UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FELIPE FURTADO LORENCI

The joint order batching and picking routing problem: algorithms and new formulation

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Santiago Valdés Ravelo

Porto Alegre
May 2022

CIP — CATALOGING-IN-PUBLICATION

Lorenci, Felipe Furtado

The joint order batching and picking routing problem: algorithms and new formulation / Felipe Furtado Lorenci. – Porto Alegre: PPGC da UFRGS, 2022.

78 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Santiago Valdés Ravelo.

1. Metaheuristic. 2. Genetic algorithm. 3. Order batching problem. 4. Order picking problem. 5. Warehouses. I. Ravelo, Santiago Valdés. II. Título.

“Scientific knowledge is in perpetual evolution;
it finds itself changed from one day to the next.”
— Jean Piaget

AGRADECIMENTOS

Primeiramente, agradeço à Deus pela força, luz e bençãos que me deu durante este processo; também sou grato pelas oportunidades, pela saúde, perseverança e eterna curiosidade e inconformismo, que são catalisadores da minha jornada, assim como o café que tomo todas as manhãs (e tardes, às vezes, noites).

Agradeço aos meus pais por todo o amor, carinho, incondicional apoio e suporte que me forneceram durante este período. Este trabalho é dedicado à vocês, que são minha eterna fonte de inspiração. Estendo esta gratidão à todos os meus familiares, em particular aos meus avós, que mesmo por alguns momentos em distância física, sempre me enviaram energias positivas e me mantiveram presente em suas orações.

Não tenho palavras para expressar meus agradecimentos à minha companheira Isabella, que mesmo nos momentos mais complicados esteve ao meu lado, sempre me encorajando e me mostrando caminhos mais simples para problemas que aparentemente seriam insolúveis. Te agradeço de verdade por todo teu companheirismo, afeto, amor e compreensão. Obrigado por tornar os meus dias mais doces. Também agradeço aos meus sogros, por toda a torcida e parceria.

Agradeço ao meu orientador, Profº Santiago, por todos os ensinamentos, pela paciência, excelente acompanhamento e direcionamento. Também agradeço por sua disponibilidade, sempre respondendo minhas dúvidas de forma rápida e eficiente. Te agradeço por toda a caminhada, sem o senhor, este trabalho não seria possível.

Também, sou eternamente grato à Universidade Federal do Rio Grande do Sul (UFRGS) e ao instituto de informática (INF), por promover um ensino público de qualidade ótima e por Eu ter tido a oportunidade de fazer parte dessa história.

Finalmente, mas não menos importante, deixo registrada minha gratidão ao meu fiel amigo e companheiro felino Fofão. Te agradeço por todas as noites acordados em que passamos juntos, e claro, também por todas as manhãs que tu me acordou e me ajudou a levantar.

A todos vocês, meu muito obrigado!

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

WMS	Warehouse management systems
ERP	Enterprise resource planning
OPP	Order picking problem
OBP	Order batching problem
JOBPRP	Joint order batching and picking routing problem
GA	Genetic algorithm
OO-GA	Object-oriented genetic algorithm
GGA	Grouped genetic algorithm
DP	Dynamic programming
TSP	Traveling salesman problem
STSP	Steiner traveling salesman problem
MIP	Mixed-integer programming
PTS	Partial tour sub-graph
TS	Tabu Search
ABHC	Attribute-based hill climber
VNS	Variable Neighborhood Search
PSO	Particle swarm optimization
ACO	Ant colony optimization
CluVRP	Clustered vehicle routing problem
ILP	Integer linear Program
MTZ	Miller-Tucker-Zemlin
BPP	Bin packing problem
LU	Lenght units
OSD	Orders similarity degree

LIST OF FIGURES

ABSTRACT

Efficiently managing large deposits and warehouses is not an easy task. The amount of variables and processes involved from the moment a consumer purchases a single product until its receipt is quite considerable. There are two major problems involving warehouses processes: the order picking problem (OPP) and the order batching problem (OBP). The OPP aims to minimize the distance traveled by a picker while collecting a set of products (orders). The OBP seeks to assign orders to batches with a capacity limit to minimize the sum of distances traveled during the retrieving of products from all batches. When these two problems are approached together, they become the Joint Order Batching and Picking Routing Problem (JOBPRP). This work proposes a novel formulation for JOBPRP and develops a dynamic programming based heuristic, and a grouping genetic algorithm with controlled gene transmission to JOBPRP. To assess our proposals, we executed computational experiments over literature datasets. The mathematical model was used within a mixed-integer programming solver (Gurobi) and tested on the smaller instances to evaluate the quality of the solutions of our metaheuristic approach. Our computational results evidence high stability for all tested instances and much lower objective value than the previously reported in the literature while maintaining a reasonable computational time.

Keywords: Metaheuristic. Genetic algorithm. Order batching problem. Order picking problem. Warehouses.

Problema do loteamento e coleta simultânea de pedidos: formulações e algoritmos

RESUMO

Administrar grandes depósitos e armazéns de forma eficiente não é uma tarefa fácil. A quantia de variáveis e processos envolvidos desde o momento em que o consumidor realiza a compra de um único produto, até o seu recebimento, é bastante considerável. Dentro deste contexto, existem dois principais problemas envolvendo processos em armazéns: o problema de coleta de pedidos (OPP) e o problema de loteamento de pedidos (OBP). O OPP tem por objetivo minimizar a distância viajada por um funcionário enquanto ele faz a coleta de uma lista de produtos (pedidos). O OBP busca agrupar pedidos em lotes, que possuem um determinado limite de capacidade, de forma que a soma das distâncias viajadas durante a coleta dos produtos de todos os lotes seja minimizada. Quando estes dois problemas são abordados de forma conjunta, a estratégia é conhecida como problema de loteamento e coleta simultânea de pedidos (JOBPRP). Este trabalho propõe uma nova formulação matemática para o JOBPRP e apresenta novas soluções algorítmicas para tal problema: uma heurística baseada em dois níveis de programação dinâmica e um algoritmo genético de agrupamento com controle de transmissão de genes. Para avaliar nossas propostas, executamos experimentos computacionais com conjuntos de dados fornecidos pela literatura. O modelo matemático foi utilizado em um software solucionador de programas inteiros-mistos (Gurobi), onde se realizaram testes com pequenas instâncias para aferir a qualidade das soluções da nossa abordagem metaheurística. Nossos resultados computacionais evidenciaram alta estabilidade para todas as instâncias testadas e menores valores objetivo que os reportados previamente na literatura, mantendo um tempo de execução razoável.

Palavras-chave: Metaheurística, Algoritmo Genético, Problema de loteamento de pedidos, Problema de coleta de pedidos, Armazéns.

1 INTRODUCTION

Traditionally, the efficient management of large deposits and warehouses is not an easy task. The amount of variables and processes involved from the moment a consumer purchases a single product until its receipt is quite considerable. Now, imagine the case of retailers and wholesalers who sell a large number of products and receive thousands of different orders daily, so that each order demands different quantities and types of each item: the problem gets a little more complex.

In the last decades, the growth of internet services boosted e-commerce (sales through websites and applications) that has been standing out as a sustainable, scalable, and profitable business model. Just in 2021, the estimated movement of this sales model was around $ 469.2 billion, only in the United States (HOFSTETTERL, 2021). Statistics also point to a projection of $ 563.388 billion for 2025 (DEPARTMENT, 2022). This huge market is based on attractive prices, intuitiveness at the purchase moment, and, of course, short delivery times. In this way, to improve all logistics from sale to receipt, efficient warehouse management systems (WMS) interconnected with Enterprise Resource Planning (ERP) systems become indispensable for merchants that aim financial health on their companies (BOYSEN; de Koster; WEIDINGER, 2019).

Taking into account the current WMS, a significant challenge within the warehouses is the dynamics that involve the picking of products purchased by customers, once studies indicate that about 55% of the operational costs in these places are related to this process (PANSART; CATUSSE; CAMBAZARD, 2018). As a result of this fact, practical combinatorial optimization problems arise, such as the Order Picking Problem (OPP) and the Order Batching Problem (OBP).

The OPP aims to minimize the distance traveled by a human or robot (Figure 1.1) employee (pickers) in the aisles of a warehouse during the picking of a product's list. OBP is an interconnected problem that aims to group customer orders in batches to be assigned to the pickers, where each batch must satisfy a maximum capacity limit (weight/volume). The idea is to find a distribution of orders into batches that minimizes the distance's sum of the routes traveled by each picker to retrieve the requested products. Both OPP and OBP are problems that belong to NP-Hard complexity class (GADEMANN; VELDE, 2005; KULAK; ŞAHIN; TANER, 2012).

Note that OBP depends on some routing metric to find possible order batches. When this problem is treated isolated, options to deal with distances may use order-

Figure 1.1 – Amazon's Kiva robots. Source: <https://roboticsandautomationnews.com/2020/01/21/amazon-now-has-200000-robots-working-in-its-warehouses/28840/>. Accessed in Jan. 2022.

picking routing policies (e.g., S-Shape, Largest Gap, Midpoint) or advanced heuristics. However, an alternative is to optimize the correspondent route of each batch during the batching process individually, that is, an approach that performs OBP jointly with OPP (SCHOLZ; WäSCHER, 2017). This strategy is known as Joint Order Batching and Picking Routing Problem (JOBPRP) (BRIANT et al., 2020).

In this work, we present a novel mathematical model and propose a heuristic and a grouping genetic approach to JOBPRP on single-blocks warehouses. Our model works independently on the warehouse layout (i.e., it could be multi-block, multi-floor, or assume other formats beyond rectangular) and was used for solving smaller instances. The heuristic is based on two levels of dynamic programming (DP) and provides suitable results in contrast to the literature. The metaheuristic approach applies another heuristic process to generate the initial solutions, assigning the most similar orders to the same batch. The distance traveled in each batch is computed with a DP algorithm, and the crossover applies a controlled gene transmission technique.

To assess our metaheuristic proposal, we executed comparative tests with a mathematical optimization solver running our model. Also, we extend the metaheuristic experiments over large datasets. Our results evidence high stability in terms of standard deviation (considering 30 experiments with different seeds, for each instance) and lower objective values when confronted with other studies.

The remainder of this work is organized as follows: in the next chapter we formally define the JOBPRP. In chapter 3 we review the literature. In chapter 4, our new mathematical formulation is presented. In chapter 5, we explain our two level DP heuristic approach to JOBPRP. In chapter 6, we give a detailed explanation of our genetic algorithm. Chapter 7 shows the results of experimental tests, including comparisons with previous approaches of the literature. In the last section, we discuss important points of this research, highlighting our future work directions.

2 PROBLEM DEFINITION

In this chapter, we give an intuitive overview of JOBPRP, show examples of use, and formalize the problem definition. As discussed before, reducing warehouses operational costs is a quite challenging task and involves many processes. We focus on those that occur at the dispatch area (also known as depot), where the manager receives orders from clients and assigns them to employees that will pick the specified consumer goods in the warehouse.

In this scenario, a warehouse layout is usually represented by a rectangular area (Figure 2.1) divided into parallel aisles (Figure 2.2), which interconnect through two other cross aisles (one below and one above) (Figure 2.3).

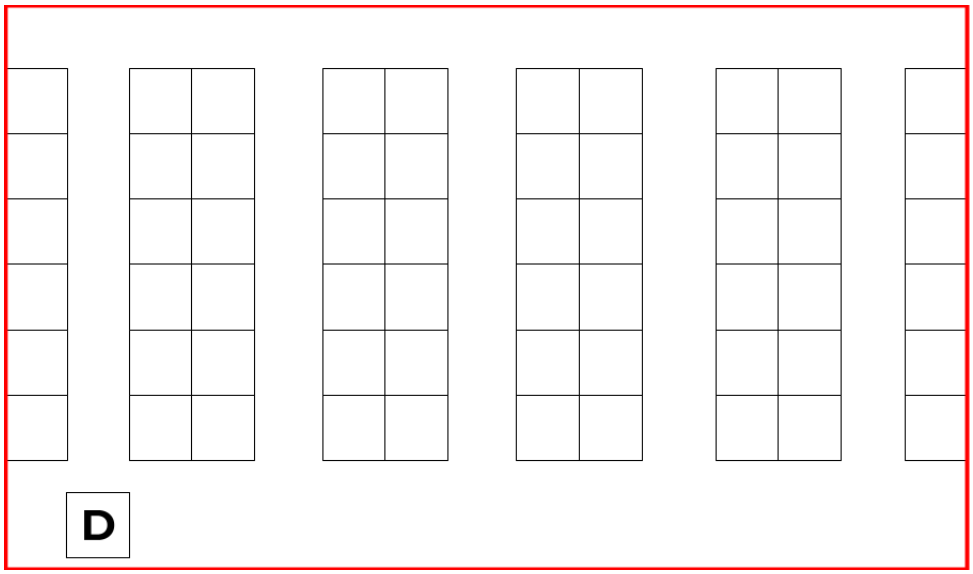

Figure 2.1 – Warehouse with rectangular area. Source: the author.

The products are stored on shelves located to the right and left of each parallel aisle (Figure 2.4). Each shelve has an associated code, called **picking-position** (Figure 2.5), and any two shelves facing each other at the same aisle are considered to have the same picking-position. Figure 2.7 illustrates a warehouse with 5 parallel aisles and some products on their shelves.

Each product has an associated **address** (Figure 2.6) defined by a pair $\langle$aisle, picking-position$\rangle$, that aims to identify the product's locations in the warehouse. In the example of Figure 2.7, if we consider the parallel aisles numbered from left to right, beginning at 0 and ending at 4, and the picking-position at each aisle from bottom to top, beginning at 0 and ending at 6, then the graphic card is located at address $\langle$0,2$\rangle$, while the video-game is at address $\langle$3,3$\rangle$, the same address of the Pen Drive. In the same figure,

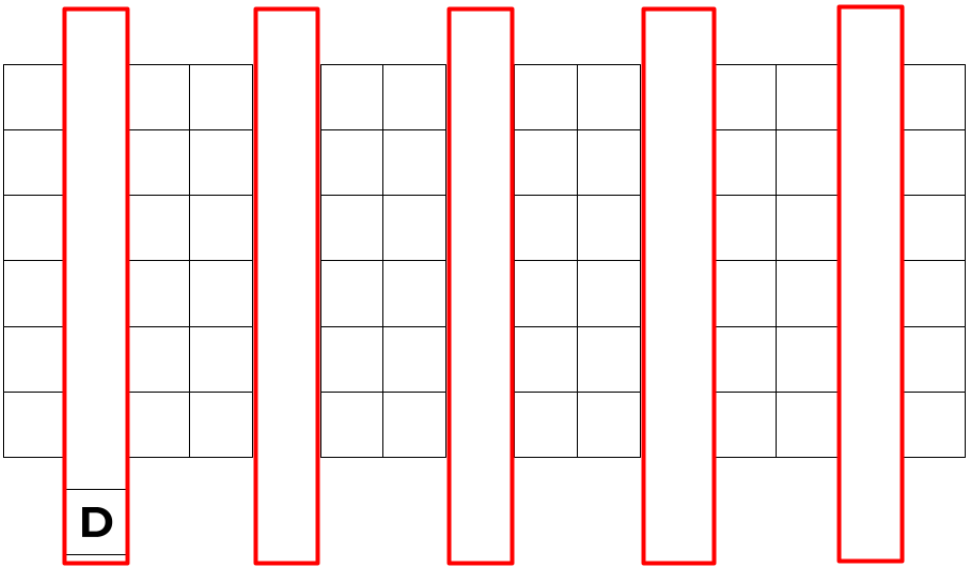

Figure 2.2 – The warehouse aisles. Source: the author.

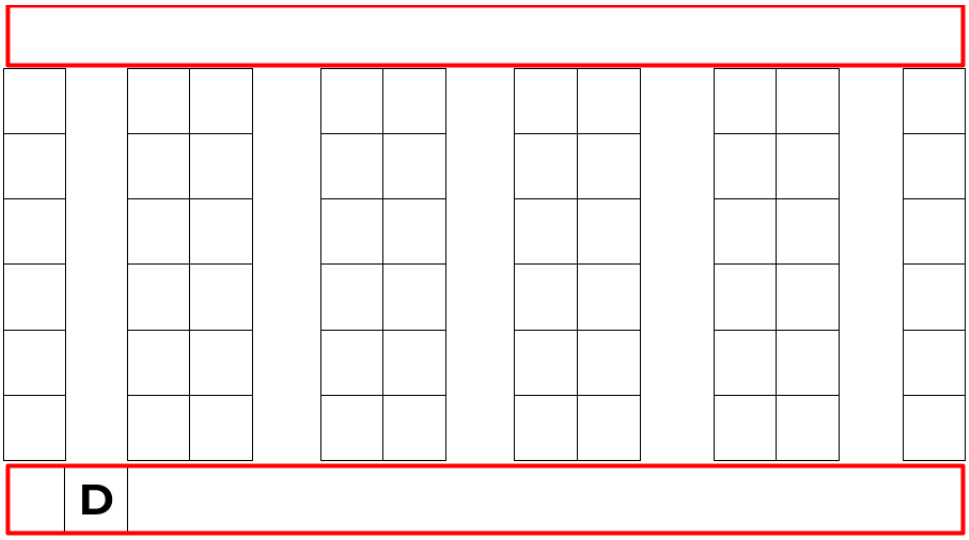

Figure 2.3 – The warehouse cross aisles. Source: the author.

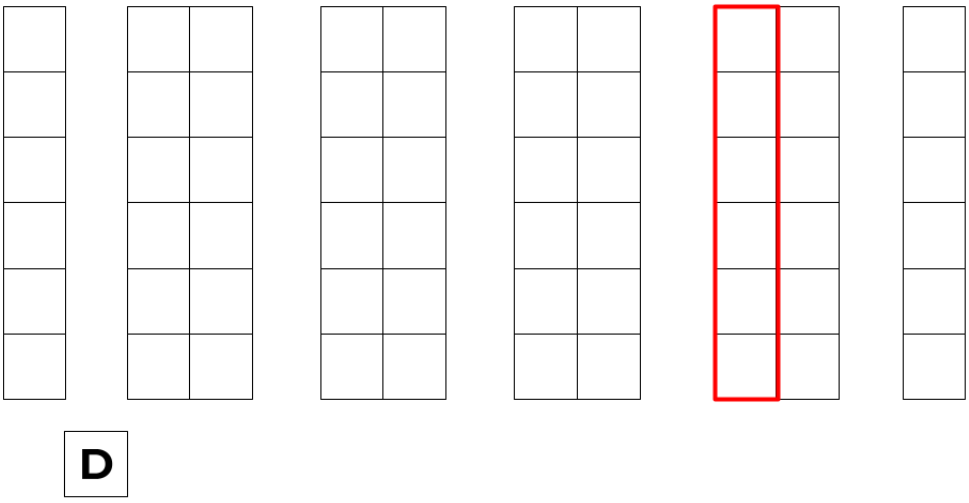

Figure 2.4 – The warehouse shelves. Source: the author.

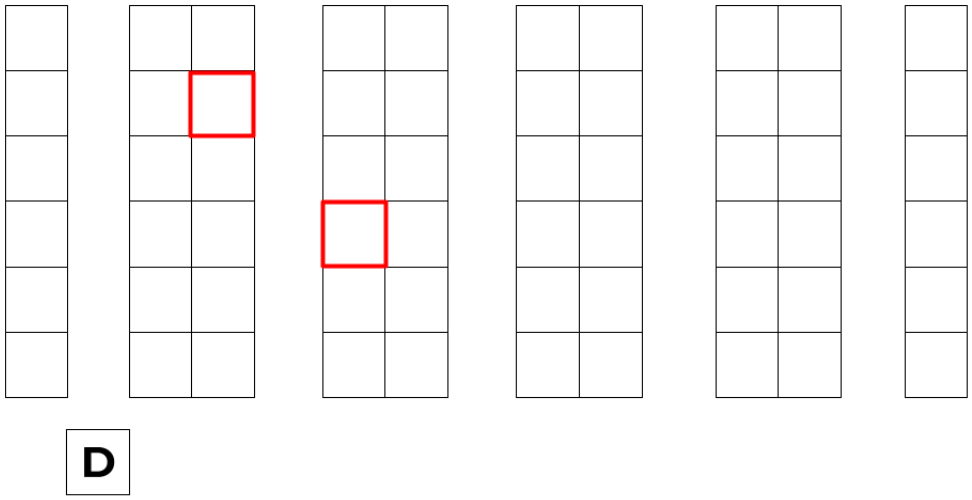

Figure 2.5 – Example of picking positions. Source: the author.

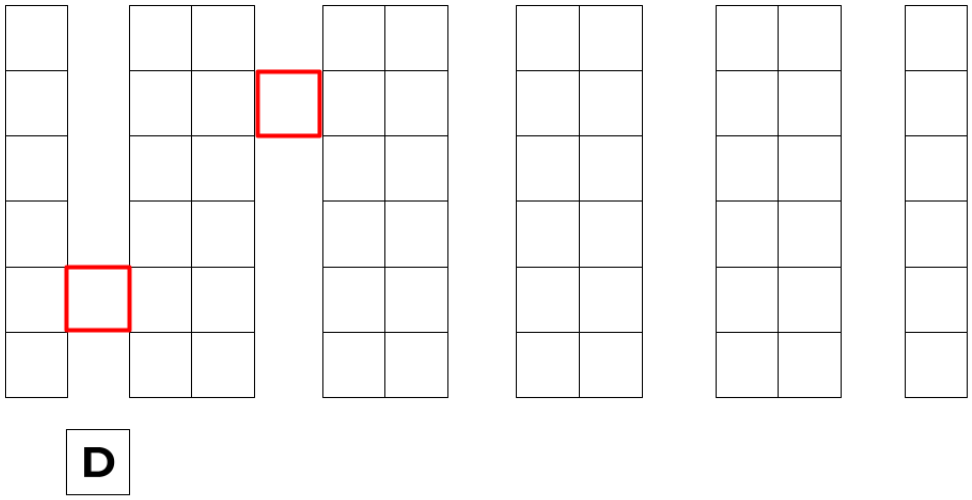

Figure 2.6 – Example warehouse's addresses. Source: the author.

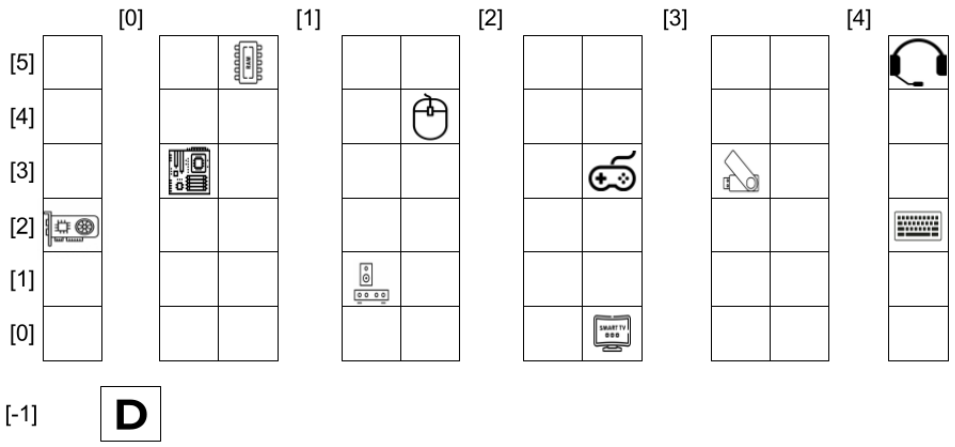

Figure 2.7 – An example of warehouse with highlighted products. Source: the author.

the location **D** denotes the depot, placed at the interception between the aisle 0 and the inferior cross-aisle; thus, its address is $\langle 0,\text{-}1 \rangle$.

