lowest fitness value is loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store visited 7 distinct stores and collected 7047 units, traveling 317.30 kilometers (table 46).

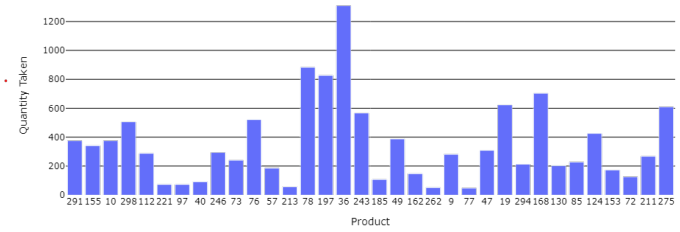
Table 46: Statistics for each request of the Tabu Search with Intensification and Diversification algorithm's solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled	Quantity Taken	Fitness	Quantity Requested
loja2	91	loja10	1	21.78	11890	545.91	62360
loja3	96	loja5, loja4, loja6, loja7, loja2, loja10, loja9	7	236.51	64190	271.41	71977
loja4	186	loja6	1	88.84	28370	319.34	142006
loja5	12	loja10, loja9, loja3, loja8, loja1, loja6, loja7	7	317.30	7047	22.21	12347

Representation of the Solution

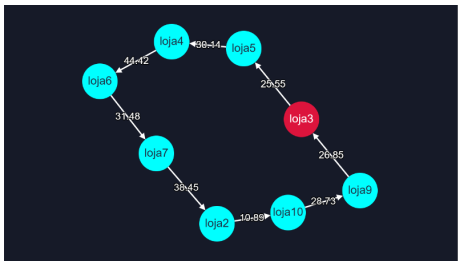


(a) Route for the store loja2's request.

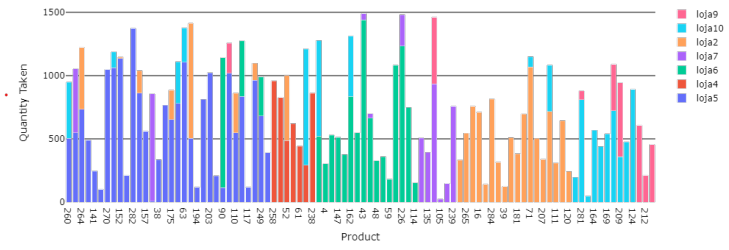


(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 114: Requesting store's loja2 part of the solution.

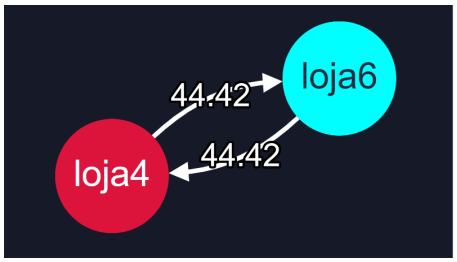


(a) Route for the store loja3's request.

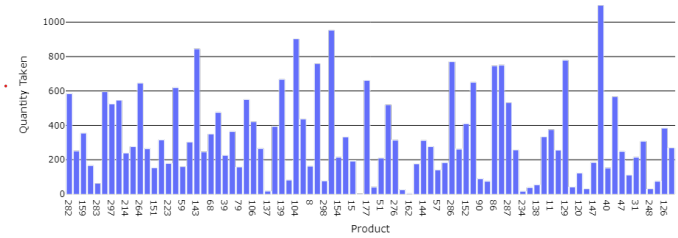


(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 115: Requesting store's loja3 part of the solution.

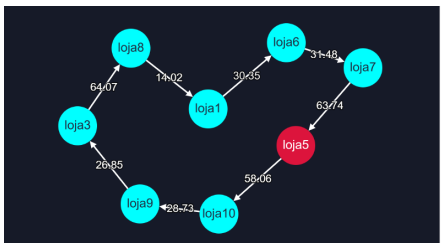


(a) Route for the store loja4's request.

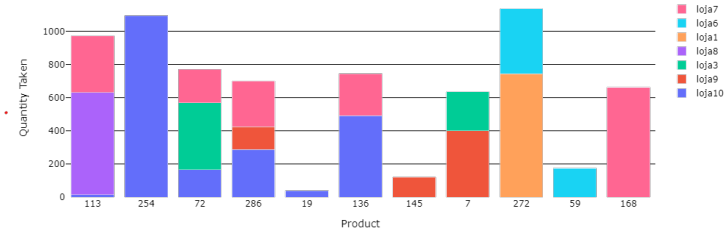


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 116: Requesting store's loja4 part of the solution.



(a) Route for the store loja5's request.