An **order** is a collection of pairs $\langle$product, address$\rangle$ (examples on Table 2.1), and represents goods to be retrieved. A **batch** is a set of orders. The orders are indivisible, i.e., all the products specified in an order must be allocated to the same batch. Each batch is assigned to a picker to collect the requested products.

Table 2.1 – Orders example. Source: the author.

Orders			
Order 1		**Order 2**	
Product	**Address**	**Product**	**Address**
Headset	$\langle 4,5 \rangle$	RAM Memory	$\langle 1,5 \rangle$
Keyboard	$\langle 4,2 \rangle$	Pen Drive	$\langle 3,3 \rangle$
Bluetooth Mouse	$\langle 2,4 \rangle$		
Pen Drive	$\langle 3,3 \rangle$		
Order 3		**Order 4**	
Product	**Address**	**Product**	**Address**
Soundbar	$\langle 1,1 \rangle$	Motherboard	$\langle 0,3 \rangle$
Smart TV	$\langle 3,0 \rangle$	Graphic card	$\langle 0,2 \rangle$
Headset	$\langle 4,5 \rangle$	Video-game	$\langle 3,3 \rangle$
		Pen Drive	$\langle 3,3 \rangle$
Order 5		**Order 6**	
Product	**Address**	**Product**	**Address**
Headset	$\langle 4,5 \rangle$	RAM Memory	$\langle 1,5 \rangle$
Pen Drive	$\langle 3,3 \rangle$	RAM Memory	$\langle 1,5 \rangle$
Keyboard	$\langle 4,2 \rangle$		

The path traveled by the picker during the batches' products retrieving is known as **picking-route** (Figure 2.8), defined as a list of pairs $\langle$label, address$\rangle$, where **label** can be a product to be picked or the depot. Each picking-route has an associated distance equal to the sum of length units between each pair of consecutive addresses in its list. Every picking-route starts and ends at the depot. Each product has an associated weight, and since pickers have a capacity limit (weight/volume), each batch also needs a load bound. We define **c-capacitated picking-route** as a picking-route with the maximum capacity equal to c (i.e., the weight's sum of all products picked at the route should be less than or equal to c).

To guarantee the existence of feasible solutions, we assume that the total weight of an order does not exceed the capacity limit of a picker. Besides, we suppose that when a product is contained in an order, its availability is guaranteed on the stock. Note that

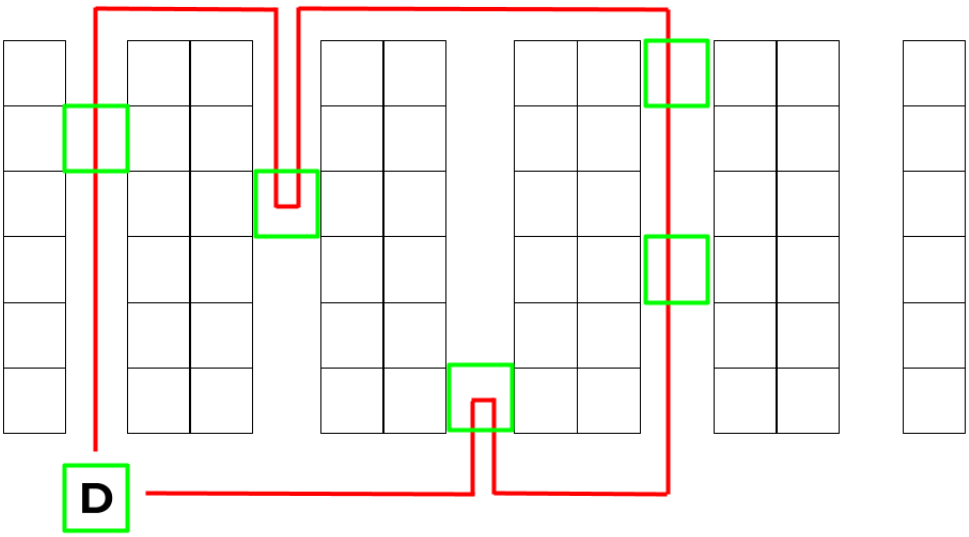

Figure 2.8 – An example of picking-route. Source: the author.

some products can be requested in multiple orders, as well as some orders may demand more than just a single unit of a product.

To exemplify the JOBPRP, consider the orders in Table 2.1 and assume that each item has a weight 1 and the capacity limit of each picker is equal to 6. The task is to group the orders so that the total weight of each batch does not exceed 6, as this is the maximum load that a picker can carry on. Hence, each batch has a corresponding 6-capacitated picking-route, and the objective is to minimize the sum of its distances.

A possible solution to the problem is the following batches configuration:

- **Batch 1:** order 1 and order 2
- **Batch 2:** order 3 and order 6
- **Batch 3:** order 4
- **Batch 4:** order 5

Furthermore, feasible options to 6-capacitated picking-routes are:

- **Batch 1:** Depot, $\langle 0,-1 \rangle \rightarrow$ Ram Memory, $\langle 1,5 \rangle \rightarrow$ Bluetooth Mouse, $\langle 2,4 \rangle \rightarrow$ Pen Drive, $\langle 3,3 \rangle \rightarrow$ Headset, $\langle 4,5 \rangle \rightarrow$ Keyboard, $\langle 4,2 \rangle \rightarrow$ Depot, $\langle 0,-1 \rangle$
- **Batch 2:** Depot, $\langle 0,-1 \rangle \rightarrow$ Soudbar, $\langle 1,1 \rangle \rightarrow$ RAM Memory, $\langle 1,5 \rangle \rightarrow$ Headset, $\langle 4,5 \rangle \rightarrow$ Smart TV, $\langle 3,0 \rangle \rightarrow$ Depot, $\langle 0,-1 \rangle$
- **Batch 3:** Depot, $\langle 0,-1 \rangle \rightarrow$ Graphic Card, $\langle 0,2 \rangle \rightarrow$ Motherboard, $\langle 0,3 \rangle \rightarrow$ Video-game and Pen Drive, $\langle 3,3 \rangle \rightarrow$ Depot, $\langle 0,-1 \rangle$
- **Batch 4:** Depot, $\langle 0,-1 \rangle \rightarrow$ Pen Drive, $\langle 3,3 \rangle \rightarrow$ Headset, $\langle 4,5 \rangle \rightarrow$ Keyboard, $\langle 4,2 \rangle$) $\rightarrow$ Depot, $\langle 0,-1 \rangle$

However, thinking about distance minimization, is the above solution an optimal batches-routes assignment? This is the central question that JOBPRP aims to answer, which is formally defined below.

Problem 1 *Joint Order Batching and Picking Routing Problem* JOBPRP

Input: *a tuple* $\langle n, c, \mathcal{E}, \omega, \phi, \delta \rangle$*, where:*

- $n \in \mathbb{N}$ *indicates the total number of orders;*
- $c \in \mathbb{N}$ *indicates the maximum weight capacity of the batches;*
- $\mathcal{E}$ *is the warehouse addresses set. The depot is contained on* $\mathcal{E}$*. We denote as* $|\mathcal{E}|$ *the total number of warehouse addresses;*
- ω *is a vector, where* ω_o *represents the weight's sum of the products in the* o*-th order (*$1 \leq o \leq n$*);*
- ϕ *is a vector, where* $\phi_o : \mathcal{E} \rightarrow \{0, 1\}$ *is a binary function that assumes value* 1 *if address* $i \in \mathcal{E}$ *is required at the* o*-th order, and* 0 *otherwise (*$1 \leq o \leq n$*);*
- $\delta : \mathcal{E} \times \mathcal{E} \rightarrow \mathbb{R}$ *is a function that returns the distance between a pair of addresses* $i, j \in \mathcal{E}$*.*

Output*: a set of* c*-capacitated picking-routes, such that each order is completely contained in exactly one picking-route, and the sum of all picking-routes' distances is minimized.*

In this chapter, we intuitively present the main concepts and definitions concerning the target problem of this work. We also define the JOBPRP formally and provide application examples. In the next chapter, we present a quick literature review about OPP, OBP and JOBPRP.

3 LITERATURE REVIEW

In this chapter, we present the theoretical foundation of our research. In the sections 3.1 and 3.2 we realize a brief overview about OPP, OBP and JOBPRP, highlighting the most relevant results found at scientific journals and events proceedings from optimization and operational research areas.

3.1 Order Picking Problem

A common problem faced within warehouses is the OPP. An order is a set of products from the warehouse storage. When an order arrives, the manager assigns it to a picker for collecting the orders' items. The picker leaves from the shipping area (also known as dispatch or depot), retrieves the products and returns to depot. The problem objective is to minimize the distance traveled by the picker.

A well-known DP algorithm for solving the OPP in rectangular single-block warehouses (multiple parallel aisles with two cross-aisles, one above, one below) was presented by (RATLIFF; ROSENTHAL, 1983). The proposed DP algorithm finds an optimal solution for these layouts in linear time on the number of aisles and picking locations. We discuss this approach in detail in subsection 3.1.1.

An extension of Ratliff and Rosenthal's DP approach was proposed by (ROODBERGEN, 2001), to the case where the warehouse's layout is two-block, i.e., when it also has a cross-aisle in the middle of parallel aisles, resulting in 3 cross-aisles. The picking lists sizes of their experiments varied from 1 to 50 items, and the results attested that their DP maintains the quality and a proportional computation time from the single-block original approach. Another point is the laborious implementation, which demands many details.

The OPP can be generalized for multi-blocks and considered as a Steiner traveling salesman problem (STSP). Authors from (THEYS et al., 2010) formulated this problem as a TSP. They proposed a heuristic which, despite being more generic than state-of-the-art from that epoch, presents more improved results considering the specific implementation process that other works destined to implement solutions to the same problem.

Specific mathematical programming approaches were also applied to OPP. In (SCHOLZ et al., 2016), the authors presented a huge mathematical programming formulation for the single-picker routing problem (another name to OPP) on single-block

warehouses. They adapted the theory delivered by (RATLIFF; ROSENTHAL, 1983) to develop particular constraints based on the possible movements and transitions between aisles and cross-aisles. They proved that their formulation was superior to standard TSP formulations applied to the picking routing problem on benchmark tests. A sparse mixed-integer programming (MIP) formulation strengthened by preprocessing and valid inequalities was developed by (PANSART; CATUSSE; CAMBAZARD, 2018). They reported and compared the results with the TSP solver Concorde.

The OPP accept variations, as the picker routing in rectangular mixed/scattered shelves. This problem also receives (as part of the input) an order with the respective products to be picked; however, the products have multiple items scattered on shelves around the warehouse. The task is to decide the picking sequence and which unit of each product should be picked to minimize the total traveled distance. The research of (WEIDINGER, 2018) and (WEIDINGER; BOYSEN; SCHNEIDER, 2019) dealt with this problem through heuristics and MIP Models. The latter work considered the variation of multiple depots.

3.1.1 A Dynamic Programming approach

In this subsection, we summarize and rewrite the content of (RATLIFF; ROSENTHAL, 1983), a DP approach to OPP, once their mechanism and theory are essential for developing our work (we discuss how this DP is applied to our algorithms in chapters 5 and 6). Ratliff and Rosenthal treat the warehouse single-block configuration, jointly with the respective products to be picked, as a graph.

Their algorithm's main idea is based on proven theorems and aims to find a minimum length tour sub-graph representing the picking-route (defined in the previous chapter). The approach constructs the minimum length tour sub-graph from the union of the shortest partial tours sub-graphs (PTS). PTS are also sub-graphs, and their meaning is explained below.

They assume that the warehouse has m aisles, and the order has n products. The most left aisle is identified by 1, the next aisle by 2, and so on. The above intersection of an aisle j with the superior cross-aisle is identified by a_j, while the below intersection with the inferior cross-aisle is identified by b_j. The depot is identified by v_0, and the points to be visited (the product's locations, among a_j and b_j) are identified from v_1 to v_n. The sub-graph which represents the route traveled at aisle j is defined by A_j.

The single-block layout allows the definition of two types of PTS: the L_j^- and the L_j^+ PTS. The first one considers all the sub-graph to the left from a_j and b_j. The second is the union between L_j^- and A_j ($L_j^+ = L_j^- \cup A_j$). The relation between these two PTS is depicted in Figure 3.1. Observe that a tour always starts at L_1^+ PTS, and the minimum length tour is at L_m^+ PTS.

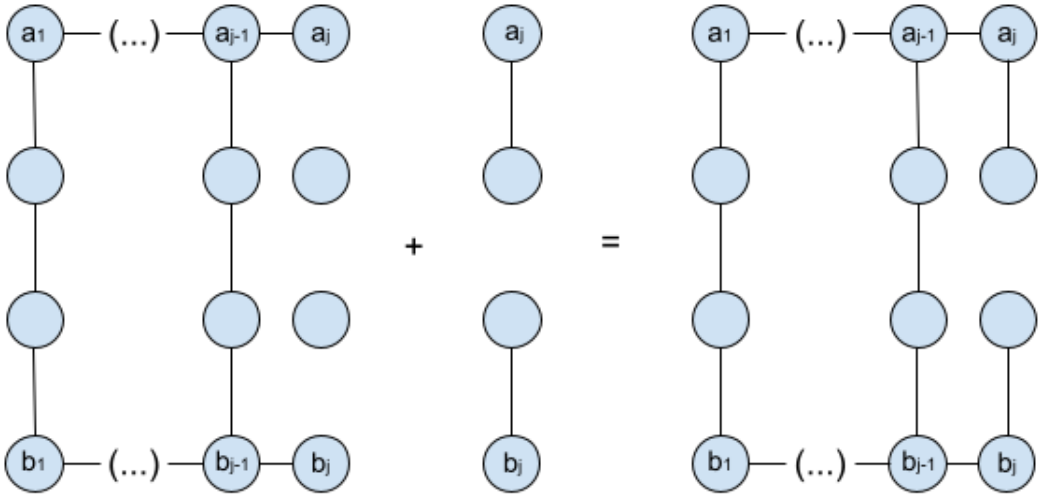

Figure 3.1 – Relation $L_j^+ = L_j^- \cup A_j$. Source: the author.

In Figure 3.1, are abstracted the content and the route of aisles and cross-aisles. Still, there are only six possibilities of arc configurations for any aisle j (Figure 3.2) and five possibilities of arc configurations between aisle j and $j + 1$ (Figure 3.3).

These arc configurations occur in the following situations for the aisles:

(i) when the picker traverse the aisle, from a cross-aisle to another, one time;

(ii) when the picker enters at the aisle by the superior interception (a_j), pick the products from the order which are located there, and leaves the aisle by the same interception a_j;

(iii) when the picker enters at the aisle by the inferior interception (b_j), pick the products from the order which are located there, and leaves the aisle by the same interception b_j;

(iv) when the picker enters at the aisle by a_j (or b_j), pick products at aisle j, leaves j by the same interception, continues the picking route at the warehouse, and while returning to depot, enters again at j, but now by b_j (or a_j) and leaves the aisle by the same interception;

(v) when the picker traverse the aisle twice, from a cross-aisle to another;

(vi) when the picker do not enter at the aisle, because the order do not request products from there.

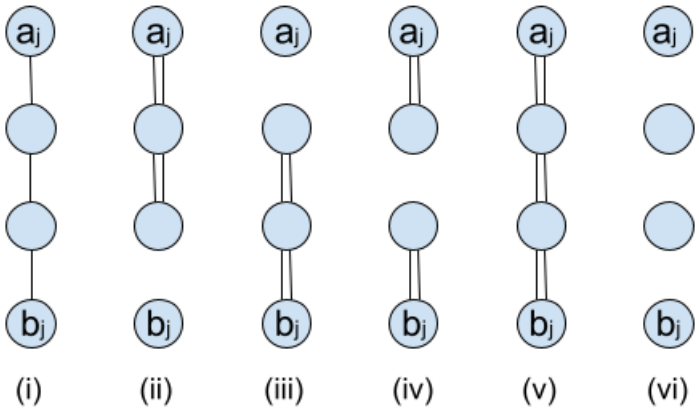

Figure 3.2 – Possible arc configuration in aisles. Source: adapted from (RATLIFF; ROSENTHAL, 1983).

The five arc configurations of cross-aisles occur in the following situations:

(i) when the transition between aisles j and $j + 1$ is effected with one arc in both superior and inferior cross-aisles;

(ii) when the transition between aisles j and $j + 1$ is effected with two arcs in the superior cross-aisle;

(iii) when the transition between aisles j and $j+1$ is effected with two arcs in the inferior cross-aisle;

(iv) when the transition between aisles j and $j + 1$ is effected with two arcs in both cross-aisles;

(v) when there are not transitions between aisles j and $j+1$. This last case occurs when there are no products to be picked to the right of aisle j.

Ratliff and Rosenthal proved that with the arc configurations depicted in Figures 3.2 and 3.3 a PTS can be represented through patterns, which they defined as **equivalence classes**. An equivalence class is a triplet (degree parity of a_j, degree parity of b_j, connectivity), where the first element informs if the number of arcs linked to a_j is odd (U), even (E) or zero (0), the second gives the same information to b_j, and the third informs about the components arrangement (0C, if there are no component; 1C, if one component; or 2C, if two components). Purposely, they use U to denote odd ("uneven") to avoid confusion with zero. The possible equivalence classes are the following seven: (U, U, 1C), (0, E, 2C), (E, 0, 2C), (E, E, 1C), (E, E, 2C), (0, 0, 0C) and (0, 0, 1C).

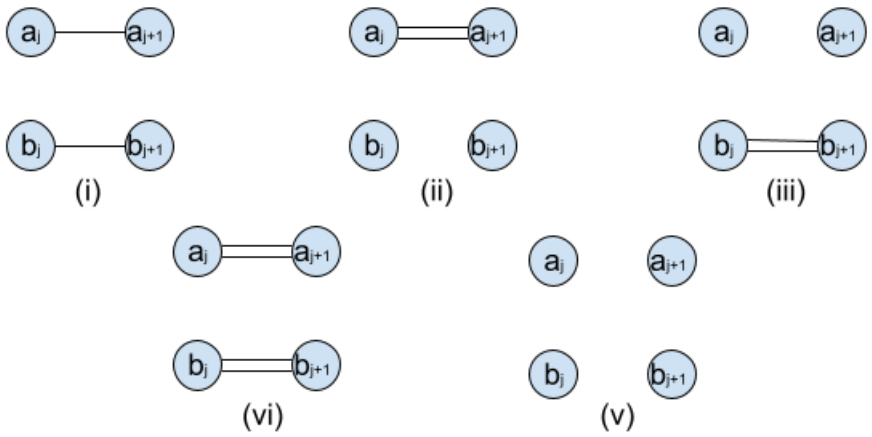

Figure 3.3 – Possible arc configuration in cross-aisles. Source: adapted from (RATLIFF; ROSENTHAL, 1983).

The optimal tour sub-graph is composed by connecting the PTS. This process is guided by two tables (Tables 3.1 and 3.2), that delivery which equivalence class we find when append some configuration of Figure 3.2 to a L_j^- PTS or when we append some configuration of Figure 3.3 to a L_j^+ PTS.

Table 3.1 – Resultant equivalent classes from adding each of arc configurations in Figure 3.2 to the PTS in the L_j^- equivalence classes. Source: adapted from (RATLIFF; ROSENTHAL, 1983).

L_j^- **Equiv. Class**	**(i)**	**(ii)**	**(iii)**	**(iv)**	**(v)**	**(vi)**
(U, U, 1C)	(E, E, 1C)	(U, U, 1C)	(U, U, 1C)	(U, U, 1C)	(U, U, 1C)	(U, U, 1C)
(E, 0, 1C)	(U, U, 1C)	(E, 0, 1C)	(E, E, 2C)	(E, E, 2C)	(E, E, 1C)	(E, 0, 1C)
(0, E, 1C)	(U, U, 1C)	(E, E, 2C)	(0, E, 1C)	(E, E, 2C)	(E, E, 1C)	(0, E, 1C)
(E, E, 1C)	(U, U, 1C)	(E, E, 1C)	(E, E, 1C)	(E, E, 1C)	(E, E, 1C)	(E, E, 1C)
(E, E, 2C)	(U, U, 1C)	(E, E, 2C)	(E, E, 2C)	(E, E, 2C)	(E, E, 1C)	(E, E, 2C)
(0, 0, 0C)	(U, U, 1C)	(E, 0, 1C)	(0, E, 1C)	(E, E, 2C)	(E, E, 1C)	(0, 0, 0C)
(0, 0, 1C)	-	-	-	-	-	(0, 0, 1C)

Table 3.2 – Resultant equivalent classes from adding each of arc configurations in Figure 3.3 to the PTS in the L_j^+ equivalence classes. Source: adapted from (RATLIFF; ROSENTHAL, 1983).

L_j^+ **Equiv. Class**	**(i)**	**(ii)**	**(iii)**	**(iv)**	**(v)**
(U, U, 1C)	(U, U, 1C)	-	-	-	-
(E, 0, 1C)	-	(E, 0, 1C)	-	(E, E, 2C)	(0, 0, 1C)
(0, E, 1C)	-	-	(0, E, 1C)	(E, E, 2C)	(0, 0, 1C)
(E, E, 1C)	-	(E, 0, 1C)	(0, E, 1C)	(E, E, 1C)	(0, 0, 1C)
(E, E, 2C)	-	-	-	(E, E, 2C)	-
(0, 0, 0C)	-	-	-	-	(0, 0, 0C)
(0, 0, 1C)	-	-	-	-	(0, 0, 1C)

Ratliff and Rosenthal describe that to find the distance of the minimum length tour, we assemble a solution table with seven lines (which of them corresponding to an equivalence class e), and the columns indexed from 1 to $2m - 1$: considering an aisle j, the odd indexes store at the lines data about the L_j^+ PTS, and the even indexes store data

about L_j^- PTS. There is a triplet in each cell of the table: the first element is the minimum distance length PTS for the equivalence class, the second indicates the equivalence class of the predecessor PTS from which the current PTS was constructed, and the third informs which arc configuration (from Figure 3.2 to L_j^+ PTS or from Figure 3.3 to L_j^- PTS) was added to the predecessor PTS to obtain the current PTS.

The algorithm starts the solution table filling process at L_1^+. Repair that L_1^- does not exist, once there is nothing to the left of a_1 and b_1. It also justifies the first operation: for each equivalence class e associated to the first column of the solution table, we search at Table 3.1 which arc configurations could be appended to (0,0,0C) (a null graph) that have the equivalence class e as a result. Among the possible arc configurations, we choose the one that minimizes the PTS distance. In the cell, we store the distance, the equivalent class of the predecessor (in this case, null), and the chosen arc configuration.

The process is similar to the next PTS, L_2^-. For each equivalence class e, we search at Table 3.2 which operations results e: from the options (an equivalence class f and an arc configuration g), we choose the one that minimizes the sum between the distance associated to the equivalent class f at last PTS and the distance traveled on arc configuration g. We again store at the cell the distance, the equivalent class of the predecessor f, and the arc configuration g. Thus, the idea to complete the fields of L_2^+ is the same as L_1^+, the difference is the total distance of the PTS, which is computed from those that we just calculated for the L_2^-' equivalence classes.

The solution table is filled until the column $2m - 1$, and the optimal distance of the tour can be found in one of (E, 0, 1C), (0, E, 1C), (E, E, 1C), or (0, 0, 1C) from this column.

We can determine the physical graph that represents the order-picking route. The triplets stored in each cell inform the distance, the equivalence class of the predecessor, and the arc configuration. Hence, we trace back the solution table, starting from the optimal PTS in $2m - 1$. The composition of aisles and cross-aisles arcs give us the complete minimum length tour sub-graph.

3.2 Order Batching Problem

Besides determining the best route to pick the products from some order, another recurrent problem in warehouses is how to increase the efficiency factor of the order picking process as a whole. Imagine the case of companies that receive thousands of

orders daily - is it advantageous to assign a single order to a picker, considering that maybe the trolley is not full yet? This question leaves us to the Order Batching Problem (OBP), a common strategy widely applied in industry, aiming to reduce operational costs.

The idea is to assign more than one order to a picker without exceeding a capacity limit, and we call these orders groups by batches. An order is indivisible, i.e., products from an order should be picked at the same picking route. Thus, the problem aims to minimize the sum of distances traveled by each picker, dividing a set of orders received by management in batches.

A well-known study about OBP was conducted by Henn and Wäscher (HENN; WäSCHER, 2012), where two approaches based on the Tabu Search (TS) principle were proposed: a classic TS and an attribute-based hill climber (ABHC), achieving very competitive results. The instances' groups provided by these authors are still used nowadays as a standard benchmark in warehouse optimization problems.

Koch and Wäscher (KOCH; WäSCHER, 2016) also explored OBP, developing a grouping genetic algorithm (GGA), which works with a local search procedure to improve the results, and the routing was done by S-Shape strategy. Their approach has been proven to be more performative than genetic algorithms (GA) based on object-oriented encoding schemes previously used in the literature.

To the best of our knowledge, the results found by Menéndez, Pardo, Alonso-Ayuso, Molina, and Duarte (MENéNDEZ et al., 2017) with a Variable Neighborhood Search (VNS) methodology are the best to OBP, outperforming previous attempts in state of the art. Their approach (MS-VNS) is a multi-start VNS, where each iteration starts by generating a different initial solution, improved with a basic VNS, and post-optimized with a general VNS strategy.

Just like OPP, OBP also have variations and extensions. Recently, motivated by the new warehouse layouts and automatizing technologies being adopted by these places, (XIE et al., 2021) introduced the OBP with split orders, meaning that it is allowed to divide the orders and mix their products into batches that must be recomposed at the end of the picking.

Note that OBP depends on some routing metric to find possible order batches. When this problem is treated isolated, options to deal with distances may use order-picking routing policies (e.g., S-Shape, Largest Gap, Midpoint) or advanced heuristics. However, an alternative is to individually optimize the correspondent route of each batch during the batching process, that is, an approach that performs OBP jointly with OPP - this

strategy is known as Joint Order Batching and Picking Routing Problem (JOBPRP).

3.2.1 Joint Order Batching and Picking Routing Problem

In (KULAK; ŞAHIN; TANER, 2012), the authors presented novel tabu search (TS) algorithms integrated with a novel clustering algorithm to solve JOBPRP for multi-blocks warehouses. The clustering algorithm generates an initial solution for the TS algorithms to provide fast and effective solutions to the order batching procedure. They randomly generated various instances, considering the number of orders, the weight of items, and picking coordinates. Their results indicated advantages (solution quality and computational efficiency) in calculating OPP and OBP jointly.

In (CHENG et al., 2015), an efficient hybrid algorithm was proposed for solving the JOBPRP. This algorithm is composed of particle swarm optimization (PSO) for finding the best batches configuration by minimizing the sum of the traveling distances. These distances are calculated by an ant colony optimization (ACO) algorithm, which searches for the most effective path for each batch. Their results reinforce that the JOBPRP strategy improves picking performance and allows customer demands to be met rapidly.

Once that JOBPRP is a strategy to improve objective values of OBP through OPP, it is possible to compare the results of both JOBPRP and OBP. In this sense, recently, Aerts, Cornelissens, and Sörensen (AERTS; CORNELISSENS; SöRENSEN, 2021) approached JOBPRP on single-block warehouses, modeling the problem as a clustered vehicle routing problem (CluVRP) and solving it with a two-level VNS metaheuristic. They delivered good results compared with the literature, except with Menéndez et. al. (MENéNDEZ et al., 2017) that maintains its superiority, mainly with larger batches capacities.

This chapter presented papers and methods that support and justify our work, reviewing the OPP, OBP, and finally, the JOBPRP. In the next chapter, we present a new mathematical formulation to JOBPRP.

4 MATHEMATICAL FORMULATION

We propose a new integer linear program (ILP) for JOBPRP. We use this model within an exact solver to solve small instances, which also helps estimate the accuracy of the algorithm proposed in the next section. From now on, we consider an instance $I = \langle n, c, \mathcal{E}, \omega_o, \phi_o, \delta \rangle$ of JOBPRP.

Our formulation is defined over two sets of binary variables and one of integer variables. The first set of binary variables controls the assignment of orders to batches. Once there are n orders (labeled from 1 to n), and each order is inserted in exactly one batch, the minimum number of batches necessary to satisfy the problem is 1 (when all orders are assigned to the same batch), and the maximum number is n (when each order is assigned to a different batch). To simplify notation, we define the set $\mathcal{N} = \{m \in \mathbb{N} | 1 \leq m \leq n\}$, and the variables x_{ob} are declared for each $o \in \mathcal{N}$ and $b \in \{m \in \mathbb{N} | 1 \leq m \leq o\}$. We enforce $b \leq o$ to avoid some symmetric solutions in which the batches are interchangeable:

$$x_{ob} = \begin{cases} 1, \text{ if order } o \text{ is in batch } b \\ 0, \text{ otherwise.} \end{cases}$$

The second set of binary variables describes the picking-routes, declaring a variable for all $i, j \in \mathcal{E}$ $(i \neq j)$, $b \in \mathcal{N}$:

$$z_{ijb} = \begin{cases} 1, \text{ if } i \text{ is visited immediately before } j \text{ in the} \\ \quad \text{picking-route of batch } b \\ 0, \text{ otherwise.} \end{cases}$$

The integer variables indicate the ordinality of the addresses visited in the batches' picking-routes. If the address $i \in \mathcal{E}$ is visited by the picking-route of batch $b \in \mathcal{N}$, then u_{ib} indicates the ordinal position (starting at the depot) in which b visits i. These variables take value from 1 to $|\mathcal{E}|$ (there are no more than $|\mathcal{E}|$ addresses), and they are mainly used to avoid sub-tours in a feasible solution.

The problem's objective is to minimize the sum of all picking-routes' distances, which can be written as follows:

$$\min \sum_{b=1}^{n} \sum_{i \in \mathcal{E}} \sum_{\substack{j \in \mathcal{E} \\ j \neq i}} \delta(i, j) z_{ijb}$$

To guarantee that each order is assigned to exactly one batch, we define the set of constraints below:

$$\sum_{b=1}^{o} x_{ob} = 1 \quad \forall o \in \mathcal{N} \tag{4.1}$$

To ensure that the capacity limit of each batch is not exceeded, we state the following group of constraints:

$$\sum_{o=b}^{n} \omega_o x_{ob} \leq c \quad \forall\, b \in \mathcal{N} \tag{4.2}$$

The next two constraints groups guarantee, for all addresses and batches, that if an address i is not requested in some of the orders of batch b, then i cannot precede nor succeed any other address in the picking-route of b:

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq i}} z_{jib} \leq \sum_{o=b}^{n} \phi_o(i) x_{ob} \quad \forall\, b \in \mathcal{N},\ i \in \mathcal{E} \tag{4.3}$$

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq i}} z_{ijb} \leq \sum_{o=b}^{n} \phi_o(i) x_{ob} \quad \forall\, b \in \mathcal{N},\ i \in \mathcal{E} \tag{4.4}$$

Also, we need to guarantee that every address has at most one predecessor and one successor at each picking-route:

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq i}} z_{jib} \leq 1 \quad \forall\, b \in \mathcal{N},\ i \in \mathcal{E} \tag{4.5}$$

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq i}} z_{ijb} \leq 1 \quad \forall\, b \in \mathcal{N},\ i \in \mathcal{E} \tag{4.6}$$

To include an address in a picking-route, we need to assure for all addresses, orders, and batches, that if an address i is requested in at least one of the orders in batch b, then i must be preceded and succeeded by another address in the picking-route of b:

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq i}} z_{jib} \geq \phi_o(i) x_{ob}\ \forall\, b \in \mathcal{N},\ b \leq o \leq n,\ i \in \mathcal{E} \tag{4.7}$$

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq i}} z_{ijb} \geq \phi_o(i) x_{ob}\ \forall\, b \in \mathcal{N},\ b \leq o \leq n,\ i \in \mathcal{E} \tag{4.8}$$

Another important issue is to avoid sub-routes on picking-routes. Thus, we included the next constraints' set, based on Miller-Tucker-Zemlin (MTZ) (MILLER; TUCKER; ZEMLIN, 1960) formulation of the traveling salesman problem (TSP):

$$u_{jb} \geq u_{ib} + 1 - |\mathcal{E}|(1 - z_{ijb}) \; \forall b \in \mathcal{N}, i \neq j \in \mathcal{E} \setminus \{0\} \tag{4.9}$$

The following constraints aim to reduce symmetric solutions and establish a direction to the picker. Observe that, for each batch, the same picking-route can be traveled in two different directions; hence we can reduce the search space by enforcing feasible solutions to only use one of those directions. For that, we define a sorting function $\sigma : \mathcal{E} \to \mathbb{N}$ which associates a different natural index to each address, being 0 the depot's index (i.e., $\sigma(d) = 0$). Since all picking-routes start at the depot, our proposal is to follow the direction where the second address will have a lower σ value:

$$\sum_{i \in \mathcal{E} \setminus \{d\}} z_{dib}\sigma(i) \leq \sum_{i \in \mathcal{E} \setminus \{d\}} z_{idb}\sigma(i) \quad \forall \, b \in \mathcal{N} \tag{4.10}$$

Finally, we focus on breaking batch symmetries. Note that exchanging the total content from a batches' pair does not affect the objective value, and if there are empty batches, then equivalent solutions are obtained by swapping any used batch with an empty one. Thus, the following constraints' set enforces the usage of the first batches, guaranteeing that empty batches will be the last ones of a feasible solution:

$$\sum_{o=b}^{n} x_{ob} n \geq \sum_{o=b+1}^{n} x_{o(b+1)} \quad \forall \, 1 \leq b \leq n-1 \tag{4.11}$$

The above discussion lead us to the following model for JOBPRP:

Model 1 *Integer linear program for instance* $\langle n, c, \mathcal{E}, \omega_o, \phi_o, \delta \rangle$ *of* JOBPRP*:*

$$\min \sum_{b=1}^{n} \sum_{i \in \mathcal{E}} \sum_{\substack{j \in \mathcal{E} \\ j \neq i}} \delta(i,j) z_{ijb}$$

$$s.t. \; constraints \; (4.1) - (4.11)$$

$$x_{ob}, \; z_{ijb} \in \{0,1\}, u_{ib} \in \mathbb{N}$$

5 A DP-BASED HEURISTIC APPROACH

Our heuristic approach is named "2-level-DP-JOBPRP" (Algorithm 1). It works in two DP phases: first, the algorithm groups the orders into batches with a knapsack-based (KARP, 1972) strategy. Then, we filter the addresses that should be visited for each batch and compute the respective picking-route. The lengths' sum of all picking-routes is the problem objective value. The idea behind this approach is to reduce the number of batches (with the knapsack strategy, the batches are filled to the possible fullest), which implies fewer visits to the depot to unload items, resulting in a total picking-route's length decreasing.

```
   Input  : an instance ⟨n, c, E, ω, φ, δ⟩ of JOBPRP
   Output : the solution S, which corresponds to the batches' picking-routes
 1 begin
 2     B ← ∅;
 3     L ← ∅;
 4     S ← ∅;
 5     B ← generateInitialBatches(ω, c);
 6     for b ← 1 to |B| do
 7         L ← L ∪ {prepareLocations(n, E, φ, B_b)};
 8     end
 9     for b ← 1 to |B| do
10         S ← S ∪ {optDistance(L_b, δ)};
11     end
12     return S;
13 end
```

Algorithm 1: 2-level-DP-JOBPRP heuristic

In Algorithm 1, between lines 2 and 4, we initialize the vectors $\mathcal{B}$, $\mathcal{L}$ and $\mathcal{S}$. $\mathcal{B}$ represents the batches configuration, such that each array item stores the orders indexes which are included in the batch. Each position of $\mathcal{L}$ contains the addresses that should be visited in the correspondent batch picking-route, and $\mathcal{S}$ stores the problem solution, which is the batches' picking-routes.

At line 5, the batches are arranged, process described by Algorithm 2. Between lines 6 and 11, the following procedures are performed: at line 7, for each batch, the algorithm analyzes and filters the addresses required by the orders and appends them to $\mathcal{L}$. At line 10, the picking-routes are computed with the DP procedure explained in subsection 3.1.1 and are appended to $\mathcal{S}$. At line 12, we return the problem solution.

The generation of the initial batches' configuration is represented by Algorithm 2. It receives as input the weights of each order (ω) and the capacity limit of each batch (c), and returns the batches' arrangement $\mathcal{B}$. At line 2 the set $\mathcal{B}$ is initialized as empty, and

```
   Input  : ω, c
   Output : the batches configuration B
 1 begin
 2     B ← ∅;
 3     I ← indexes from ω;
 4     while |I| > 0 do
 5         X ← knapsack(ω, c);
 6         for x ∈ X do
 7             O ← O ∪ {I_x};
 8         end
 9         B ← B ∪ {O};
10         for x ∈ X do
11             ω ← ω \ {ω_x};
12             I ← I \ {I_x};
13         end
14         O ← ∅;
15     end
16     return B;
17 end
```

Algorithm 2: generateInitialBatches procedure

at line 3, $\mathcal{I}$ receives the indexes of the orders' weights vector (which corresponds to an identification to each order).

Between lines 4 and 15, while there are non-allocated orders, the following procedures are executed: at line 5, an algorithm based on knapsack (depicted by Algorithm 3) is performed to determine which orders from those available (the orders which there are still in ω) should be included in the current batch. The indexes of the chosen orders are stored in $\mathcal{X}$. At line 7 we append the selected indexes from $\mathcal{I}$ to $\mathcal{O}$, the set of orders, which is appended to $\mathcal{B}$ at line 9. At lines 11 and 12 we update ω and $\mathcal{I}$, respectively. Latter, we reset $\mathcal{O}$ to empty at line 14. The batches arrangement $\mathcal{B}$ is returned at line 16.

The Algorithm 3 always receives as input an updated vector ω and the fixed capacity limit c. This algorithm returns a vector $\mathcal{X}$, containing the indexes of the current ω which maximizes the total weight of the batch without exceed the capacity limit c. At line 2, the indexes' set is declared as empty, and at line 3, a table M with dimensions $|\omega|$ and c (starting the indexing from 0) is initialized and filled with the value 0. The position $M[i][j]$ from the table informs the maximum weight that the batch can hold, case the orders from 1 to i are included or not in the batch, without exceed the capacity limit j.

Between lines 4 and 12 we iteratively fill the table M from bottom to top. If the weight of the order is greater than j (line 6), this order is not included to the batch; in another case, we decide if the order is included or not (line 9). From line 13 to line 23 we just trace-back the table M to determine which are the selected orders. Their correspondent ω indexes are appended to $\mathcal{X}$, which is returned to Algorithm 2 at line 24.

```
   Input  : ω, c
   Output : the set 𝒳 of ω's indexes, which should be included in the batch
 1 begin
 2     𝒳 ← ∅;
 3     M ← 0^((|ω|+1)×(c+1));
 4     for i ← 1 to |ω| do
 5         for j ← 1 to c do
 6             if ω[i − 1] > j then
 7                 M[i][j] ← M[i − 1][j];
 8             else
 9                 M[i][j] = max(M[i − 1][j], ω[i − 1] + M[i − 1][j − ω[i − 1]]);
10             end
11         end
12     end
13     p ← |ω|;
14     q ← c;
15     while p > 0 and q > 0 do
16         if M[p][q] = M[p − 1][q] then
17             p ← p − 1;
18         else
19             𝒳 ← 𝒳 ∪ {p − 1};
20             p ← p − 1;
21             q ← q − ω[p];
22         end
23     end
24     return 𝒳;
25 end
```

Algorithm 3: knapsack procedure

The highlight of the 2-level-DP-JOBPRP heuristic is the fast execution time, which for the large instances tested, does not exceed two-hundredths of a second. Although the objective values delivered by this heuristic are higher than those provided by the literature, they are consistent and follow a pattern according to the results already reported (more details in the chapter 7). Thus, we present a new metaheuristic in the next section, which aims to refine the objective values found when solving JOBPRP.

6 A GGA WITH GENE TRANSMISSION FOR JOBPRP

Genetic algorithms are evolution-based algorithm design strategies that aim to optimize complex combinatorial problems. Many of them are directly inspired by the evolutionary process of the species of living beings. The main idea of these algorithms is to encode a possible solution to a particular problem on a representation known as a chromosome. Operations are realized over these structures aiming to optimize the objective function of the target problem until a stop criterion is reached (ROZENBERG; BÄCK; KOK, 2012).

Focusing on nature-inspired approaches, the fundamental steps of a genetic algorithm are the generation of an initial population of solutions, the evaluation, the crossover, the mutation, and the population replacement. The operations executed over these procedures can be summarized as follows (WHITLEY, 1994):

- **Generation of an initial population:** generally, the construction of a determined set of solutions occurs randomly or using a heuristic strategy, composing the initial population of individuals;
- **Evaluation:** the solutions are evaluated according to their fitness, i.e., how much these solutions satisfies the problem. It is frequent to associate the fitness to the objective function value;
- **Crossover:** usually, the parents are randomly paired to cross and generate a child, inheriting both parents' genetic characteristics. The set of children composes the offspring;
- **Mutation:** all individuals have a chance to suffer a mutation. A mutation represents an external factor from a crossover that modifies the individual. This step is essential to ensure genetic variability and reduce the probability of local optimum solutions;
- **Population replacement:** a new population is generated from the original solutions and the offspring.

During all the processes, the individual with the best fitness is appointed as a possible solution to the problem. In most cases, the stop criteria are given by a certain number of generations, several iterations without a solution update, or a program execution time.

The following section discusses possible ways to represent a solution, introducing the idea of grouped genetic algorithms.

6.1 Grouped Genetic Algorithms

The chromosomes' encoding scheme significantly influences the genetic algorithm design and performance. The object-oriented genetic algorithm (OO-GA) and the grouped genetic algorithm (GGA) are the most common representations. Below, the pros and cons of these encodings are analyzed.

To explain the differences between OO-GA and GGA, we contextualize with the OBP. Given a set with 8 orders, we need to assign them to batches, such that the problem's constraints are satisfied. A possible manner of assembling a chromosome for this problem is declaring a vector with exactly 8 positions, each of them containing an integer. The vector's positions represent the orders, and the respective integer values indicate the batches to which the orders are assigned. In Figure 6.1, we depict two possible solutions encoded with this schema.

Figure 6.1 – Example of object-oriented scheme chromosomes. Source: the author.

In Figure 6.1, at solution A, orders 1 and 2 are assigned to batch 1, order 3 is assigned to batch 2, orders 4, 5 and 6 are assigned to batch 3, order 7 is assigned to batch 4 and order 8 is assigned to batch 5. At solution B, order 1 is assigned to batch 1, orders 2 and 3 are assigned to batch 2, orders 4 and 5 are assigned to batch 3 and orders 6, 7 and 8 are assigned to batch 4.

Note that even though solution A splits the orders into 5 batches and solution B splits the orders into 4 batches, the solution size keeps fixed to 8, the number of orders. The orders are the "objects" that we are manipulating and assigning to other entities (the batches) - this structure "one gene per item" is the most notable feature of OO-GA (HOLLAND, 1992).

This kind of schema has advantages such as faster implementation and easier application of the genetic operators over the chromosomes. On the other hand, it carries disadvantages as redundant solutions (FALKENAUER, 1996). To exemplify the concept

of redundancy, let's suppose a chromosome A' that is feasible to the OBP cited above: $A' = \{2\ 2\ 3\ 4\ 4\ 4\ 5\ 1\}$. In this case, solutions A and A' are redundant because despite their different phenotype (how a solution is visualized, by itself), they have the same meaning, which is the two first orders joined at some batch (the name or label from batch, do not matter), the third order in a different batch, orders 4, 5 and 6 together in another separate batch, and orders 7 and 8 in their own batches.

The problem of redundant solutions is that they increase exponentially with the problem input size, which implies an increase in total computation time and a higher probability of falling in local optimal due to the expansion of the search space. An alternative to such an issue is to design another chromosome's representation. Figure 6.2 illustrates equivalent solutions to A and B in a GGA scheme.

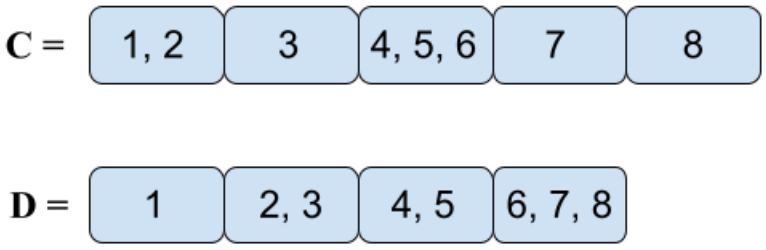

Figure 6.2 – Example of grouped genetic algorithm's chromosomes. Source: the author.

Figure 6.2 shows that C and D represent the same result of A (or A') and B, respectively. Each gene is treated as a group that contains minor elements, and it justifies why this schema fits so well to grouping problems, such as the OBP. The main advantage of GGA is the exclusion of redundant solutions, resulting in faster executions for larger inputs (FALKENAUER, 1994).

The inconveniences of GGA are the genetic operators that demand special attention once the chromosome's size is not fixed, and we have no guarantee about the distribution of the objects over the groups. Then, procedures such as crossover or mutation may cause problems like object replication in the solution or improper assignments in the groups. Attempts to avoid these problems could require more processing time than OO-GA. Thus, it is crucial to evaluate the trade-off due to the chromosomes' encoding schema and the input size over the project.

As we deal with large instances in this work, we opted for GGA. One of the possible techniques to crossover the solutions in this schema is the controlled gene transmission, which consists of selecting the best genes from both parents for the resulting

child based on an efficiency criterion specific to each problem. During the presentation of our approach, we explain how we classify the genes as inheritable or not.