(b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 117: Requesting store's loja5 part of the solution.

Genetic Algorithm

In this instance, an initial population of 1000 solutions obtained through random procedures is used. Every solution in this initial population is feasible. The average fitness of the mentioned population is 307.91.

Statistics of the Solution

In this fitness graph 118, it is represented the average fitness of the population through the generations of the Genetic algorithm. The fitness fluctuates approximately between 307.91 to 1550.13, having values with an improved fitness over the initial population. In the first, approximately, 20 generations, the average of the fitness values of the population increased consistently from around 307.91 to 1536, slightly fluctuating around the latter until the end, reaching a maximum value of 1550.13.

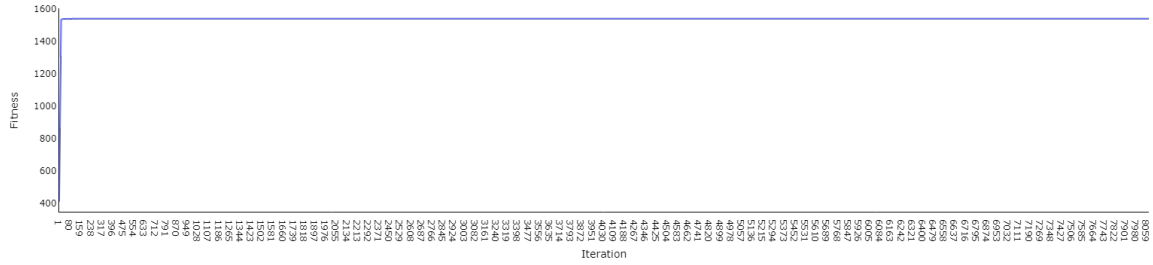


Figure 118: Fitness of the solutions found by the Genetic algorithm during its iterations.

The Genetic algorithm's solution has a fitness of 1550.13. It was requested for 385 products across all requesting stores, of which 50 were completely fulfilled. In total, 50824 units were collected, accounting to an average fulfillment of 22%. There were 4 store visits, accumulating a travel distance of 158.58 kilometers (table 47).

Table 47: Statistics of the Solution.

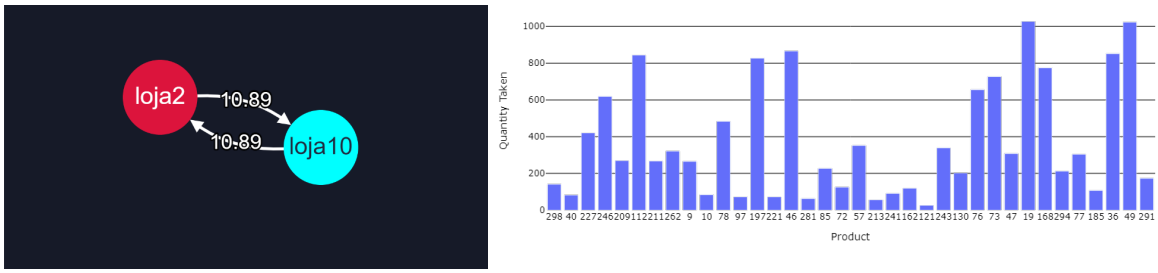
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	50	4	158.58	50824	0.22	1550.13	288690

The requesting store that accounted for the highest value of the fitness function was the store loja4 which requested 186 products, amounting to 142006 units. The vehicle employed by loja4 visited 1 store and collected 21712 units, traveling 34.60 kilometers. The second store with the highest fitness value is the store loja2, which requested 91 types of products, amounting to 62360 units. The vehicle employed by this store visited 1 store and collected 13396 units, traveling 21.78 kilometers. The store with the lowest fitness value is loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store visited 1 store and collected 2151 units, traveling 51.10 kilometers (table 48).

Table 48: Statistics for each requesting store of the Genetic algorithm's best found solution.

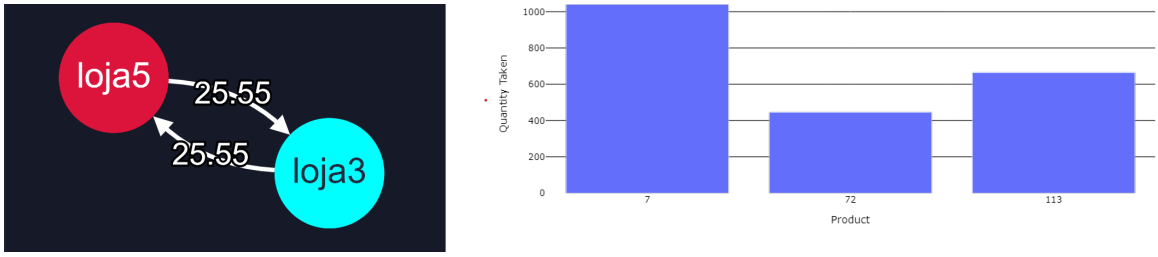
Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness of the Requesting Store	Quantity Requested
loja2	91	loja10	1	21.78	13396	615.06	62360
loja5	12	loja3	1	51.10	2151	42.09	12347
loja4	186	loja9	1	34.60	21712	627.51	142006
loja3	96	loja5	1	51.10	13565	265.46	71977

Representation of the Solution



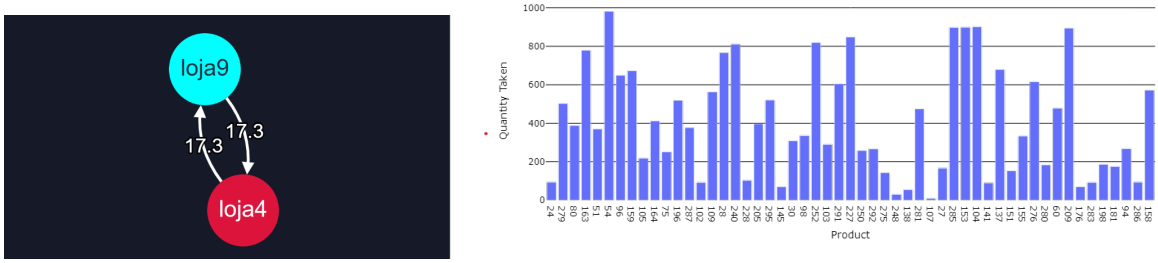
(a) Route for the store loja2's request. (b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 119: Requesting store's loja2 part of the solution.



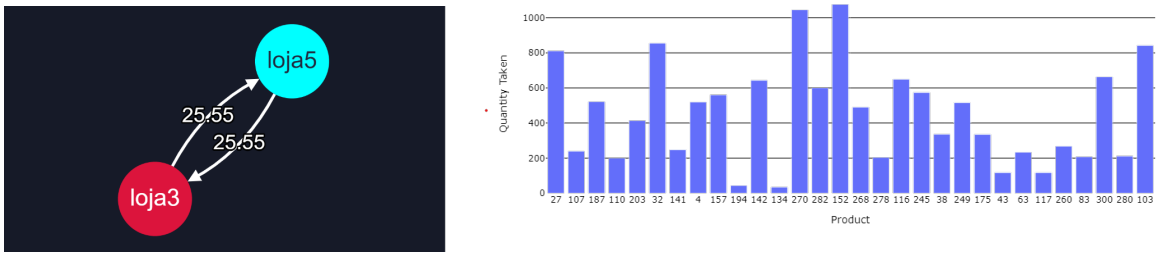
(a) Route for the store loja5's request. (b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 120: Requesting store's loja5 part of the solution.



(a) Route for the store loja4's request. (b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 121: Requesting store's loja4 part of the solution.



(a) Route for the store loja3's request.

(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 122: Requesting store's loja3 part of the solution.

Genetic Algorithm with Elitism

The initial population used in this instance of the algorithm is the same as the one used in the genetic algorithm instance 4.3.2.

Statistics of the Solution

In this fitness graph 123, it is represented the average fitness of the population through the iterations of the Genetic algorithm with Elitism. The fitness fluctuates approximately between 307.91 to 1542.51, having values with an improved fitness over the initial population. In the first, approximately, 20 generations, the average of the fitness values of the population increased consistently from around 307.91 to 1539, slightly fluctuating around the latter until the end, reaching a maximum value of 1542.51.

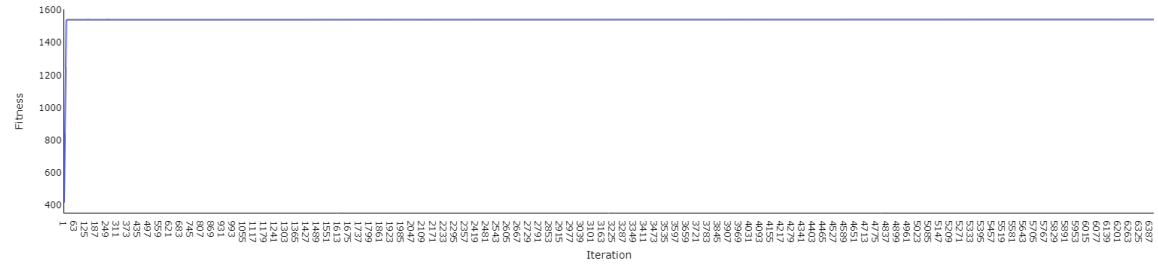


Figure 123: Fitness of the solutions found by the Genetic algorithm with Elitism during its iterations.