6.2 Main approach

Our metaheuristic approach to JOBPRP is called "Grouping Genetic Algorithm with Controlled Gene Transmission for Joint Order Batching and Picking Routing Problem" (GGACGT-JOBPRP). We choose the grouping genetic algorithm (GGA) schema instead of an object-oriented GA (OO-GA) because Falkenauer demonstrated several advantages from GGA over OO-GA on Bin Packing Problem (BPP) (FALKENAUER, 1996). Notice that the BPP seeks to minimize the number of bins (batches) used to package a set of items (orders) with different weights, such that the weight's limit of bins (maximum pickers capacity) is not exceeded. Consequently, BPP is the particular case of JOBPRP when all distances are the same.

A remarkable feature of GGA is the chromosome, which by definition is a set of groups that contains minor elements without repetition (the solution is a partition set). This representation implies in exclusion of symmetric solutions, allowing a faster convergence to the expected objective value (FALKENAUER, 1996). Focusing on JOBPRP, the solution's groups (i.e., the chromosomes' genes) can be interpreted as the batches, and their contents are the orders. Also, each solution's batch has an associated picking-route.

Thus, considering that a chromosome encodes the orders distribution into batches, we summarize the main algorithm's pseudo-code at Algorithm 4, which receives an instance $\langle n, c, \mathcal{E}, \omega, \phi, \delta \rangle$ of JOBPRP and returns the best chromosome found β at line 30.

An empty population $\mathcal{P}$ of individuals is initialized at line 2. This population is filled with POP_SIZE members through a heuristic procedure (line 4) that is further detailed by Algorithm 5. The fitness evaluation of the new individuals is performed during the generation process. The fitness value is equal to the objective value, which is equivalent to the sum of picking-route's distances. These distances are optimal, and are calculated through a dynamic programming approach proposed in (RATLIFF; ROSENTHAL, 1983), explained in subsection 3.1.1. To avoid recalculations, we store the distances traveled in each batch every time we evaluate the fitness because this information is required at crossover. After the initialization process, the most adapted individual from $\mathcal{P}$ is assigned to β (line 7), the best solution found, which is updated whenever a better solution

is found between lines 14-17 and lines 23-25.

```
    Input  : an instance ⟨n, c, E, ω, φ, δ⟩ of JOBPRP
    Output : β, best chromosome found
 1  begin
 2      P ← ∅;
 3      for i ← 1 to POP_SIZE do
 4          C ← generateIndividual(n, c, E, ω, φ, δ);
 5          P ← P ∪ {C};
 6      end
 7      β ← most adapted individual from P, considering F(β);
 8      for t ← 1 to ITERATION_LIMIT do
 9          G ← selectParents(P, PARENTS_SIZE);
10          Z ← crossover(G, n, ω, c);
11          for each x ∈ Z do
12              F(x) ← x's fitness;
13          end
14          β_z ← best solution of Z ;
15          if F(β_z) < F(β) then
16              β ← β_z;
17          end
18          for x ∈ Z \ {β_z} do
19              m ← mutation(x, ω);
20              if m is valid then
21                  update x's chromosome;
22                  F(x) ← x's fitness;
23                  if F(x) < F(β) then
24                      β ← x;
25                  end
26              end
27          end
28          P ← best POP_SIZE solutions from P ∪ Z;
29      end
30      return β;
31  end
```

Algorithm 4: Main GGACGT-JOBPRP approach

Between lines 8 and 29 the algorithm executes ITERATION_LIMIT times the procedures that are described next. First, PARENTS_SIZE individuals from $\mathcal{P}$ are selected to be the parents $\mathcal{G}$ (line 9) of the next generation. These parents are chosen with a discrete probability distribution proportional to the ranking provided by the fitness. The resulting offspring from the parents' crossover is assigned to the set $\mathcal{Z}$ (line 10), which is evaluated between lines 11 and 13. To ensure variability without losing genetic characteristics over generations, all individuals, excepting the best individual from $\mathcal{Z}$, can be mutated (line 19). The mutation process involves exchanging a pair of orders from different random batches, which may cause a batch overload (phenomena defined as **invalid mutation**). Aiming feasible solutions, if an invalid mutation is detected, it is discarded (line 20). Finally, at line 28 we define the new population as the best POP_SIZE solutions from the

offspring $\mathcal{Z}$ and the current population $\mathcal{P}$.

In the sections 6.3 and 6.4, we detail, respectively the **generateIndividual** (generation of new individuals) and **crossover** procedures.

6.3 Individual generation

The individual generation heuristic described by Algorithm 5, aims to provide an initial population with lower objective function values. Therefore, its main proposal is to group the most similar orders into the same batches, decreasing the associated picking-route. Hence, given a set of orders, we should analyze their content (i.e., which products are required in each order), and from a random order, to define which orders may be combined according to their products' locations.

To exemplify, let's suppose that a picker travels 210 length units (LU) to collect products from an order "A", and the same picker travels 210 LU to collect products from an order "B". If the total distance traveled by the picker to retrieve the items from these orders stills 210 LU when these orders are assigned to the same batch (without exceeding its capacity limit), it means that we have a perfect match between these two orders in the batch.

Input : an instance $\langle n, c, \mathcal{E}, \omega, \phi, \delta \rangle$ of JOBPRP
Output : a new individual $\mathcal{I}$
begin
  $\mathcal{I} \leftarrow \emptyset$;
  $\mathcal{O} \leftarrow \{1, 2, \ldots, n\}$;
  $i \leftarrow 0$;
  while $|\mathcal{O}| > 0$ **do**
    $b \leftarrow$ new batch with index i;
    $r \leftarrow$ remove random order from $\mathcal{O}$;
    $b \leftarrow b \cup \{r\}$;
    $w \leftarrow$ total weight of b;
    while *some order of* $\mathcal{O}$ *fits in* b **do**
      $s \leftarrow o \in \mathcal{O}$ s.t min. OSD(b, o, δ, ϕ, $\mathcal{E}$), $w + \omega_o \leq c$;
      if $s \neq \emptyset$ **then**
        $\mathcal{O} \leftarrow \mathcal{O} \setminus \{s\}$;
        $b \leftarrow b \cup \{s\}$;
        $w \leftarrow$ total weight of b;
      end
    end
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{b\}$;
    $i \leftarrow i + 1$;
  end
  return $\mathcal{I}$;
end

Algorithm 5: Individual generation algorithm

The algorithm receives as input an instance $\langle n, c, \mathcal{E}, \omega, \phi, \delta \rangle$ of JOBPRP, and returns a new feasible solution $\mathcal{I}$. It starts creating an empty individual $\mathcal{I}$, and a set $\mathcal{O}$, that contains the orders (numbered from 1 to n) which are not allocated in solution (lines 2 and 3). At line 4, we assign the value 0 to the variable i, which represents the current batch that we are accessing.

Between lines 5 and 20, we execute the main loop of the algorithm until all orders are included in some batch. At each iteration, we initialize a new batch b with index i (line 6) and choose a random order r to be removed from $\mathcal{O}$ and be pushed to b (lines 7 and 8). The total weight of b's orders is stored in the variable w (line 9). While some order from $\mathcal{O}$ fits in b (line 10), we assign to s the order o from $\mathcal{O}$ which minimizes OSD (Orders similarity degree) function without exceed the capacity limit c (line 11). If no order satisfies the load constraint, then s becomes empty. If s is not empty (line 12), then we remove it from $\mathcal{O}$, push it into b (lines 13 and 14) and update w at line 15. When the batch construction is concluded, the batch b is appended to solution (line 18).

The insight of using an OSD function is to determine which order $o \in \mathcal{O}$ generates the minimum distance for the picking-route when combined with the orders already in b. Hence, the OSD of an order $o \in \mathcal{O}$ consists of applying the dynamic programming algorithm of (RATLIFF; ROSENTHAL, 1983) to compute an optimal picking-route for retrieving all products of the orders in $b \cup \{o\}$.

6.4 Crossover procedure

The crossover algorithm applies a controlled gene transmission technique, that provides gains over other crossover methods (QUIROZ-CASTELLANOS et al., 2015). At the end of this process, the batches with more orders and shorter picking distances come first at the solution, which implies lower objective values.

Our crossover procedure is described by Algorithm 6 and applies the ideas proposed for BPP in (QUIROZ-CASTELLANOS et al., 2015) to JOBPRP. The algorithm receives as input a set of parents $\mathcal{G}$, the number of orders n, the weight's vector ω, and the maximum pickers' capacity c, returning the offspring $\mathcal{O}$ (a vector of individuals). The main loop of this algorithm (lines 3-30) evaluates if there are parents to be crossed. If so, at line 4, we define an empty set $\mathcal{L}$ that stores the orders which are not in any batch ($\mathcal{L}$ is also known as free list). At line 5, we declare two individuals: $\mathcal{T}$, that we call by "tape" (which is the crude product from the parents' crossover) and $\mathcal{C}$, that is the child itself. At

line 6, we choose two different parents from $\mathcal{G}$ to perform the crossover an remove them from $\mathcal{G}$ (line 7).

```
   Input  : 𝒢, n, ω, c
   Output : an offspring 𝒪
 1 begin
 2     𝒪 ← ∅;
 3     while |𝒢| > 0 do
 4         ℒ ← ∅;
 5         𝒯, 𝒞 ← ∅, ∅;
 6         𝒢₁, 𝒢₂ ← random individuals from 𝒢;
 7         𝒢 ← 𝒢 \ {𝒢₁, 𝒢₂};
 8         sort 𝒢₁ and 𝒢₂ batches' descending by number of orders;
 9         for i ← 0 to min(|𝒢₁|,|𝒢₂|) - 1 do
10             if 𝒟(𝒢₁) ≤ 𝒟(𝒢₂) then
11                 𝒯 ← 𝒯 ∪ {𝒢₁[i]} ∪ {𝒢₂[i]}
12             else
13                 𝒯 ← 𝒯 ∪ {𝒢₂[i]} ∪ {𝒢₁[i]}
14             end
15         end
16         if |𝒢₁| ≠ |𝒢₂| then
17             append sequentially the remaining batches to 𝒯;
18         end
19         for each g ∈ 𝒯 do
20             if all g's orders are not in 𝒞 then
21                 𝒞 ← 𝒞 ∪ {g};
22             else
23                 append non-repeated orders from g to ℒ;
24             end
25         end
26         for each o ∈ ℒ do
27             insert o in some 𝒞's batch with first-fit heuristic;
28         end
29         𝒪 ← 𝒪 ∪ {𝒞};
30     end
31     return 𝒪;
32 end
```

Algorithm 6: Crossover procedure

As we are applying a controlled gene transmission technique, we select the best genes (batches) from both parents at each crossover, ensuring a high-quality child in terms of fitness. To implement this, at line 8, both $\mathcal{G}_1$ and $\mathcal{G}_2$ solutions' batches are sorted decreasingly according to their number of orders (i.e., batches with more orders come first on the sorting).

Considering $\mathcal{D}$ as the picking-route distance associated to a batch (computed during the fitness evaluation from Algorithm 4), we construct $\mathcal{T}$ as follows: from line 9 to line 15 we compare batch-per-batch from $\mathcal{G}_1$ and $\mathcal{G}_2$, pushing first to $\mathcal{T}$ the batch with the lower distance, then the second lower and so on. If the parents have different sizes, we append to $\mathcal{T}$ the genes from the larger individual, which were not submitted to the

previous comparison (line 17).

Note that by analyzing $\mathcal{T}$ at the end of its construction, we may verify the occurrence of repeated orders over the batches. Hence, we execute a filter step that consists of traversing $\mathcal{T}$ (line 19) while checking if all the orders from the batch $g \in \mathcal{T}$ are not in the child $\mathcal{C}$ (line 20). If this is the case, the batch is appended to the child $\mathcal{C}$ (line 21), otherwise g is not appended to $\mathcal{C}$ and the non-repeated orders from g are pushed to the free list $\mathcal{L}$ (line 23). At the end of the filtering step, we insert each order o from free list $\mathcal{L}$ in some batch of $\mathcal{C}$ using a first-fit heuristic (line 29).

To exemplify the crossover procedure, let's say that the two individuals C and D from Figure 6.2 are crossing. The first step is to sort C and D according to the number of orders which are inside each batch (Figure 6.3):

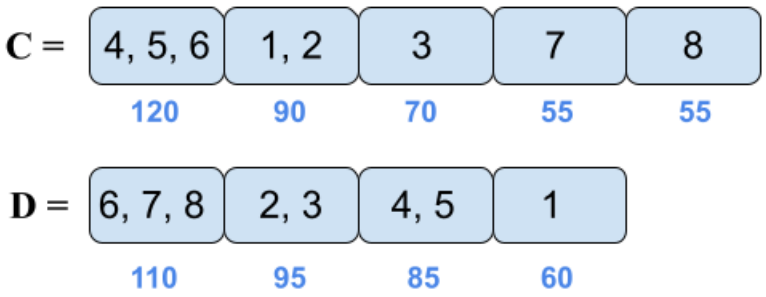

Figure 6.3 – Individuals C and D sorted by orders. Source: the author.

Note that each batch has a correspondent blue number below it in Figure 6.3. These numbers represent the batches picking-route distances, which were calculated during the fitness evaluation. The next step is to create an auxiliary individual T, which will receive the batches from C and D. We compare both C and D, batch-per-batch, appending first to T the batch which has the shortest associated distance, as illustrated in Figure 6.4:

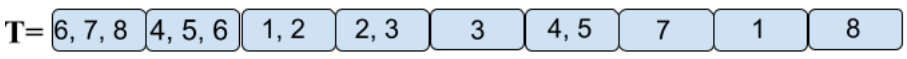

Figure 6.4 – Crude product from parent's crossover. Source: the author.

Finalizing this mixing step from the crossover, some orders can appear in more than one batch, which is why we effectuate a filtering step. It consists in exploring the solution, batch-per-batch, highlighting an order if it is already in the chromosome (Figure 6.5). The batches which contain highlighted elements are excluded from the solution.

The not highlighted orders from some excluded batches are appended in a “free list” (i.e., a list of non-allocated orders). At the end of the filtering step, we reallocate orders from the free list with a first-fit heuristic (following the batches’ sequence, we introduce a free order to the first batch with enough capacity limit).

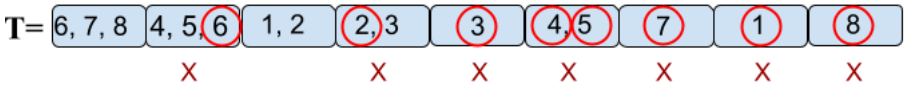

Figure 6.5 – Repeated orders. Source: the author.

In this chapter, we briefly review important features of genetic algorithms and possible solution encoding schemes. In addition, we present a group genetic algorithm for JOBPRP, which performs the generation of individuals with a heuristic based on the degree of order similarity, and performs the crossover procedure with a controlled gene transmission technique. In the next chapter, we discuss about the experimental results.

7 EXPERIMENTS AND RESULTS

To validate our metaheuristic, we conducted several experiments of our GGACGT-JOBPRP approach over Henn's set of instances (HENN; WäSCHER, 2012), provided by (MENéNDEZ et al., 2017). The same dataset was used to evaluate 2-level-DP-JOBPRP heuristic. Also, we adapted a subset of these instances to confront our results with those provided by a commercial ILP solver running Model 1. Our implementations, complete and detailed results, instances, outputs, and documentation can be found at the Github (LORENCI; RAVELO, 2022) repository.

In the rest of this chapter, we present our computational environment, the datasets, the metaheuristic parameters and analyze the summarized results.

7.1 Computational environment

The algorithms were implemented in `C++` language using the compiler `g++` version 9.3.0. The support scripts (instance treatment, tests automatizing, and input/output control) were written in `Python` version 3.8.10. The tests were single-core simultaneously executed in an AMD Ryzen 9 3900X 12-Core Processor (3.8GHz) with 32 GB RAM and Ubuntu 20.04 LTS operational system. We implemented and solved Model 1 with the mathematical optimization solver Gurobi 9.0.3.

To generate pseudo-random numbers we used the Mersenne Twister algorithm (SAITO; MATSUMOTO, 1993). Each instance was tested 30 times on the metaheuristic, with integer seeds from [1,30] interval.

7.2 Instances

Henn's set of instances is the most cited datasets over warehouse problems studies that we found. Their public files are compound by 64 instances, equally divided into four groups: ABC1, ABC2 (most requested items closer from depot), and RAN1, RAN2 (items are randomly scattered on warehouse).

We choose to work with this dataset due to its popularity and conciseness to single-block warehouse layout. Another decisive fact was the linked results: some papers present interesting jobs but don't show granular results, which harms a possible data comparison.

Henn's dataset has solid documented results, including those from (MENéNDEZ et al., 2017), that to the best of our knowledge are the best for OBP, both in execution time and objective function. We remember that even our approach aims to solve JOBPRP, it is possible to compare its yielded objective values with those produced by OBP, since the difference is that we are also optimizing each of the picking-routes. Still, the problem essence is the same.

Each instance has the same warehouse layout configuration: a single-block warehouse (two cross-aisles, one at bottom another on the top) with 10 parallel aisles and 45 picking positions, resulting in 450 addresses plus depot. Each aisle has products on both sides, right and left. Each picking position has 1 length unit (LU), and when a picker leaves or enters an aisle, it travels 1 LU more. Also, the horizontal distance from one aisle to another is 5 LU. The depot is located on the inferior cross-aisle, in front of aisle 0.

These instances are shaped by combining 40, 60, 80, and 100 orders and 30, 45, 60, and 75 as possible maximum pickers' capacities. We used this dataset as input to check the overall performance of GGACGT-JOBPRP and confront the results with the literature. From the same dataset, we randomly selected one instance from each group to be modified to a smaller instance, which we truncated to the 10 first orders and fixed the capacity limit to 30. These smaller instances were used to test our formulation and assess the quality of our algorithm solutions.

7.3 Parameters

We used the iRace package (LóPEZ-IBáñEZ et al., 2016) to determine POP_SIZE and PARENTS_SIZE GGACGT-JOBPRP's parameters. This tool automatically detects the best parameter configuration over a parameters domain, focusing on stability and solution quality.

The parameters domain is defined as follows:

- POP_SIZE is an integer, multiple of 10, belonging to the interval defined from 40 (the minimum possible number of orders) to 400 (four times the maximum possible number of orders);
- PARENTS_SIZE is an integer, multiple of 10, and its possible values are conditional to POP_SIZE, belonging to the interval defined from 10 to POP_SIZE$/2+10$.

The iRace package provided us the values 340 for POP_SIZE and 120 for PAR-

ENTS_SIZE. The ITERATION_LIMIT parameter was obtained through controlled tests. These isolated experiments aimed to find an adequate value to this parameter so that it was possible to analyze the objective function value progression over the generations. Such value was fixed to 500. The solver execution time limit was set to 3600 s. We emphasize that since the crossover always happens in pairs, the offspring size is PARENTS_SIZE$/2$.

7.4 Results and analysis

To assess GGACGT-JOBPRP, we conducted experiments over the smaller instance group. The solver just interrupted the process at the time limit (3600 s) in all the tests. The results of these experiments are shown in Table 7.1, where: NA represents the number of visited addresses during the picking-route (number of different products plus depot); LB is the lower bound; the column GAP ($\%$) within Gurobi column, is the objective GAP; OBJ and TIME represents the objective values and the execution time (in seconds); and GAP ($\%$) within GGACGT-JOBPRP column, represents the relative difference between our metaheuristic objective value and solver LB (i.e., to simplify notation, considering our approach as G, $\frac{OBJ_G - LB}{OBJ_G} \cdot 100$). Our results are promising, considering the relation between the solution quality (lower than the best feasible solution provided by the solver) and the execution time, which is approximately 60 times smaller than Gurobi.

Table 7.1 – Comparative between GGACGT-JOBPRP and Gurobi

Instance	**Gurobi**					**GGACGT-JOBPRP**		
	NA	**LB**	**OBJ**	**TIME**	**GAP($\%$)**	**OBJ**	**TIME**	**GAP (%)**
1	101	1577	1892	3600	16.64	**1794**	74.48	12.09
2	112	1832	2070	3600	11.49	**1990**	60.96	7.93
3	131	2258	2620	3600	13.81	**2500**	70.60	9.68
4	138	2478	3038	3600	18.43	**2918**	89.84	15.07

The heuristic 2-level-DP-JOBPRP runs very fast. However, the delivered objective values were inferior from the state-of-art works from the literature. Table 7.2 summarizes the experimental results, where the first column indicates the instance name. Instances names indicates the set (ABC1, ABC2, RAN1, and RAN2), the warehouse identification, the number of orders, and the capacity limit. The results are organized in three columns, where MS-VNS represents results from (MENéNDEZ et al., 2017), 2-level-JOBPRP contains the heuristic results, and GGACGT-JOBPRP contains the metaheuristic experimen-

tal data.

Columns OBJ and TIME represents the objective value and the execution time (in seconds), respectively. Column RD (%) express the relative difference between the objective value of our approach $X \in \{$2-level-DP-JOBPRP, GGACGT-JOBPRP$\}$ and MS-VNS's objective value (i.e., $\frac{OBJ_{MS-VNS} - OBJ_X}{OBJ_{MS-VNS}} \cdot 100$). Despite not returning the best solutions, the 2-level-DP-JOBPRP demonstrated potential as a faster method to obtain fair upper bounds to the problem.

When compared with the approach of (MENéNDEZ et al., 2017) (identified as MS-VNS), the GGACGT-JOBPRP proved superiority on the solutions' quality. Over the 64 experiments on the dataset provided by (MENéNDEZ et al., 2017), our algorithm got a better average objective value in the 64 cases (100% of the experiments). The summary of these results can be found in Table 7.2, where SD is the standard deviation, and CV is the coefficient of variation (%) from our tests. According to Table 7.2, our GGACGT-JOBPRP presented high stability, with small coefficient of variations ($< 1.1\%$, average 0.5%) over all tested seeds for each instance.

The improvement of our GGACGT-JOBPRP over the MS-VNS represents 3.5% of the average solution value and can be visualized in the graphics from figures 7.1, 7.2, 7.3 and 7.4. These figures illustrate the objective function contrast between the algorithms' solutions for the instances from ABC1, ABC2, RAN1 and RAN2 groups, respectively.

Although the GGACGT-JOBPRP's average execution time (400.4 s) is more elevated than MS-VNS (43.8 s), the latter uses a faster S-Shape routing policy, avoiding solving the batches' picking-routes optimally and increasing the routes' lengths. We contact (MENéNDEZ et al., 2017) requesting for their MS-VNS implementation to reproduce the experiments in our CPU. They answered that they were seeking the correct code version, but we did not receive the executable until the publication date of this work. The information provided about their hardware description was that they implemented their algorithms in Java 6, and the experiments were executed on an Intel QuadCore with 2.5 GHz. Thus, we converted their computational time for a fair comparison with ours. In that direction, we used the values from CPUBenchmark (PASSMARK, 2022) for an Intel Core2 Quad Q9300 @ 2.50GHz, and the average execution time of the MS-VNS was reduced to 31.23 s.