The Genetic algorithm with Elitism's solution has a fitness of 1542.51. It was requested for 385 products across all requesting stores, of which 54 were completely fulfilled. In total, 58073 units were collected,

accounting to an average fulfillment of 24%. There were 10 store visits, accumulating a travel distance of 380.14 kilometers (table 49).

Table 49: Statistics of the solution.

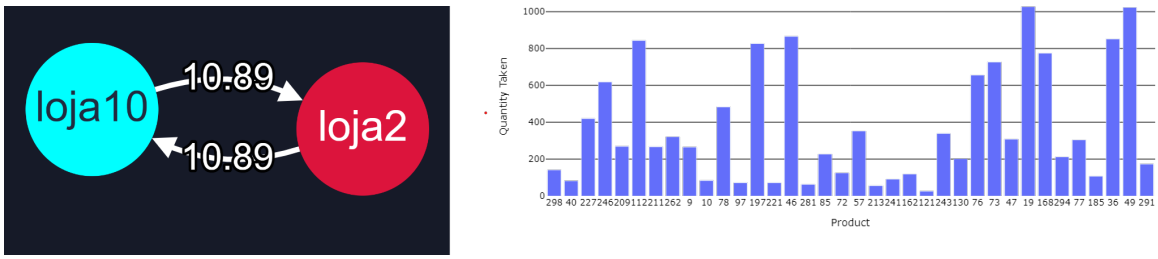
Nº of Requested Items	Nº of Fulfilled Items	Nº of Visits to Stores	Distance Travelled (Km)	Total Quantity Taken	Average Fulfillment	Fitness	Quantity Requested
385	54	10	380.14	58073	0.24	1542.51	288690

The requesting store that accounted for the highest value of the fitness function was the store loja4 which requested 186 products, amounting to 21712 units. The vehicle employed by loja4 visited 1 store and collected 21712 units, traveling 34.60 kilometers. The second store with the highest fitness value is the store loja2, which requested 91 types of products, amounting to 62360 units. The vehicle employed by this store visited 1 store and collected 13396 units, traveling 21.78 kilometers. The store with the lowest fitness value is loja5, which requested 12 types of products, amounting to 12347 units. The vehicle employed by this store visited 7 distinct stores and collected 9400 units, traveling 34.48 kilometers (table 50).

Table 50: Statistics for each requesting store of the Genetic algorithm with Elitism's best found solution.

Requesting Store	Nº of Products Requested	Visited Stores	Nº of Visited Stores	Distance Travelled (Km)	Quantity Taken	Fitness	Quantity Requested
loja2	91	loja10	1	21.78	13396	615.06	62360
loja4	186	loja9	1	34.60	21712	627.51	142006
loja3	96	loja5	1	51.10	13565	265.46	71977
loja5	12	loja9, loja10, loja4, loja6, loja7, loja1, loja8	7	272.66	9400	34.48	12347

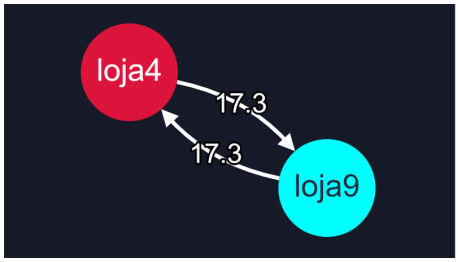
Representation of the Solution



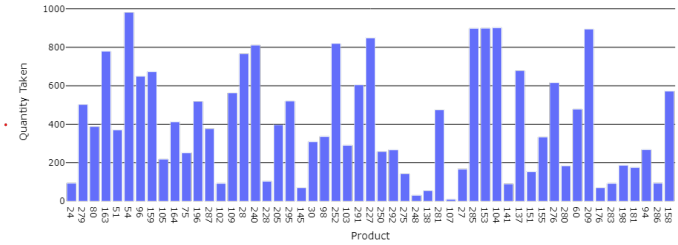
(a) Route for the store loja2's request.

(b) Products and respective quantities taken at each visited store for store loja2's request.

Figure 124: Requesting store's loja2 part of the solution.

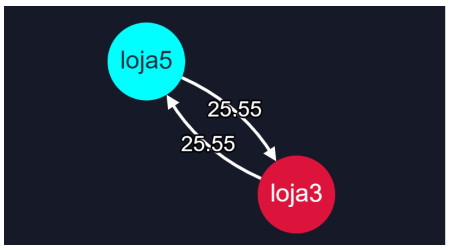


(a) Route for the store loja4's request.

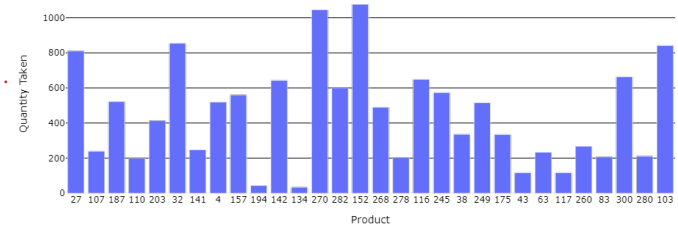


(b) Products and respective quantities taken at each visited store for store loja4's request.

Figure 125: Requesting store's loja4 part of the solution.

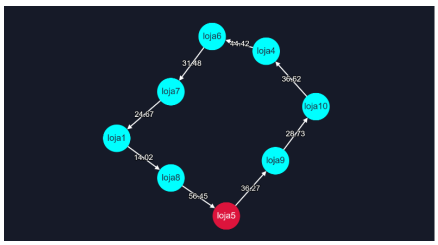


(a) Route for the store loja3's request.

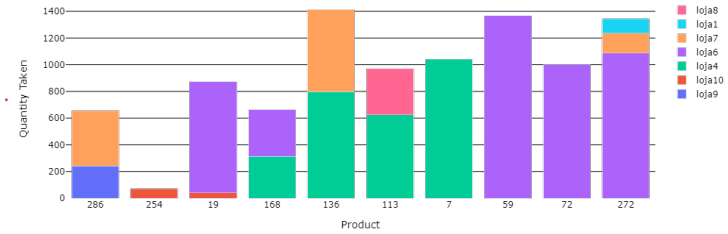


(b) Products and respective quantities taken at each visited store for store loja3's request.

Figure 126: Requesting store's loja3 part of the solution.



(a) Route for the store loja5's request.



(b) Products and respective quantities taken at each visited store for store loja5's request.

Figure 127: Requesting store's loja5 part of the solution.

4.3.3 Comparison of Solutions

As is observable from the comparison table 51, the algorithm that generated the solution with the best fitness was the genetic algorithm, followed by the genetic algorithm with elitism. Both of these algorithms started with an initial population consisting of feasible solutions generated by random mechanisms. The average fitness of the mentioned population is 307.91. In this scenario, since there are only 4 requests, the number of possible Dynamic2 solutions is more limited and as such, the genetic algorithms were not tested with an initial population consisting of Dynamic2 solutions. The genetic algorithm's best found solution has a fitness value 402.39% than its initial population average fitness and 92.07% greater than the baseline solution, dynamic2. The genetic algorithm with Elitism's best found solution has a fitness value 400.56% than its initial population average fitness and 91.61% greater than the baseline solution, dynamic2.

Regarding the single solution based algorithms, the baseline dynamic2 solution was used as the initial solution. Every instance of these algorithms found solutions with better fitness values than their initial solution. Out of all the single solution instances tested, the one performed by the Tabu Search algorithm with Intensification and Diversification, or tabu_search2 in the comparison table 51, found the best solution. This solution's fitness value is 44.55% greater than the baseline solution, with the intensification mechanism proving useful when comparing to the tabu_search algorithm's instance in the comparison table, with its best solution having a fitness value 32.79% greater than the baseline solution. The Simulated Annealing algorithm found the second best solution out of all the single solution based algorithms, with a fitness value 35.51% greater than the baseline solution. The Simulated Annealing algorithm proved to find a better solution than the Simulated Annealing with jumps to Dynamic2 solutions, or simulated_annealing_d_2 in the comparison table 51, that found its best solution with a fitness 3.54% greater than the baseline solution. As mentioned before, the number of possible Dynamic2 solutions is more limited, which affects the performance of the simulated_annealing_d_2 algorithm that uses these solutions to jump in the solution search space. Due to this, the instance of simulated_annealing_d_2 found in the comparison table 51 had its stopping criteria fulfilled relatively early, resulting in fewer iterations and number of solutions generated than the other algorithm's instances. Still, it managed to find improvements over the initial solution in its run time.