The most expansive process of GGACGT-JOBPRP is the initial population generation, where the algorithm finds the best batches configuration according to the orders similarity degree. Another fact that impacted our approach execution time is the high fixed

Table 7.2 – Summary of experimental results

Instance	MS-VNS		2-level-DP-JOBPRP			GGACGT-JOBPRP				
	OBJ	TIME	OBJ	TIME	RD	OBJ	TIME	RD	SD	CV
abc1/29s-40-30	6914.0	3.4	7274.0	0.012	-5.2	**6542.1**	242.0	5.3	15.7	0.2
abc1/30s-40-45	4683.0	6.1	5046.0	0.008	-7.8	**4469.3**	183.6	4.5	29.6	0.6
abc1/31s-40-60	4255.0	9.6	4702.0	0.008	-10.5	**4143.4**	183.9	2.6	20.7	0.5
abc1/32s-40-75	3174.0	9.1	3536.0	0.005	-11.4	**3099.5**	135.9	2.3	24.0	0.7
abc1/37s-60-30	11234.0	6.9	11554.0	0.014	-2.8	**10667.8**	392.1	5.0	45.9	0.4
abc1/38s-60-45	7278.0	14.3	8002.0	0.013	-9.9	**7026.2**	312.6	3.4	32.1	0.4
abc1/39s-60-60	5727.0	34.5	6288.0	0.009	-9.8	**5608.1**	289.6	2.0	56.1	1.0
abc1/40s-60-75	3997.0	45.7	4738.0	0.008	-18.5	**3959.1**	251.8	0.9	30.1	0.7
abc1/61s-80-30	14654.0	15.2	15026.0	0.02	-2.5	**14169.8**	581.1	3.3	97.3	0.6
abc1/62s-80-45	10370.0	32.1	11164.0	0.013	-7.7	**9925.9**	489.4	4.2	34.7	0.3
abc1/63s-80-60	7301.0	57.5	8002.0	0.012	-9.6	**7205.0**	426.3	1.3	82.3	1.1
abc1/64s-80-75	6008.0	89.6	7024.0	0.011	-16.9	**5918.6**	427.3	1.4	41.3	0.6
abc1/9s-100-30	15637.0	29.7	16736.0	0.024	-7.0	**14794.8**	640.9	5.3	131.0	0.8
abc1/0s-100-45	10420.0	64.2	11976.0	0.025	-14.9	**10308.2**	631.2	1.0	51.1	0.4
abc1/1s-100-60	8449.0	116.6	9806.0	0.014	-16.1	**8327.0**	600.8	1.4	83.6	1.0
abc1/2s-100-75	7218.0	95.8	8414.0	0.013	-16.6	**7160.4**	597.2	0.7	54.7	0.7
abc2/x9l-40-30	7091.0	3.6	7584.0	0.011	-7.0	**6800.2**	247.2	4.1	67.8	0.9
abc2/10l-40-45	5393.0	8.4	5788.0	0.009	-7.3	**5163.9**	219.8	4.2	14.6	0.2
abc2/11l-40-60	3748.0	10.5	4244.0	0.008	-13.2	**3624.2**	166.9	3.3	30.8	0.8
abc2/12l-40-75	3128.0	13.2	3546.0	0.004	-13.4	**3061.6**	147.9	2.1	19.5	0.6
abc2/17l-60-30	10250.0	7.6	10330.0	0.014	-0.8	**9651.5**	376.1	5.8	64.6	0.6
abc2/18l-60-45	7509.0	16.7	8108.0	0.01	-8.0	**7195.0**	322.9	4.1	39.0	0.5
abc2/19l-60-60	5431.0	36.5	6092.0	0.008	-12.2	**5313.0**	287.6	2.1	42.2	0.7
abc2/20l-60-75	4207.0	33.6	4872.0	0.007	-15.8	**4151.1**	256.2	1.3	27.6	0.6
abc2/45l-80-30	13428.0	14.6	14260.0	0.018	-6.2	**12920.4**	520.5	3.7	85.1	0.6
abc2/46l-80-45	8735.0	33.5	9812.0	0.014	-12.3	**8388.6**	449.2	3.9	45.9	0.5
abc2/47l-80-60	7062.0	52.7	8234.0	0.01	-16.6	**6864.8**	436.4	2.7	46.9	0.6
abc2/48l-80-75	6048.0	93.2	6868.0	0.012	-13.6	**5977.3**	421.9	1.1	40.0	0.6
abc2/3l-100-30	17408.0	24.4	18258.0	0.029	-4.9	**16668.3**	682.7	4.2	93.6	0.5
abc2/4l-100-45	11005.0	65.4	12390.0	0.016	-12.6	**10602.2**	618.1	3.6	61.5	0.5
abc2/5l-100-60	9920.0	101.2	11140.0	0.014	-12.3	**9676.0**	638.2	2.4	68.7	0.7
abc2/6l-100-75	7069.0	86.8	8376.0	0.015	-18.5	**6984.5**	599.6	1.1	50.7	0.7
ran1/29s-40-30	9569.0	3.3	9664.0	0.011	-1.0	**9099.8**	256.6	4.9	40.1	0.4
ran1/30s-40-45	6243.0	6.6	6718.0	0.007	-7.6	**5901.4**	181.2	5.4	22.3	0.3
ran1/31s-40-60	5631.0	8.8	5868.0	0.006	-4.2	**5400.4**	163.6	2.2	20.8	0.3
ran1/32s-40-75	4385.0	26.0	4652.0	0.007	-6.1	**4289.0**	155.5	1.1	40.7	0.9
ran1/37s-60-30	14960.0	7.6	15234.0	0.027	-1.8	**14417.0**	406.8	3.6	94.2	0.6
ran1/38s-60-45	9626.0	29.5	10106.0	0.013	-5.0	**9190.0**	312.1	4.4	109.9	1.1
ran1/39s-60-60	7760.0	48.2	8326.0	0.011	-7.3	**7453.5**	299.2	3.9	33.3	0.4
ran1/40s-60-75	5473.0	24.9	5934.0	0.01	-8.4	**5334.8**	241.5	2.5	36.6	0.6
ran1/61s-80-30	19677.0	17.2	19948.0	0.024	-1.4	**18654.0**	597.0	5.1	74.9	0.4
ran1/62s-80-45	14003.0	49.8	14800.0	0.019	-5.7	**13470.0**	479.0	3.8	122.8	0.9
ran1/63s-80-60	9784.0	102.0	10572.0	0.015	-8.1	**9487.4**	421.7	2.0	40.7	0.4
ran1/64s-80-75	8010.0	67.1	8662.0	0.015	-8.1	**7763.7**	411.1	1.0	41.8	0.5
ran1/9s-100-30	21127.0	29.6	22628.0	0.025	-7.1	**20285.0**	646.6	3.9	160.4	0.7
ran1/0s-100-45	14425.0	64.1	15506.0	0.019	-7.5	**13929.8**	631.6	3.4	65.7	0.4
ran1/1s-100-60	11417.0	167.6	12362.0	0.014	-8.3	**11230.0**	620.7	1.6	52.9	0.4
ran1/2s-100-75	9395.0	108.0	10270.0	0.016	-9.3	**9218.8**	600.7	1.8	50.7	0.5
ran2/x9l-40-30	10020.0	3.5	10480.0	0.01	-4.6	**9372.8**	256.7	6.4	76.7	0.8
ran2/10l-40-45	7357.0	7.1	7642.0	0.01	-3.9	**6877.8**	210.7	6.5	36.9	0.5
ran2/11l-40-60	4998.0	10.7	5366.0	0.005	-7.4	**4799.8**	155.3	3.9	24.2	0.5
ran2/12l-40-75	4028.0	15.7	4234.0	0.006	-5.1	**3921.8**	138.1	2.6	22.4	0.5
ran2/17l-60-30	14009.0	9.4	14368.0	0.021	-2.6	**13338.5**	393.3	4.7	106.3	0.7
ran2/18l-60-45	10275.0	28.9	10872.0	0.014	-5.8	**9890.8**	337.2	3.7	57.7	0.5
ran2/19l-60-60	7323.0	32.0	7994.0	0.009	-9.2	**7013.4**	277.9	4.2	46.4	0.6
ran2/20l-60-75	5541.0	24.7	6004.0	0.007	-8.4	**5359.4**	241.6	3.2	39.7	0.7
ran2/45l-80-30	17558.0	17.7	18368.0	0.035	-4.6	**16975.0**	507.1	3.3	55.9	0.3
ran2/46l-80-45	11879.0	67.1	12654.0	0.016	-6.5	**11516.3**	469.6	3.0	48.0	0.4
ran2/47l-80-60	9806.0	47.8	10456.0	0.012	-6.6	**9484.5**	425.8	3.2	49.8	0.5
ran2/48l-80-75	8076.0	87.9	8728.0	0.01	-8.1	**7883.2**	410.7	2.3	30.3	0.3
ran2/3l-100-30	23532.0	31.9	24578.0	0.021	-4.4	**22388.7**	693.2	4.8	152.9	0.6
ran2/4l-100-45	14423.0	94.4	15868.0	0.019	-10.0	**13944.4**	652.8	3.3	81.2	0.5
ran2/5l-100-60	13203.0	93.8	14360.0	0.017	-8.8	**12756.0**	639.0	3.3	44.3	0.3
ran2/6l-100-75	9553.0	238.0	10248.0	0.014	-7.3	**9479.8**	616.6	0.7	106.7	1.1
Average	9340.8	43.7	10041.5	0.013	-8.4	**9007.7**	400.4	3.1	56.4	0.5

iteration limit value, which we purposely defined to extract the objective convergence over the generations (as example, in the graphic in Figure 7.5, we detail the objective function decay in an experiment with seed 2 and the instance abc1/29s-40-30). The complete set of graphics (objective value over generations) can be found in Appendix A, where the gains from the genetic procedures over the initial heuristic solution become evident due to the curve decay for all the seeds.

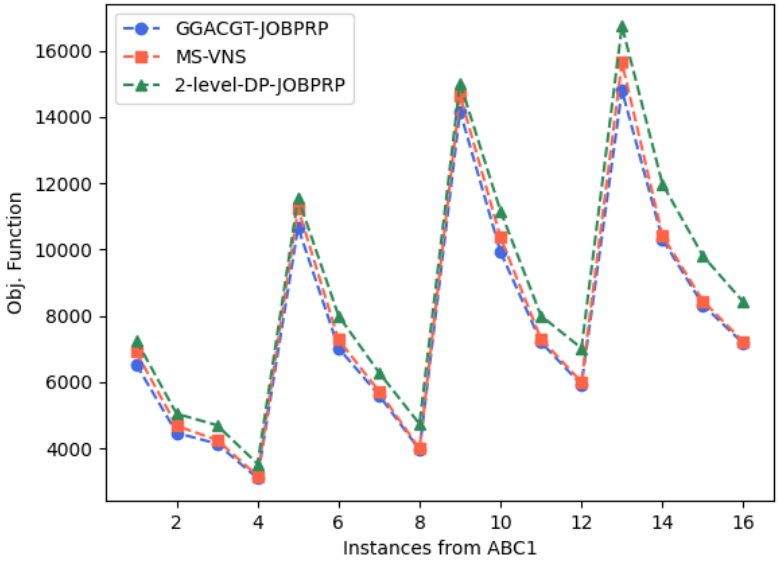

Figure 7.1 – ABC1 instances group - comparative with (MENéNDEZ et al., 2017)

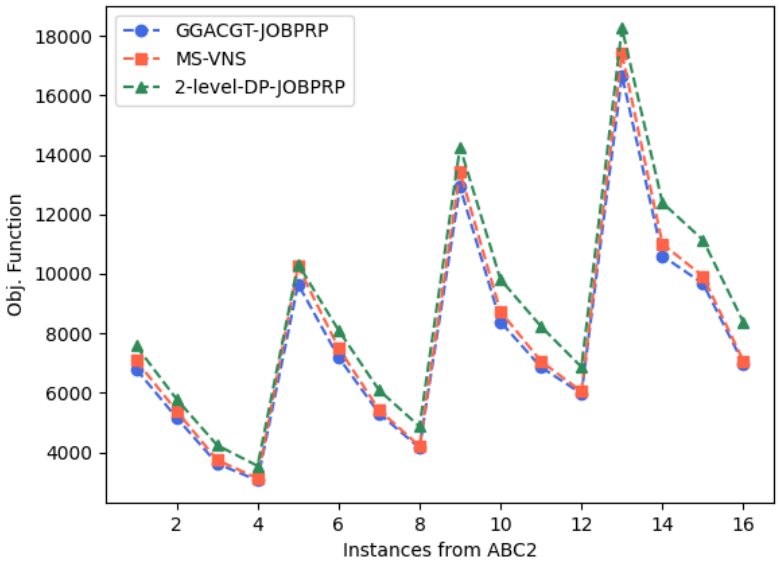

Figure 7.2 – ABC2 instances group - comparative with (MENéNDEZ et al., 2017)

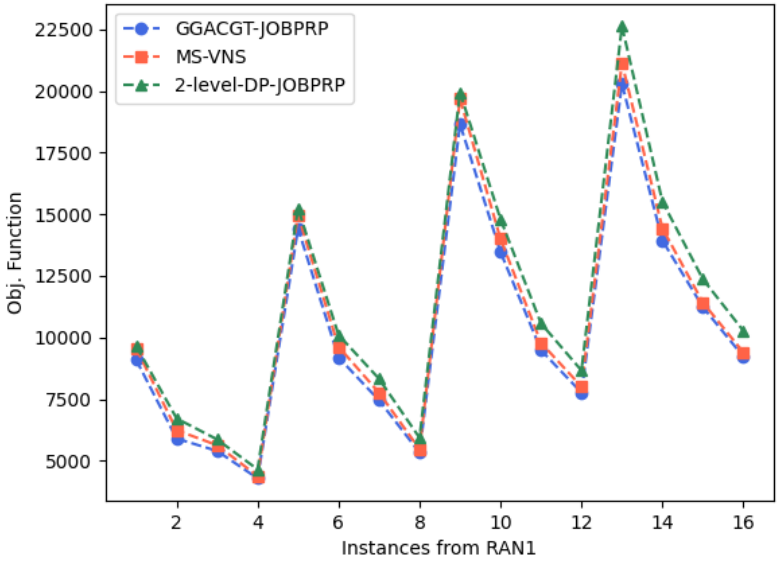

Figure 7.3 – RAN1 instances group - comparative with (MENéNDEZ et al., 2017)

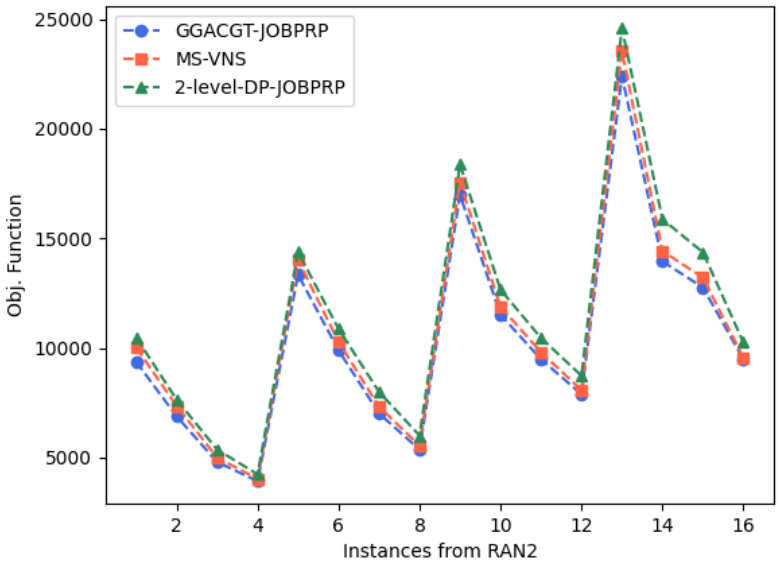

Figure 7.4 – RAN2 instances group - comparative with (MENéNDEZ et al., 2017)

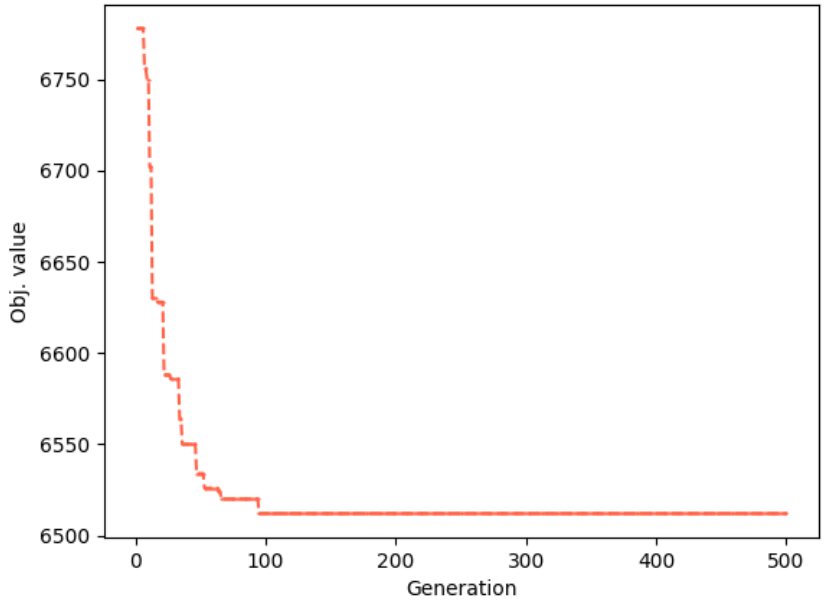

Figure 7.5 – Objective function from abc1/29s-40-30-0 instance over generations (seed 2)

8 CONCLUSIONS

In this work, we presented a novel ILP model, a heuristic (2-level-DP-JOBPRP), and a genetic algorithm (GGACGT-JOBPRP) for the JOBPRP. The 2-level-DP-JOBPRP applies first a knapsack-based DP procedure to generate an initial batches arrangement; another DP algorithm is performed in each batch to compute the traveled distance in the picking-routes. This heuristic proved to be a fast and straightforward method to generate feasible solutions to JOBPRP. These upper bounds can be improved with post optimization algorithms.

The GGACGT-JOBPRP constructs a chromosome's population using a heuristic that assigns the orders to the batches, providing initial solutions with lower objective values based on the order's similarity degree. Another differential of our approach is the crossover operator that adapts controlled gene transmission, generating high-quality new individuals.

We tested our GGACGT-JOBPRP over several instances from the literature, and our results were superior in solutions to the previously best-known approach, considering the objective values in 100% of the experiments. We also tested our mathematical formulation within a commercial solver over a small dataset, verifying the quality of our metaheuristic solutions for these smaller instances.

One of the major difficulties encountered during this work was to find clear datasets accompanied by granular and specific results. Thus, we also plan to construct and share new detailed datasets of instances in warehouse problems.

As future work, we intend to improve our mathematical model and present a new matheuristic to JOBPRP, hybridizing the crossover operator from GGACGT-JOBPRP. We are evaluating faster ways to generate the initial population, maintaining the robustness and the effectiveness of the orders similarity degree approach. Finally, we are planning to extend the operations for multi-block warehouses.

REFERENCES

AERTS, B.; CORNELISSENS, T.; SöRENSEN, K. The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem. **Computers & Operations Research**, v. 129, p. 105168, 2021. ISSN 0305-0548. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0305054820302859>.

BOYSEN, N.; de Koster, R.; WEIDINGER, F. Warehousing in the e-commerce era: A survey. **European Journal of Operational Research**, v. 277, n. 2, p. 396–411, 2019. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221718307185>.

BRIANT, O. et al. An efficient and general approach for the joint order batching and picker routing problem. **European Journal of Operational Research**, v. 285, n. 2, p. 497–512, 2020. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221720300977>.

CHENG, C.-Y. et al. Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. **International Journal of Production Economics**, v. 170, p. 805–814, 2015. ISSN 0925-5273. Decision models for the design, optimization and management of warehousing and material handling systems. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0925527315000894>.

DEPARTMENT, S. R. **Retail e-commerce revenue in the United States from 2017 to 2025**. 2022. <https://www.statista.com/statistics/272391/us-retail-e-commerce-sales-forecast/>. Accessed: 2022-01-04.

FALKENAUER, E. A new representation and operators for genetic algorithms applied to grouping problems. **Evolutionary Computation**, v. 2, n. 2, p. 123–144, 1994.

FALKENAUER, E. A hybrid grouping genetic algorithm for bin packing. **Journal of Heuristics**, v. 2, p. 5–30, 1996.

GADEMANN, N.; VELDE, V. D. S. Order batching to minimize total travel time in a parallel-aisle warehouse. **IIE Transactions**, Taylor & Francis, v. 37, n. 1, p. 63–75, 2005. Available from Internet: <https://doi.org/10.1080/07408170590516917>.

HENN, S.; WäSCHER, G. Tabu search heuristics for the order batching problem in manual order picking systems. **European Journal of Operational Research**, v. 222, n. 3, p. 484–494, 2012. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221712004389>.

HOFSTETTERL, S. **2021 E-Commerce Year In Review**. 2021. <https://www.forbes.com/sites/sarahhofstetter/2021/12/27/2021-e-commerce-year-in-review/>. Accessed: 2022-01-04.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136.

KARP, R. M. Reducibility among combinatorial problems. In: ____. **Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department**. Boston, MA: Springer US, 1972. p. 85–103. ISBN 978-1-4684-2001-2. Available from Internet: <https://doi.org/10.1007/978-1-4684-2001-2_9>.

KOCH, S.; WäSCHER, G. A grouping genetic algorithm for the order batching problem in distribution warehouses. **Journal of Business Economics volume**, v. 86, p. 131–153, 2016. Available from Internet: <https://link.springer.com/article/10.1007/s11573-015-0789-x>.

KULAK, O.; ŞAHIN, Y.; TANER, M. Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. **Flexible Services and Manufacturing Journal - FLEX SERV MANUF J**, v. 24, 03 2012.

LORENCI, F. F.; RAVELO, S. V. **JOBPRP implementation**. 2022. <https://github.com/lorencifelipe/JOBPRP>. Accessed: 2022-02-13.

LóPEZ-IBáñEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, v. 3, p. 43–58, 2016. ISSN 2214-7160. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2214716015300270>.

MENéNDEZ, B. et al. Variable neighborhood search strategies for the order batching problem. **Computers & Operations Research**, v. 78, p. 500–512, 2017. ISSN 0305-0548. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0305054816300168>.

MILLER, C. E.; TUCKER, A. W.; ZEMLIN, R. A. Integer programming formulation of traveling salesman problems. **J. ACM**, Association for Computing Machinery, New York, NY, USA, v. 7, n. 4, p. 326–329, oct 1960. ISSN 0004-5411. Available from Internet: <https://doi.org/10.1145/321043.321046>.

PANSART, L.; CATUSSE, N.; CAMBAZARD, H. Exact algorithms for the order picking problem. **Computers & Operations Research**, v. 100, p. 117–127, 2018. ISSN 0305-0548. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0305054818301862>.

PASSMARK. **AMD Ryzen 9 3900X vs Intel Core2 Quad Q9300 @ 2.50GHz**. 2022. <https://www.cpubenchmark.net/compare/AMD-Ryzen-9-3900X-vs-Intel-Core2-Quad-Q9300/3493vs1044>. Accessed: 2022-02-20.

QUIROZ-CASTELLANOS, M. et al. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. **Computers & Operations Research**, v. 55, p. 52–64, 2015. ISSN 0305-0548. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0305054814002676>.

RATLIFF, H.; ROSENTHAL, A. Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. **Operations Research**, v. 31, p. 507–521, 06 1983.

ROODBERGEN, K. J. **Layout and Routing Methods for Warehouses**. Thesis (PhD), 01 2001.