Table 51: Table with the instances of the algorithms for Scenario 3

Algorithm	Instance	Fitness	Solutions Generated	Parameters	Iterations	Initial Solution/Population
dynamic2	1	804.03	1	Rank: 1	1	None
simulated_annealing	1	1088.50	8071611	Initial Temperature: 1000, Cooling Rate: 0.999, Stopping Temperature: 0.0001, Temperature Iterations: 500	16111	dynamic2
simulated_annealing_d_2	1	832.41	69639	Initial Temperature: 1000, Cooling Rate: 0.999, Stopping Temperature: 0.0001, Temperature Iterations: 500, max_iter_without_new_sol: 1000	139	dynamic2
tabu_search	1	1067.93	8072021	Tabu List size: 50, Maximum Number of Iterations without Improvement: 10, subset_size: 500	29657	dynamic2
tabu_search2	1	1158.87	8071611	Tabu List size: 50, Elite List size: 20, Maximum Number of Iterations without Improvement: 10, subset_size: 500	24700	dynamic2
genetic_algorithm	1	1550.13	8072260	Population Size: 1000, Generation Limit: 8073, Crossover Rate: 0.6, Mutation Rate: 0.4, Selection: tournament_selection(size=5)	8073	random: 1000
genetic_algorithm_elitism	1	1542.51	8072810	Population Size: 1000, Generation Limit: 6407, Crossover Rate: 0.6, Mutation Rate: 0.4, Selection: tournament_selection(size=5), Elitism Size: 100	6407	random: 1000

Chapter 5

Conclusions and Future Work

In this day and age, the success of a business relies on its capacity to successfully integrate into a complex network of entities that are linked by material and information flows, where inventory management and routing are some of the main concerns. These flows are characterized by decision-making processes that adapt to the network's environment and its entities. As a result, a decision making system is required to provide solutions that represent the optimal way the network and its entities can collect and provide inventory in order to reduce costs and maximize profit.

In the context of this dissertation, the stores of the same retailer group are connected in a network and are monitored by a central system. The problem occurs when there are stock disruptions in the network and outside entities are unable to answer the stores' supply requests, forcing these stores to become the sole entities responsible of requesting and collecting products from each other. The mentioned decision making system identifies the stores in the network that are in need of specific products, along with the quantities required, as well as the stores capable of supplying required goods. The requesting stores send requests, that contain the specific products and respective quantities these stores need, to the system, which in turn taking into account the necessities and ensuring that there are no stock-outs for every store in the network, provides a solution. This solution specifies, for each requesting store, which stores must be visited and in what sequence, as well as which type of products and in what quantity to take from each visited store, in an attempt to maximize the collection of production per travel distance for the network. Once the solution is given, the requesting stores employ a vehicle to pick up the products in the specified places following the provided route and respecting the system's decisions. The problem at hand is modeled as an Inventory Routing Problem, given that it involves both inventory management and routing decisions. The problem in discussion is an optimization problem.

In order to tackle this optimization problem, the main goal of this optimization, which is maximizing the

stores' on-shelf availability without creating stock-outs for any of the networks' entities, was divided into four specific objectives. Objective 1, was achieved by performing a study of the inventory routing problem and optimization algorithms, conducting a requirement elicitation with domain experts, resulting in the identification of the key aspects and characteristics of the problem, and in the definition of a framework for representing the solution and entities of the problem. From there, for the objective 2, a mathematical formulation of the optimization problem, was developed, where the objective function and constraints of the problem were mathematically defined. The objective or fitness function represents the maximization of the collection of products per travel distance, without causing stock-outs at any supplier, for the entire network. After this, the optimization algorithms and another algorithm, by the name of Dynamic2, that generates baseline solutions, were developed.

Regarding the 3 and 4 objectives, in order to test and compare the developed optimization algorithms, three different sized scenarios were generated. Each of these scenarios has a different amount of data associated with them, whether it be in the number of stores, types of products or number of requests, as can be seen in the scenario table 2. Generally speaking, the genetic algorithms both with and without elitism found the solutions with the highest fitness values out of all the optimization algorithms developed. Instances of the genetic algorithms with initial populations consisting of feasible solutions generated through random mechanisms and solutions generated by the Dynamic2 algorithm were tested. Excluding Scenario 2, the genetic algorithms with random initial population found their best solutions with higher fitness values than the ones with Dynamic2 initial populations. This might be because having an initial population compromised of just random solutions can allow the algorithm to search more of the solution search space, due to the solution diversity this type of population can present. In spite of the Dynamic2 solutions being typically superior in terms of fitness value than the ones generated through random mechanisms, it makes the exploration of the solution search space more confined since the logic for generating these Dynamic2 solutions can be similar. In Scenario 2, as mentioned before, compared to smaller scenarios, the number of stores in the network and requests, enlarges the solution search space, consequently making the search for solutions computationally more demanding. Due to time limitations, the algorithm instances tested in this scenario had fewer iterations to search for solutions. As a result of all these factors, the task of finding promising solutions in this scenario becomes more challenging. In this scenario, the genetic algorithm and the genetic algorithm with elitism using an initial population consisting of Dynamic2 solutions were the only instances capable of improving the baseline Dynamic2 algorithm. This suggests that starting with a population of solutions that has fitness values close to the baseline solution and re-