ROZENBERG, G.; BÄCK, T.; KOK, J. N. (Ed.). **Handbook of Natural Computing**. Springer, 2012. ISBN 978-3-540-92909-3. Available from Internet: <https://doi.org/10.1007/978-3-540-92910-9>.

SAITO, M.; MATSUMOTO, M. SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator. **In Monte Carlo and Quasi-Monte Carlo Methods**, v. 64, n. 2, p. 607–622, 1993.

SCHOLZ, A. et al. A new mathematical programming formulation for the single-picker routing problem. **European Journal of Operational Research**, v. 253, n. 1, p. 68–84, 2016. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221716300388>.

SCHOLZ, A.; WäSCHER, G. Order Batching and Picker Routing in manual order picking systems: the benefits of integrated routing. **Central European Journal of Operations Research**, v. 25, p. 491–520, 2017.

THEYS, C. et al. Using a tsp heuristic for routing order pickers in warehouses. **European Journal of Operational Research**, v. 200, n. 3, p. 755–763, 2010. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221709000514>.

WEIDINGER, F. Picker routing in rectangular mixed shelves warehouses. **Computers & Operations Research**, v. 95, p. 139–150, 2018. ISSN 0305-0548. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0305054818300819>.

WEIDINGER, F.; BOYSEN, N.; SCHNEIDER, M. Picker routing in the mixed-shelves warehouses of e-commerce retailers. **European Journal of Operational Research**, v. 274, n. 2, p. 501–515, 2019. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221718308749>.

WHITLEY, D. A genetic algorithm tutorial. **Statistics and Computing**, v. 4, n. 2, p. 65–85, jun. 1994. ISSN 1573-1375. Available from Internet: <https://doi.org/10.1007/BF00175354>.

XIE, L. et al. Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems. **European Journal of Operational Research**, v. 288, n. 1, p. 80–97, 2021. ISSN 0377-2217. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0377221720304781>.

APPENDIX A - OBJECTIVE VALUES OVER GENERATIONS PER SEED

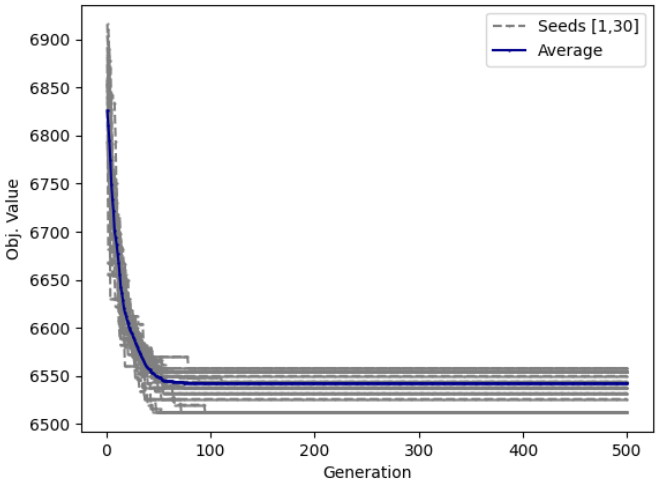

Figure 8.1 – Obj. Value over generations - instance abc1/29s-40-30. Source: the author.

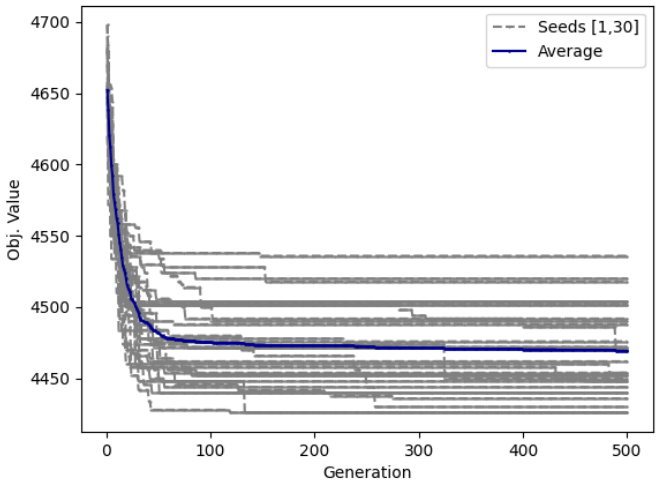

Figure 8.2 – Obj. Value over generations - instance abc1/30s-40-45. Source: the author.

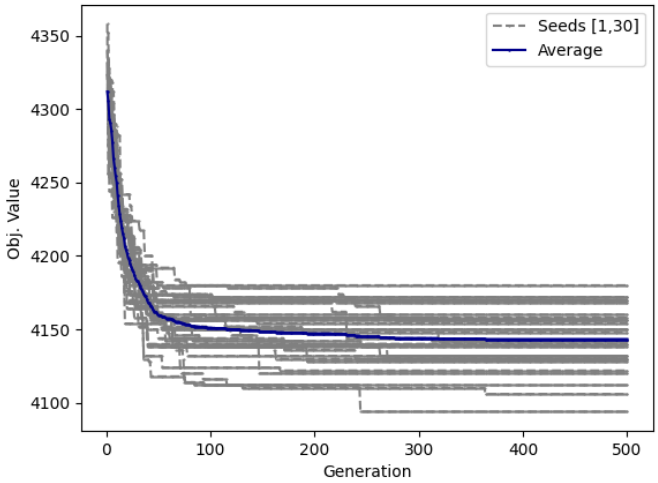

Figure 8.3 – Obj. Value over generations - instance abc1/31s-40-60. Source: the author.

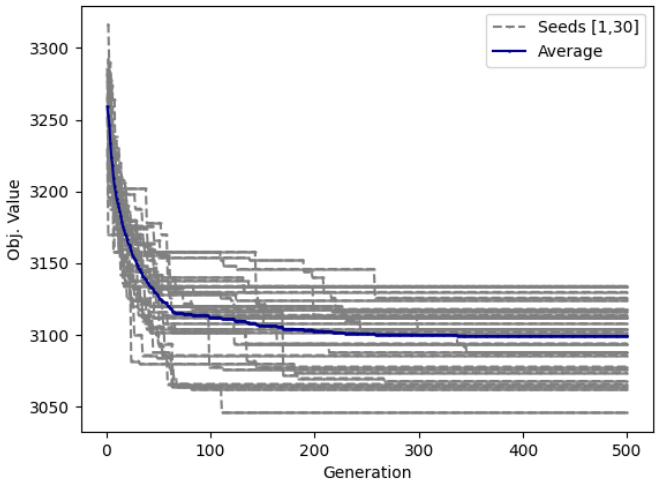

Figure 8.4 – Obj. Value over generations - instance abc1/32s-40-75. Source: the author.

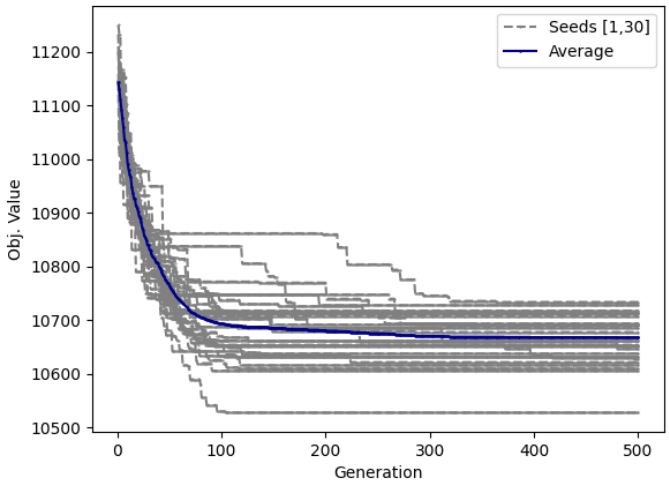

Figure 8.5 – Obj. Value over generations - instance abc1/37s-60-30. Source: the author.

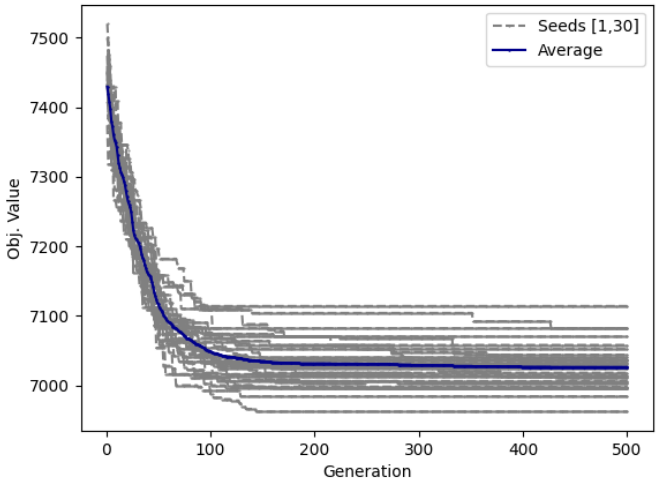

Figure 8.6 – Obj. Value over generations - instance abc1/38s-60-45. Source: the author.

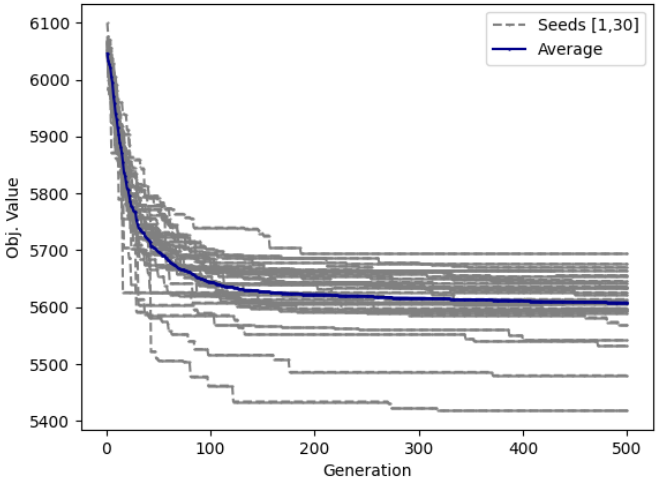

Figure 8.7 – Obj. Value over generations - instance abc1/39s-60-60. Source: the author.

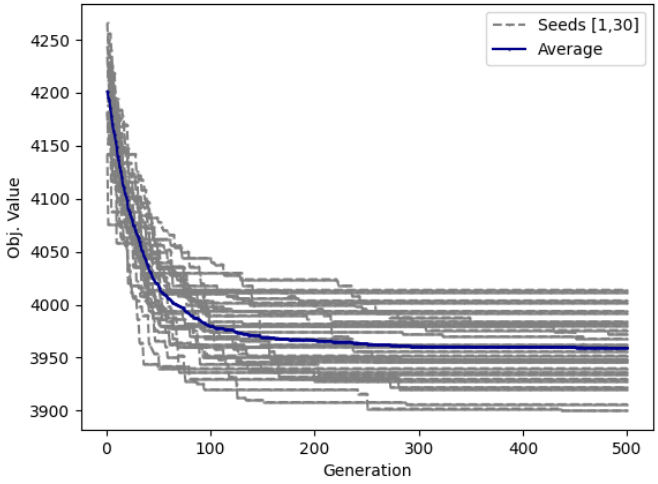

Figure 8.8 – Obj. Value over generations - instance abc1/40s-60-75. Source: the author.

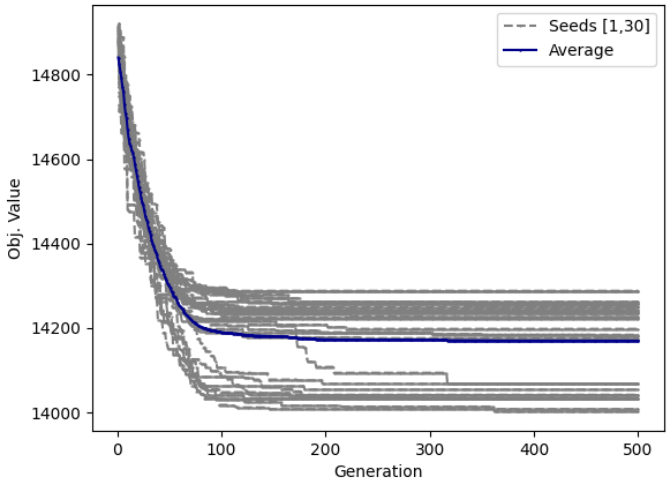

Figure 8.9 – Obj. Value over generations - instance abc1/61s-80-30. Source: the author.

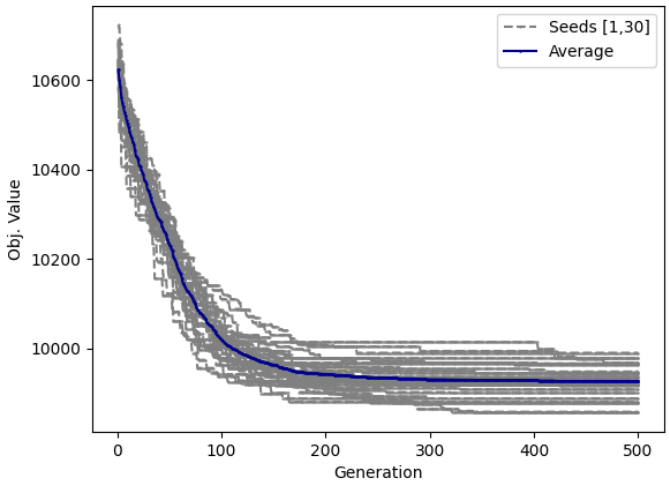

Figure 8.10 – Obj. Value over generations - instance abc1/62s-80-45. Source: the author.

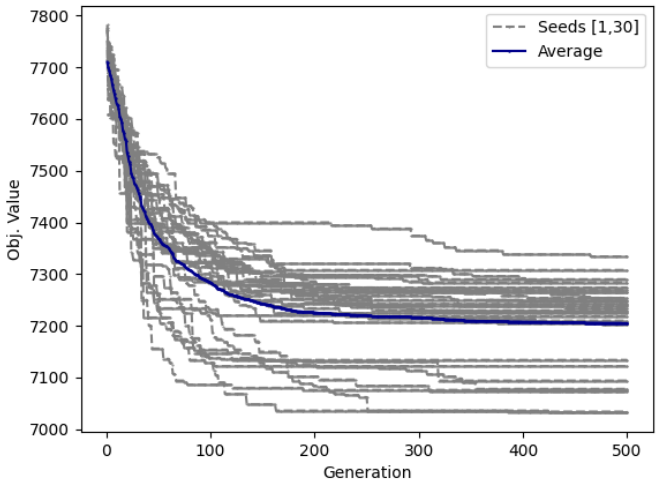

Figure 8.11 – Obj. Value over generations - instance abc1/63s-80-60. Source: the author.

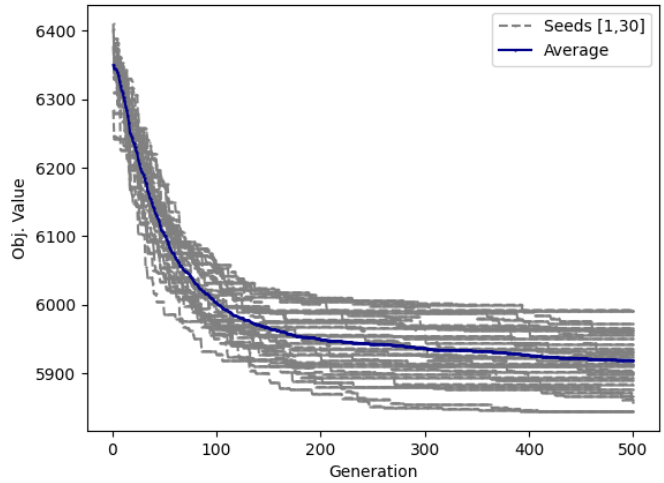

Figure 8.12 – Obj. Value over generations - instance abc1/64s-80-75. Source: the author.

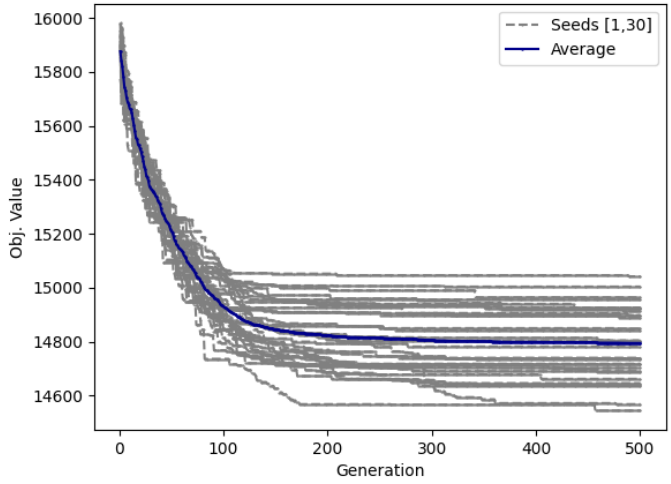

Figure 8.13 – Obj. Value over generations - instance abc1/69s-100-30. Source: the author.

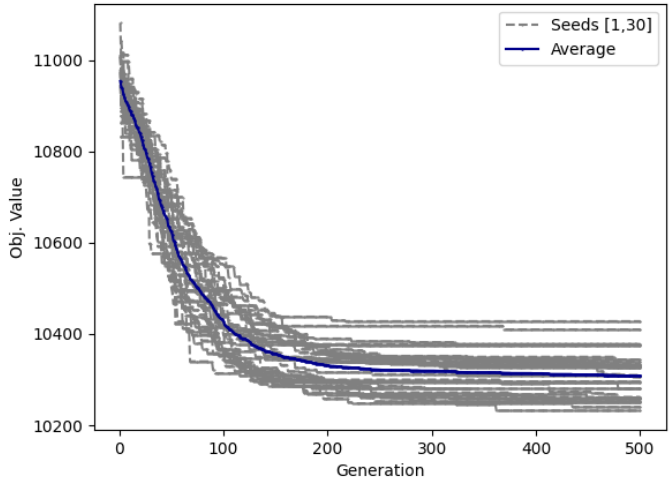

Figure 8.14 – Obj. Value over generations - instance abc1/70s-100-45. Source: the author.

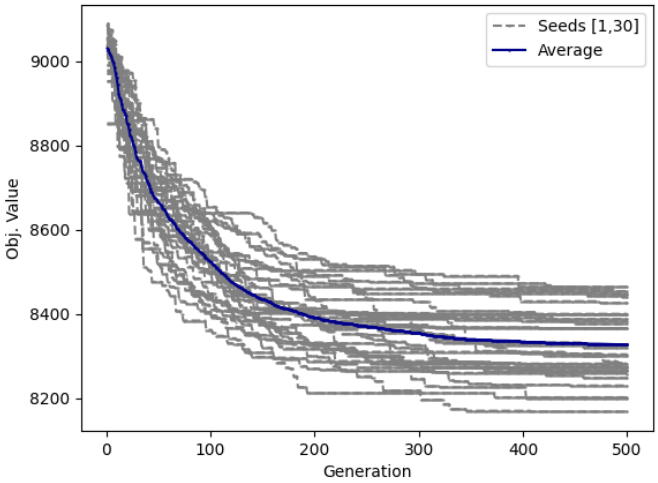

Figure 8.15 – Obj. Value over generations - instance abc1/71s-100-60. Source: the author.

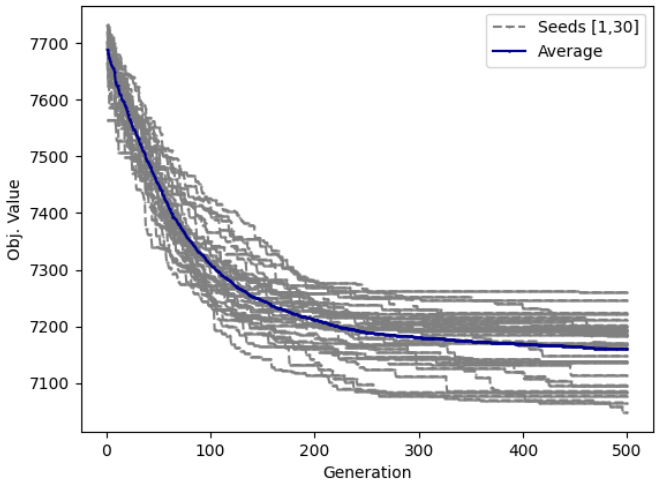

Figure 8.16 – Obj. Value over generations - instance abc1/72s-100-75. Source: the author.

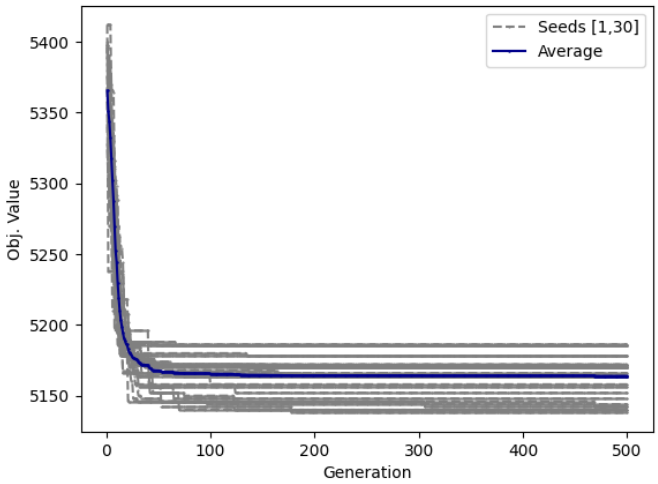

Figure 8.17 – Obj. Value over generations - instance abc2/10l-40-45. Source: the author.

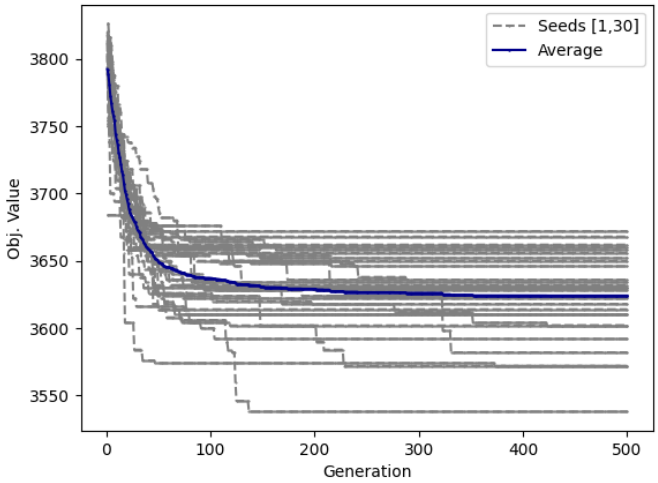

Figure 8.18 – Obj. Value over generations - instance abc2/11l-40-60. Source: the author.

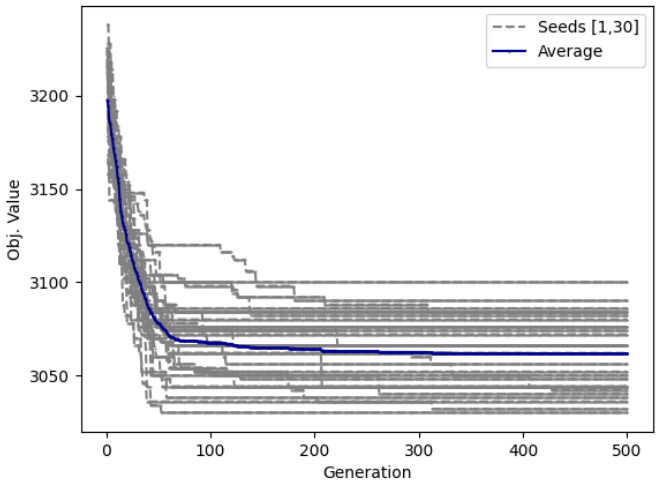

Figure 8.19 – Obj. Value over generations - instance abc2/12l-40-75. Source: the author.