combining and mutating these solutions can find a better solution than the baseline in fewer generations than starting, for example, with a random population. In scenario 2, the genetic algorithm with elitism's instances' best found solutions had higher fitness values than the ones found by the Genetic algorithm. This might be because passing the best found solutions of the previous generation on to the next one and recombining these solutions can yield better results in situation when there are not many generations for the algorithm to go through, when compared to the other two scenarios. Furthermore, in scenarios 1 and 3, the Genetic algorithm proved to find better solutions than the Genetic algorithm with Elitism when starting with an initial population consisting of solutions generated through random mechanisms, while the Genetic algorithm with Elitism's best found solutions had higher fitness values than the Genetic algorithm when starting with an initial population consisting of solutions generated by the Dynamic2 algorithm.

Regarding the single solution based algorithms, the Tabu Search algorithm with Intensification and Diversification mechanisms had, all around, the best results. The balance between the intensification and diversification mechanisms proved efficient when compared to the Tabu Search algorithm, that does not feature the medium-term memory that its counterpart does. The Simulated Annealing algorithm using jumps to Dynamic2 solutions found the best solution out of all the single solution based algorithms in scenario 1, which is the smaller sized scenario of the three, by jumping to Dynamic2 solutions, that have an underlying logic for the collection of products and for the routes, and improving these solutions. However, for example, in scenario 3, this algorithm proved to have the worst of the best found solutions in terms of fitness value, because as explained in the subsection 4.3.3, the number of possible Dynamic2 solutions is more limited. The Simulated Annealing algorithm proved to have good, all around results, having the second best found solution in scenario 2, of all the single solution based algorithms starting with an initial solution generated through random mechanisms, also having the second best found solution in terms of fitness values, of all the single solution based algorithms, in scenario 3.

In conclusion, in scenario 1 and 3, all the developed optimization algorithms managed to find solutions that improved on the baseline Dynamic2's solution fitness value. In scenario 2, due to the computational complexity and time restrictions, only the the Genetic algorithm and the Genetic algorithm with Elitism using an initial population consisting of solutions generated by the Dynamic2 algorithm managed to find solutions with improved fitness value over the baseline Dynamic2 solution. In this scenario, the developed optimization algorithms were also tested with a solution generated through random mechanisms as initial solution, in the case of the single solution based algorithms, and a population of these solution as initial

population for the Genetic algorithm and Genetic algorithm with Elitism. These instances also found solutions with improved fitness values over the fitness of their initial solutions or populations. For scenarios 1 and 3, the optimization algorithm with the best found solution in terms of fitness value was the Genetic algorithm with random initial population, finding solutions with fitness values 56.14% and 92.07% greater than the baseline Dynamic2 solution's fitness value, respectively. In scenario 2, it was the Genetic algorithm with Elitism with an initial population consisting of solutions generated by the Dynamic2 algorithm that found the best result, finding its best solution with a fitness value approximately 1.00% greater than the baseline Dynamic2 solution's fitness value.

As future work, it is recommended that the developed optimization algorithms are once again tested with higher computational power, so that more conclusions can be drawn from larger sized scenarios, such as scenario 2, and more parameter values for the optimization algorithms can also be tested. A detailed analysis regarding the execution time of the optimization algorithms developed should also be explored since these algorithms have time restrictions when applied to the retail environment. It would also be of interest to develop and experiment hybrid metaheuristics, in order to further enhance the balance between intensification and diversification, by joining elements of both single solution and population based algorithms. The research can also evolve in terms of the inclusion sustainability factors, for example, the life cycle of a product.

Bibliography

Peter Adby. *Introduction to optimization methods*. Springer Science & Business Media, 2013.

Roberto Baldacci, Maria Battarra, and Daniele Vigo. The vehicle routing problem: latest advances and new challenges. *by Bruce Golden, S. Raghavan and Edward Wasil. Springer. Chap. Routing a heterogeneous fleet of vehicles*, pages 3–27, 2008.

Raúl Baños, Julio Ortega, Consolación Gil, Antonio L Márquez, and Francisco De Toro. A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. *Computers & industrial engineering*, 65(2):286–296, 2013.

Farah Belmecheri, Christian Prins, Farouk Yalaoui, and Lionel Amodeo. Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows. *Journal of intelligent manufacturing*, 24(4):775–789, 2013.

Russell Bent and Pascal Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information sciences*, 237:82–117, 2013.