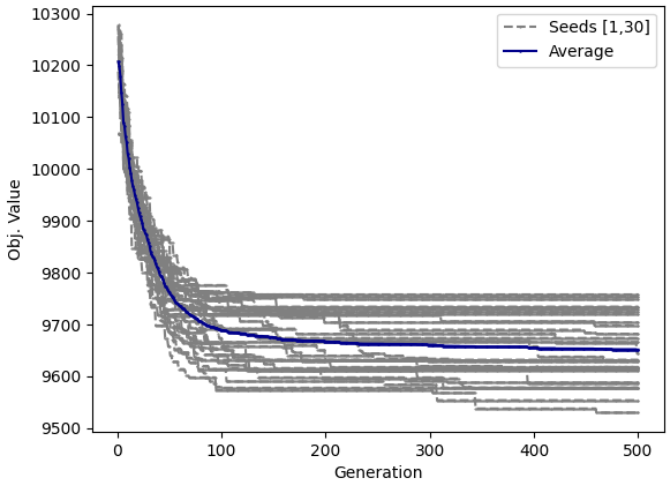

Figure 8.20 – Obj. Value over generations - instance abc2/17l-60-30. Source: the author.

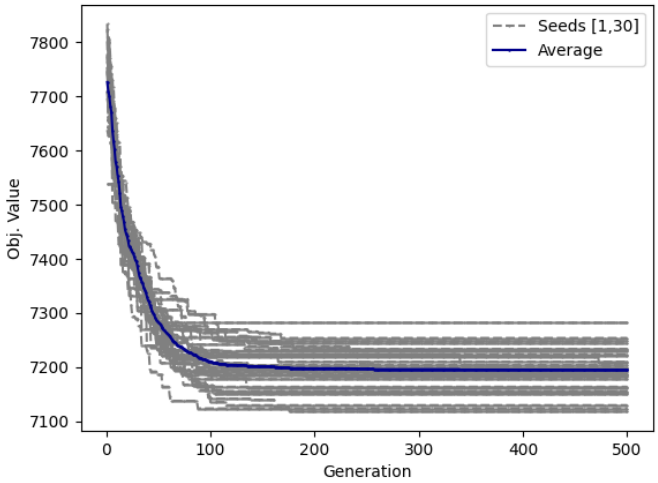

Figure 8.21 – Obj. Value over generations - instance abc2/18l-60-45. Source: the author.

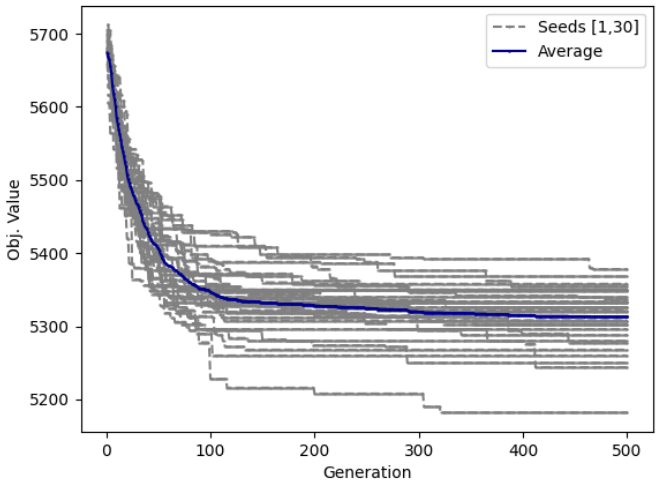

Figure 8.22 – Obj. Value over generations - instance abc2/19l-60-60. Source: the author.

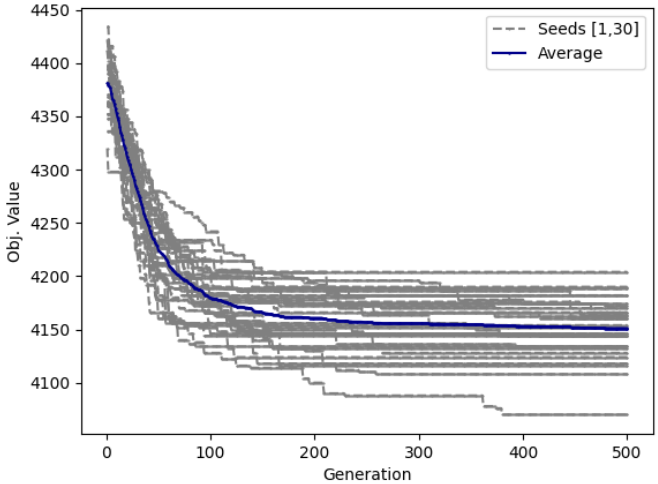

Figure 8.23 – Obj. Value over generations - instance abc2/20l-60-75. Source: the author.

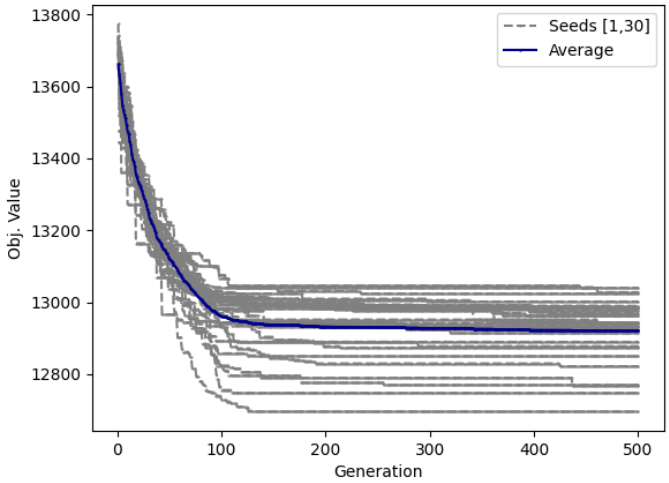

Figure 8.24 – Obj. Value over generations - instance abc2/45l-80-30. Source: the author.

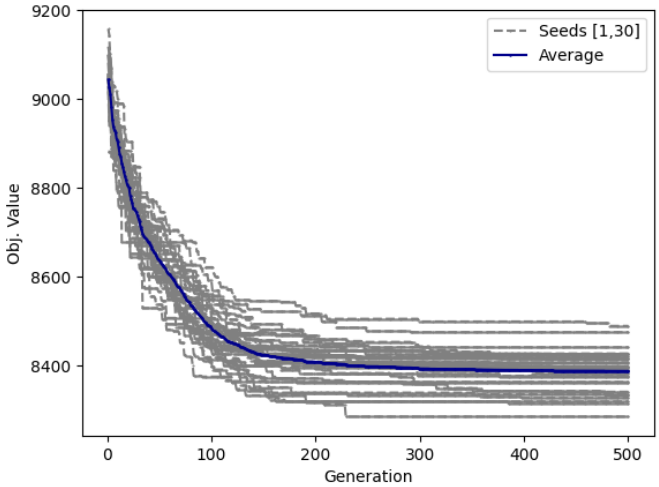

Figure 8.25 – Obj. Value over generations - instance abc2/46l-80-45. Source: the author.

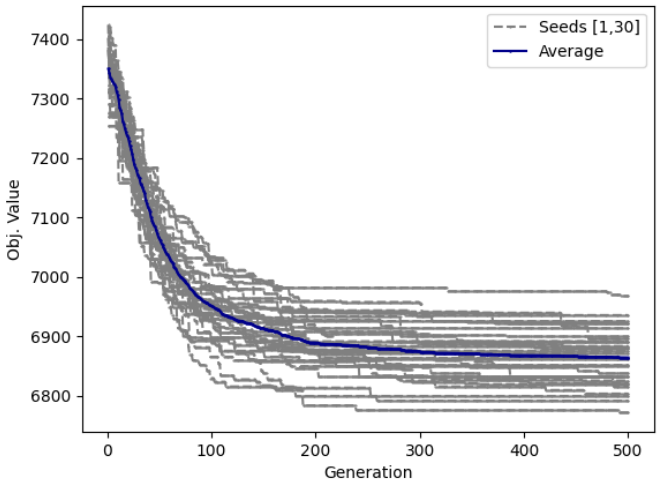

Figure 8.26 – Obj. Value over generations - instance abc2/47l-80-60. Source: the author.

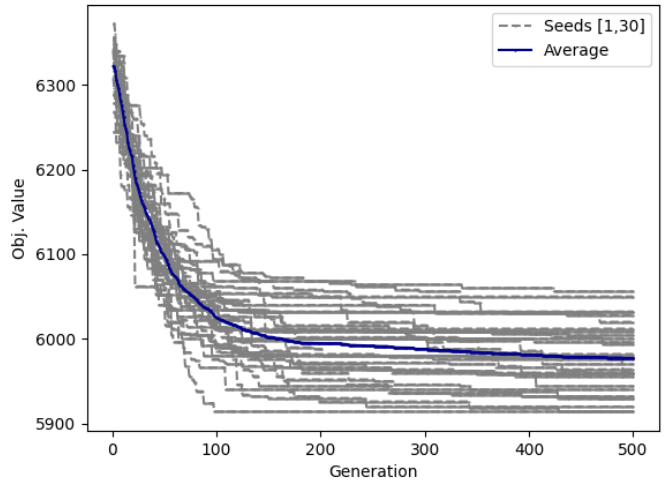

Figure 8.27 – Obj. Value over generations - instance abc2/48l-80-75. Source: the author.

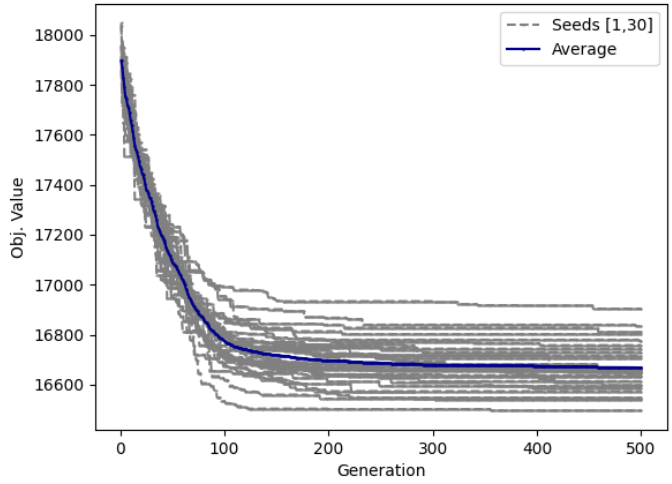

Figure 8.28 – Obj. Value over generations - instance abc2/53l-100-30. Source: the author.

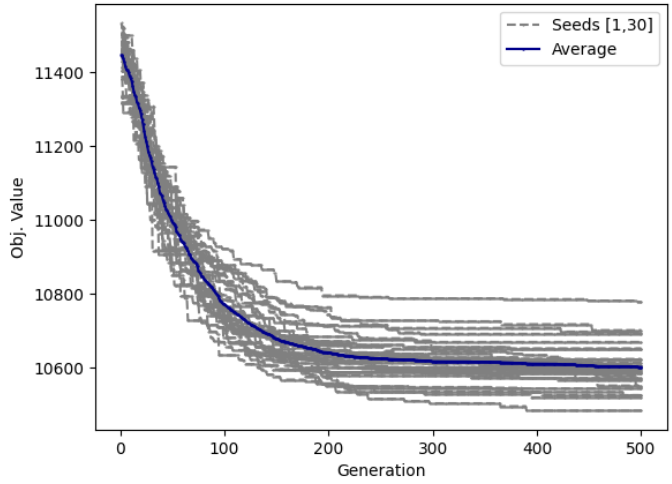

Figure 8.29 – Obj. Value over generations - instance abc2/54l-100-45. Source: the author.

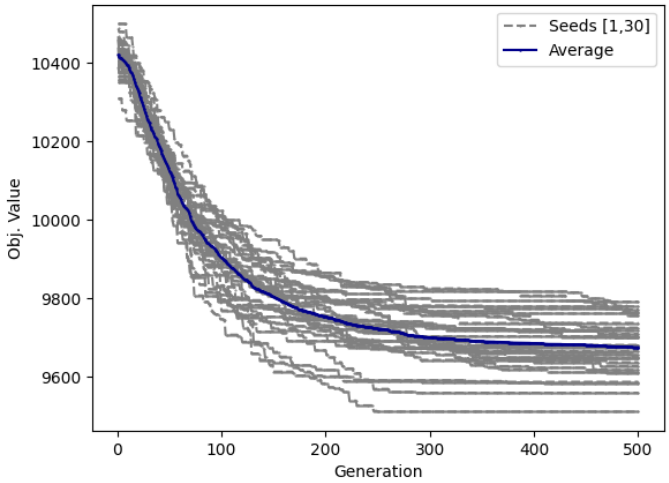

Figure 8.30 – Obj. Value over generations - instance abc2/55l-100-60. Source: the author.

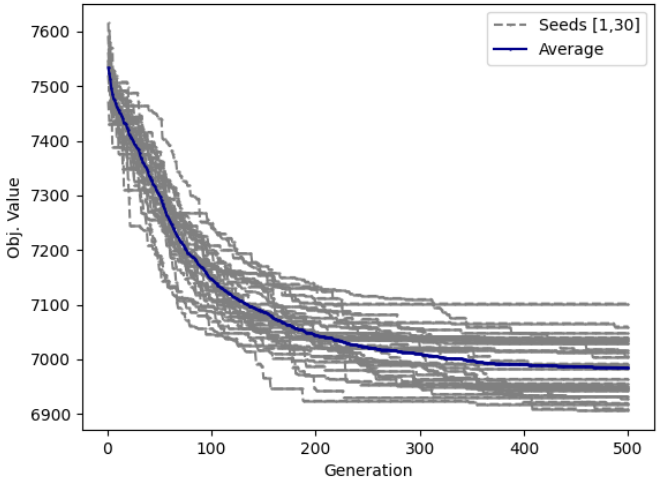

Figure 8.31 – Obj. Value over generations - instance abc2/56l-100-75. Source: the author.

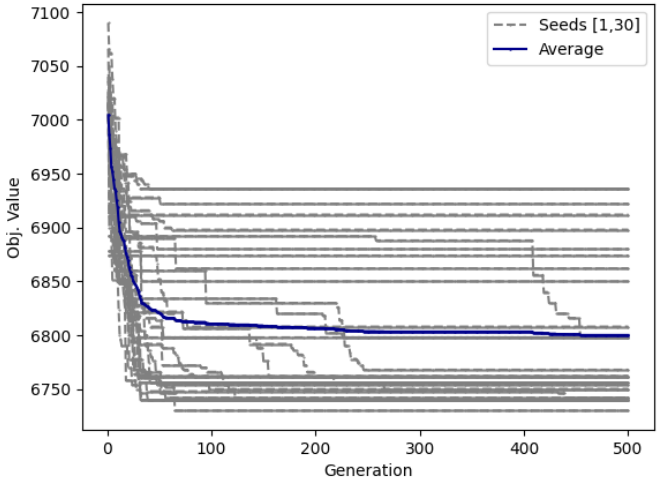

Figure 8.32 – Obj. Value over generations - instance abc2/9l-40-30. Source: the author.

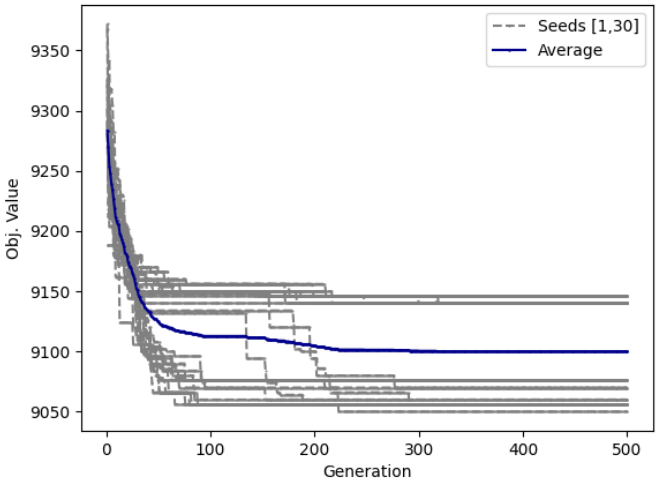

Figure 8.33 – Obj. Value over generations - instance ran1/29s-40-30. Source: the author.

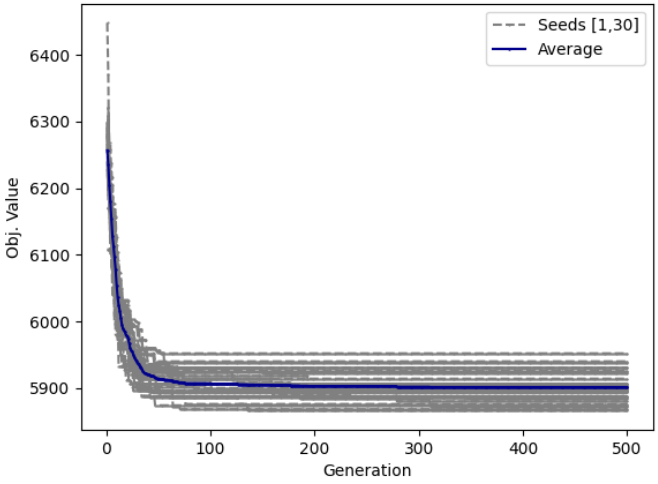

Figure 8.34 – Obj. Value over generations - instance ran1/30s-40-45. Source: the author.

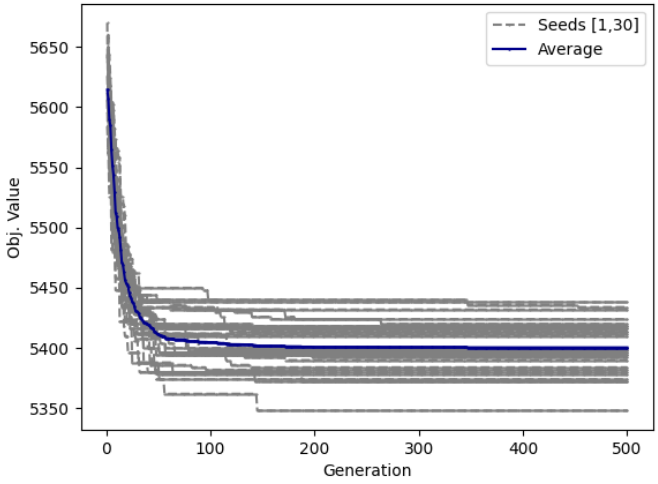

Figure 8.35 – Obj. Value over generations - instance ran1/31s-40-60. Source: the author.

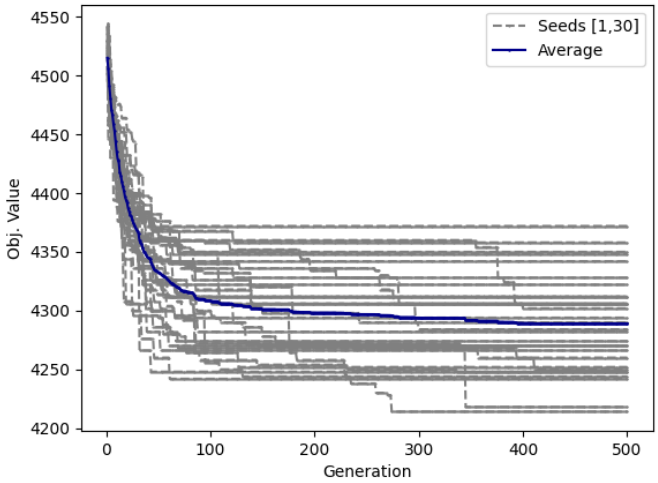

Figure 8.36 – Obj. Value over generations - instance ran1/32s-40-75. Source: the author.

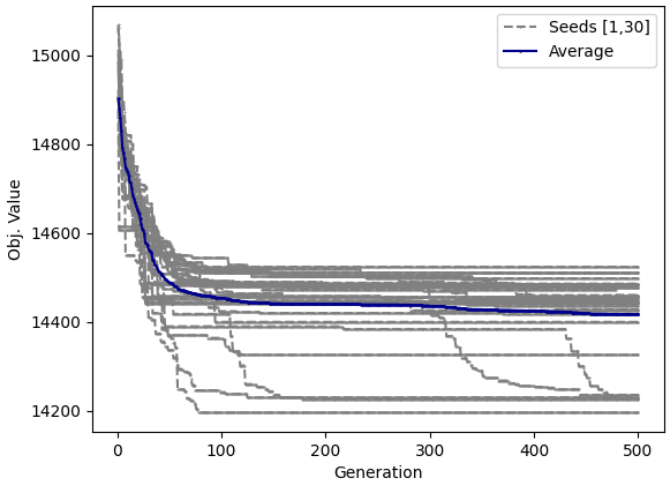

Figure 8.37 – Obj. Value over generations - instance ran1/37s-60-30. Source: the author.

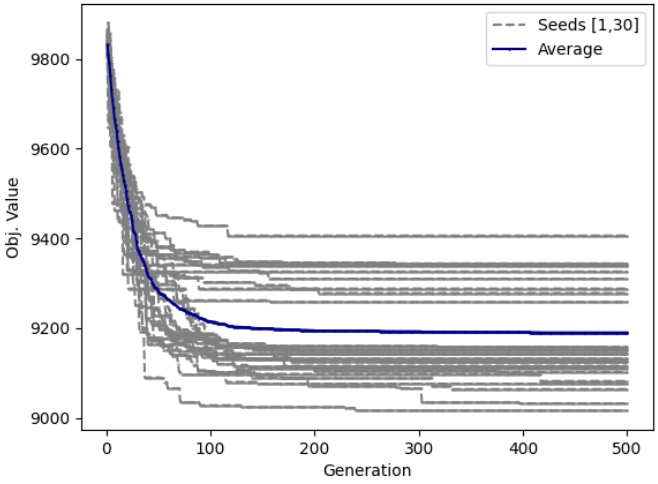

Figure 8.38 – Obj. Value over generations - instance ran1/38s-60-45. Source: the author.

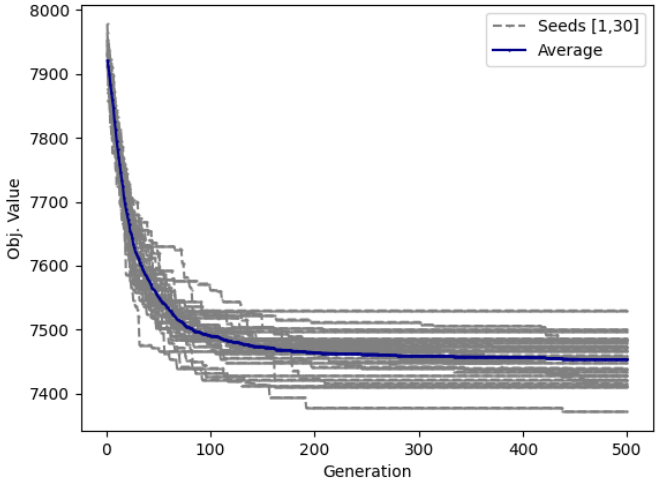

Figure 8.39 – Obj. Value over generations - instance ran1/39s-60-60. Source: the author.