Edmund K Burke, Edmund K Burke, Graham Kendall, and Graham Kendall. *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2014.

Leandro C Coelho, Jean-François Cordeau, and Gilbert Laporte. Thirty years of inventory routing. *Transportation Science*, 48(1):1–19, 2014.

Pablo Cortés, Rodrigo A Gómez-Montoya, Jesús Muñuzuri, and Alexander Correa-Espinal. A tabu search approach to solving the picking routing problem for large-and medium-size distribution centres con-

sidering the availability of inventory and k heterogeneous material handling equipment. *Applied Soft Computing*, 53:61–73, 2017.

Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2-3):243–278, 2005.

Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

Teofilo F Gonzalez. *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC, 2007.

Dorit S Hochba. Approximation algorithms for np-hard problems. *ACM Sigact News*, 28(2):40–52, 1997.

Laetitia Jourdan, Matthieu Basseur, and E-G Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.

Philip N Klein and Neal E Young. Approximation algorithms for np-hard optimization problems. In *Algorithms and theory of computation handbook: general concepts and techniques*, pages 34–34. 2010.

Mykel J Kochenderfer and Tim A Wheeler. *Algorithms for optimization*. Mit Press, 2019.

Manoj Kumar, Dr Husain, Naveen Upreti, Deepti Gupta, et al. Genetic algorithm: Review and application. *Available at SSRN 3529843*, 2010.

Manuel Laguna, Rafael Marti, and Rafael Cunquero Martí. *Scatter search: methodology and implementations in C*. Springer Science & Business Media, 2003.

Douglas M Lambert and Martha C Cooper. Issues in supply chain management. *Industrial marketing management*, 29(1):65–83, 2000.

Yongbo Li, Hamed Soleimani, and Mostafa Zohal. An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives. *Journal of cleaner production*, 227: 1161–1172, 2019.

Holger R Maier, Saman Razavi, Zoran Kapelan, L Shawn Matott, J Kasprzyk, and Bryan A Tolson. Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. *Environmental modelling & software*, 114:195–213, 2019.

Wil Michiels, Emile HL Aarts, and Jan Korst. *Theoretical aspects of local search*, volume 13. Springer, 2007.

Nguyen Minh Ngoc, Dinh Thanh Viet, Nguyen Hoang Tien, Phuoc Minh Hiep, N Anh, LN Troung, N Anh, L Trung, V Dung, and L Thao. Russia-ukraine war and risks to global supply chains. *International Journal of Mechanical Engineering*, 7(6), 2022.

Weerakorn Ongsakul and Vo Ngoc Dieu. *Artificial intelligence in power system optimization*. Crc Press, 2013.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

Yang-Byung Park, Jun-Su Yoo, and Hae-Soo Park. A genetic algorithm for the vendor-managed inventory routing problem with lost sales. *Expert systems with applications*, 53:149–159, 2016.

Pablo Pedregal. *Introduction to optimization*, volume 46. Springer, 2004.

Tejinder Singh, Jeffrey E Arbogast, and Nicoleta Neagu. An incremental approach using local-search heuristic for inventory routing problem in industrial gases. *Computers & Chemical Engineering*, 80: 199–210, 2015.

SN Sivanandam and SN Deepa. Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37. Springer, 2008.

Jaikishan T Soman and Rahul J Patil. A scatter search method for heterogeneous fleet vehicle routing problem with release dates under lateness dependent tardiness costs. *Expert Systems with Applications*, 150:113302, 2020.

Kenneth Sörensen and Fred Glover. Metaheuristics. *Encyclopedia of operations research and management science*, 62:960–970, 2013.

Hartmut Stadtler, Hartmut Stadtler, Christoph Kilger, Christoph Kilger, Herbert Meyr, and Herbert Meyr. *Supply chain management and advanced planning: concepts, models, software, and case studies*. Springer, 2015.

El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

Franziska Theurich, Andreas Fischer, and Guntram Scheithauer. A branch-and-bound approach for a vehicle routing problem with customer costs. *EURO Journal on Computational Optimization*, 9:100003, 2021.

Pieter Vansteenwegen and Manuel Mateo. An iterated local search algorithm for the single-vehicle cyclic inventory routing problem. *European Journal of Operational Research*, 237(3):802–813, 2014.

Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

Yiyong Xiao and Abdullah Konak. A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem. *Journal of Cleaner Production*, 167:1450–1463, 2017.

Xin-She Yang. *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons, 2010a.

Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010b.

Xin-She Yang, Suash Deb, and Simon Fong. Metaheuristic algorithms: optimal balance of intensification and diversification. *Applied Mathematics & Information Sciences*, 8(3):977, 2014.