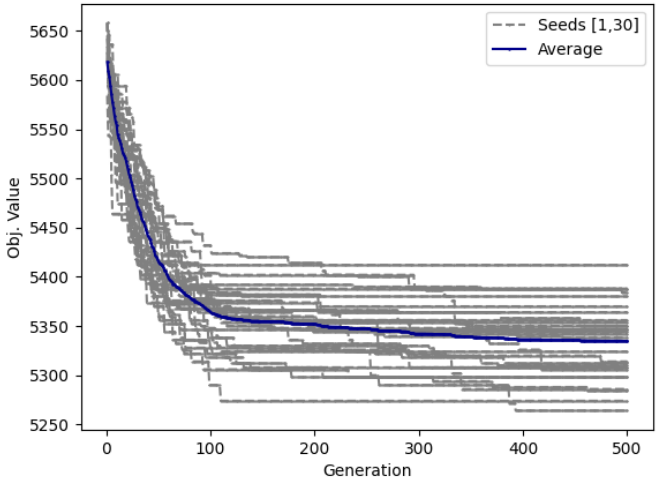

Figure 8.40 – Obj. Value over generations - instance ran1/40s-60-75. Source: the author.

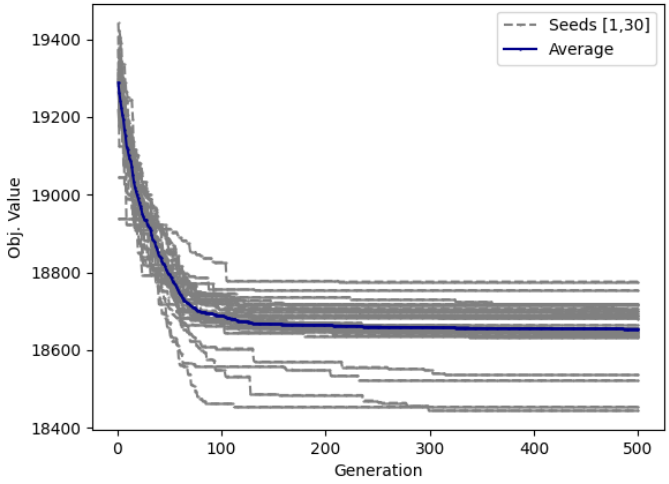

Figure 8.41 – Obj. Value over generations - instance ran1/61s-80-30. Source: the author.

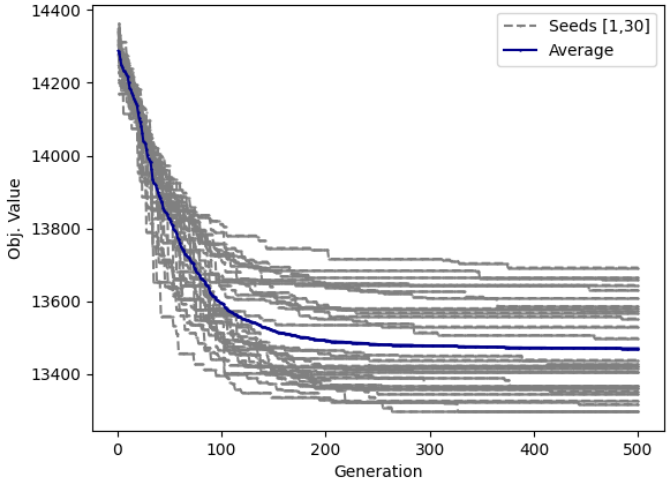

Figure 8.42 – Obj. Value over generations - instance ran1/62s-80-45. Source: the author.

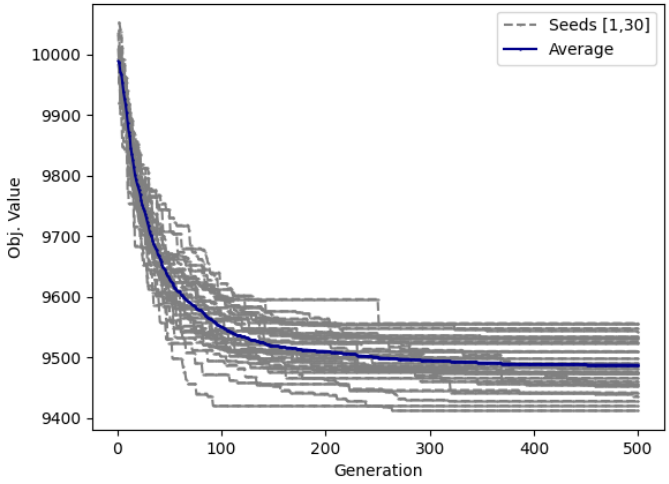

Figure 8.43 – Obj. Value over generations - instance ran1/63s-80-60. Source: the author.

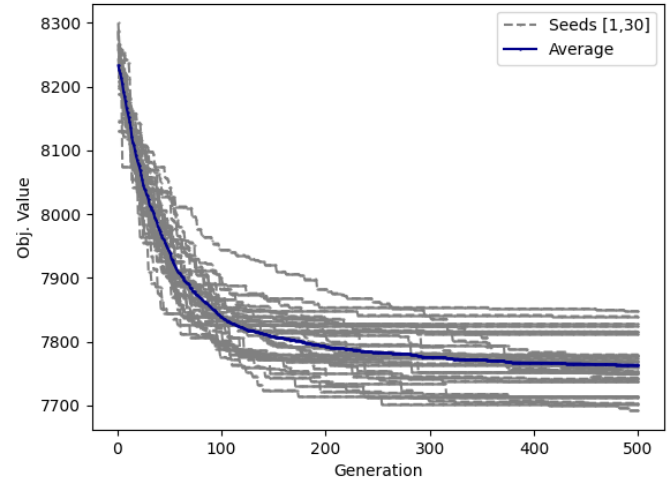

Figure 8.44 – Obj. Value over generations - instance ran1/64s-80-75. Source: the author.

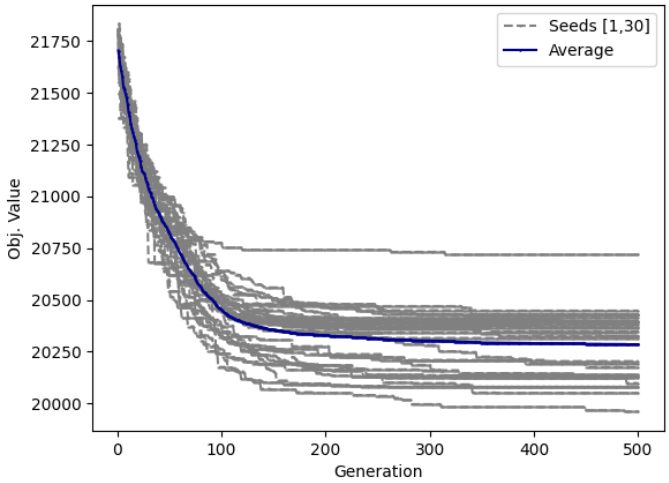

Figure 8.45 – Obj. Value over generations - instance ran1/69s-100-30. Source: the author.

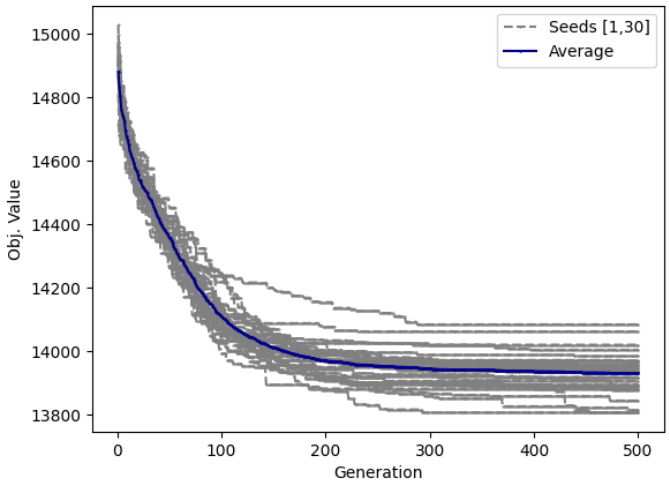

Figure 8.46 – Obj. Value over generations - instance ran1/70s-100-45. Source: the author.

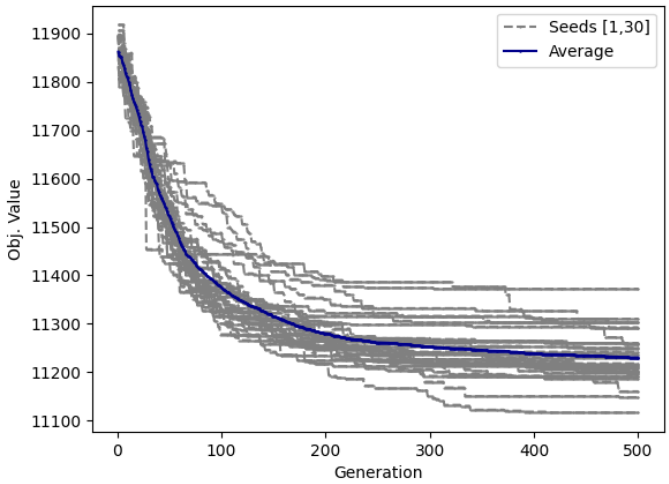

Figure 8.47 – Obj. Value over generations - instance ran1/71s-100-60. Source: the author.

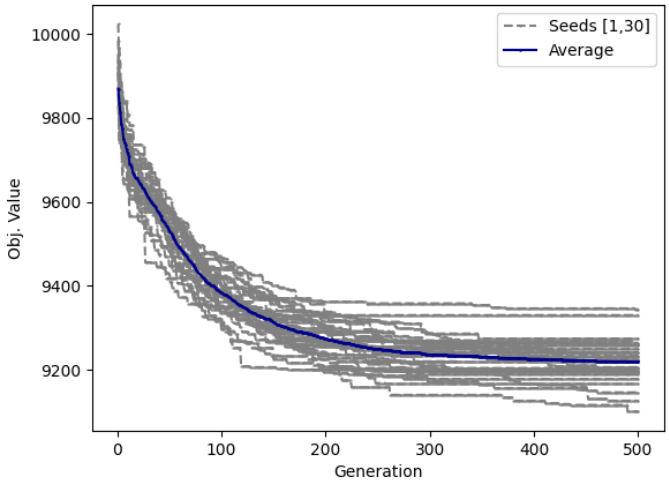

Figure 8.48 – Obj. Value over generations - instance ran1/72s-100-75. Source: the author.

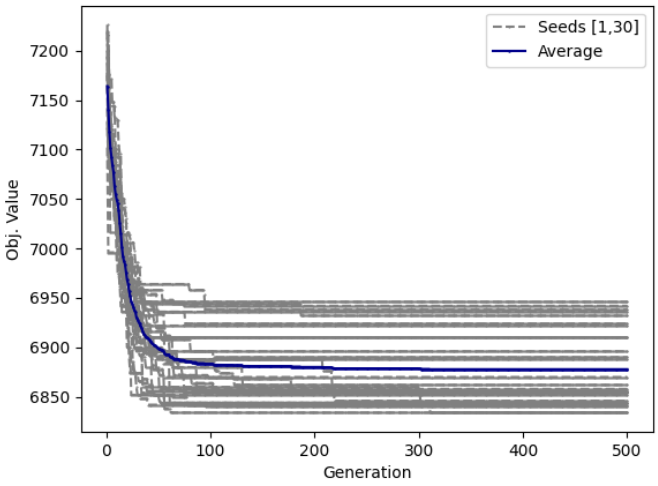

Figure 8.49 – Obj. Value over generations - instance ran2/10l-40-45. Source: the author.

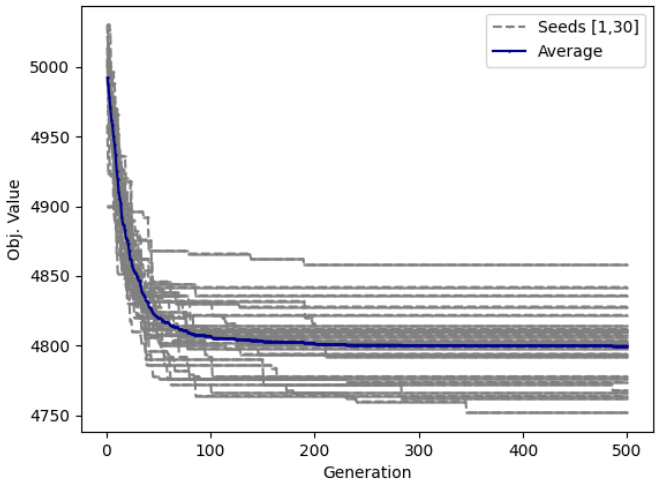

Figure 8.50 – Obj. Value over generations - instance ran2/11l-40-60. Source: the author.

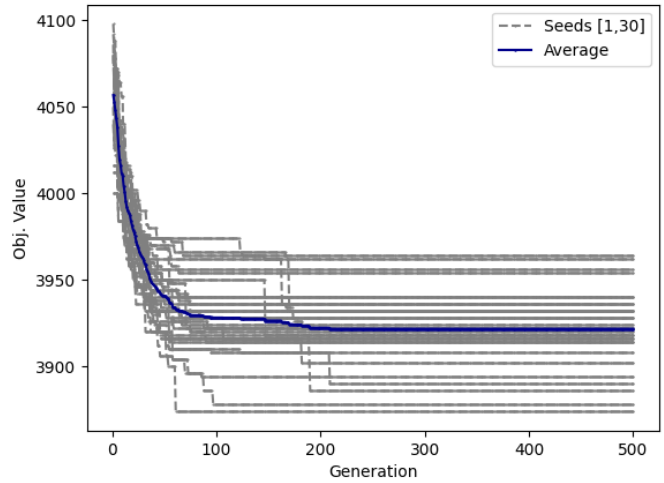

Figure 8.51 – Obj. Value over generations - instance ran2/12l-40-75. Source: the author.

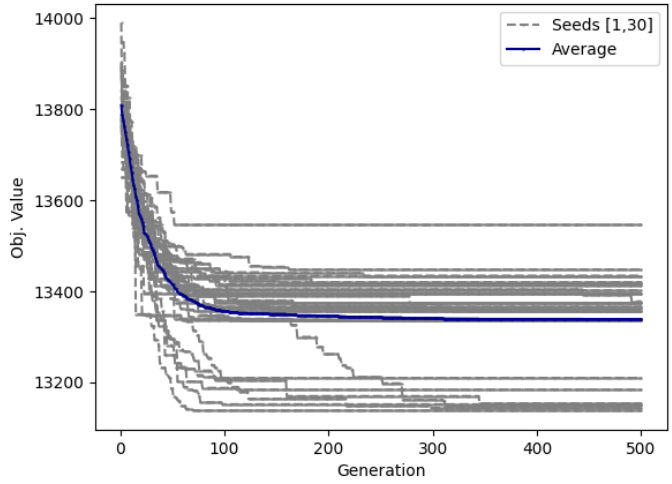

Figure 8.52 – Obj. Value over generations - instance ran2/17l-60-30. Source: the author.

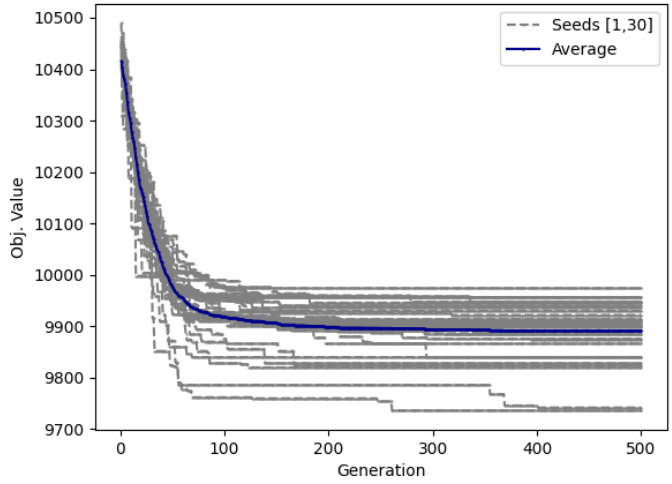

Figure 8.53 – Obj. Value over generations - instance ran2/18l-60-45. Source: the author.

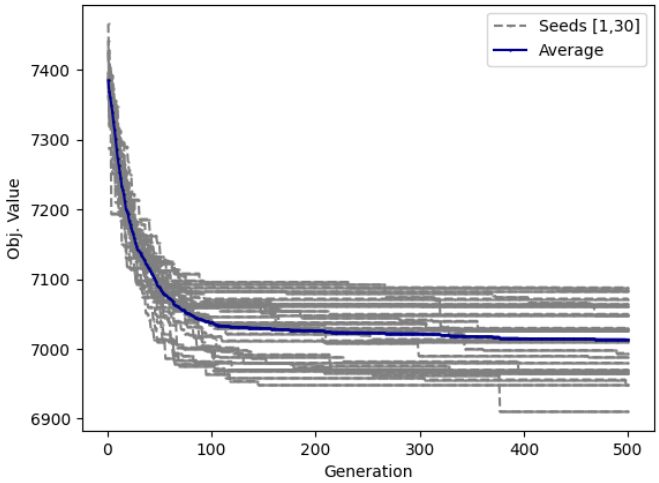

Figure 8.54 – Obj. Value over generations - instance ran2/19l-60-60. Source: the author.

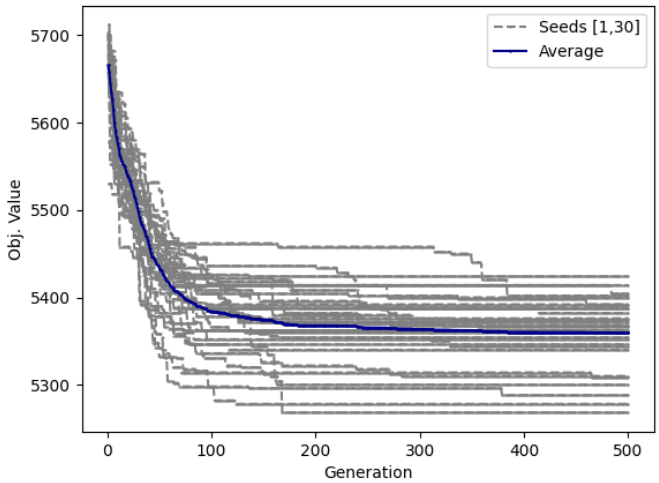

Figure 8.55 – Obj. Value over generations - instance ran2/20l-60-75. Source: the author.

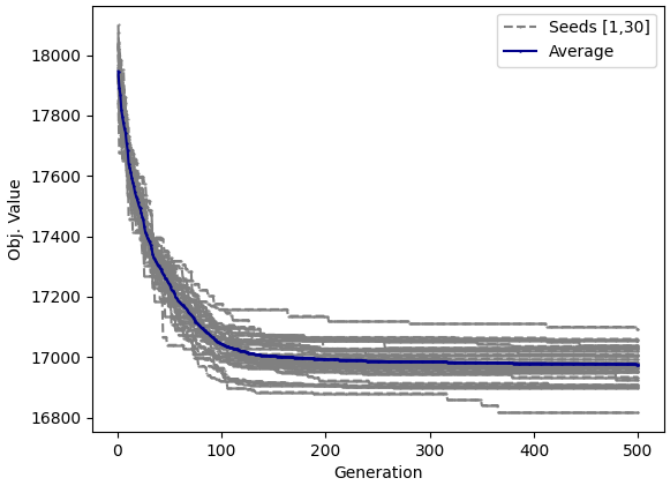

Figure 8.56 – Obj. Value over generations - instance ran2/45l-80-30. Source: the author.

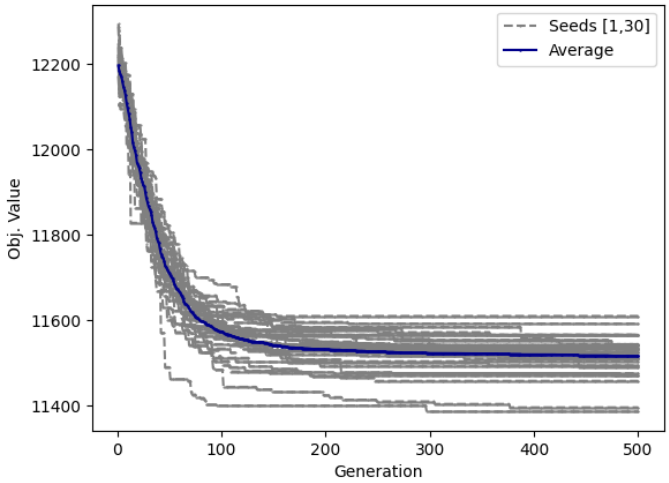

Figure 8.57 – Obj. Value over generations - instance ran2/46l-80-45. Source: the author.

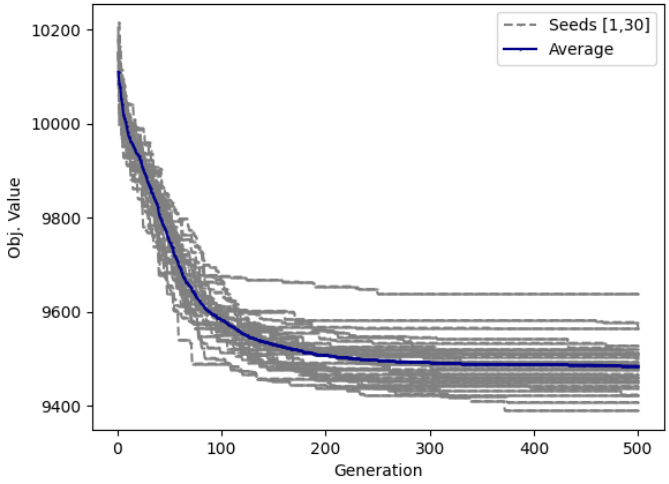

Figure 8.58 – Obj. Value over generations - instance ran2/47l-80-60. Source: the author.

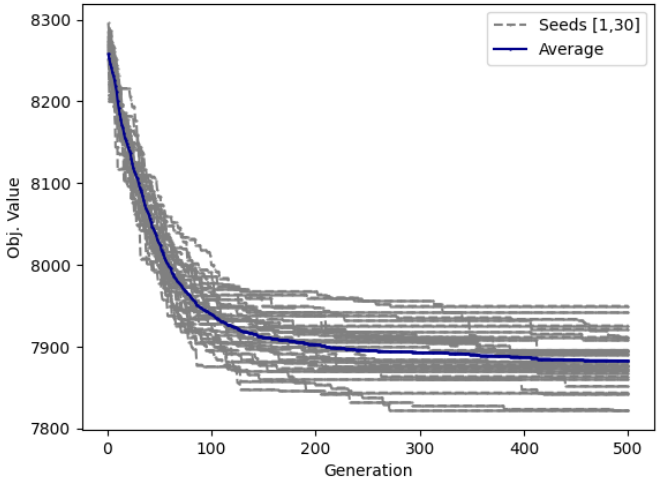

Figure 8.59 – Obj. Value over generations - instance ran2/48l-80-75. Source: the author.

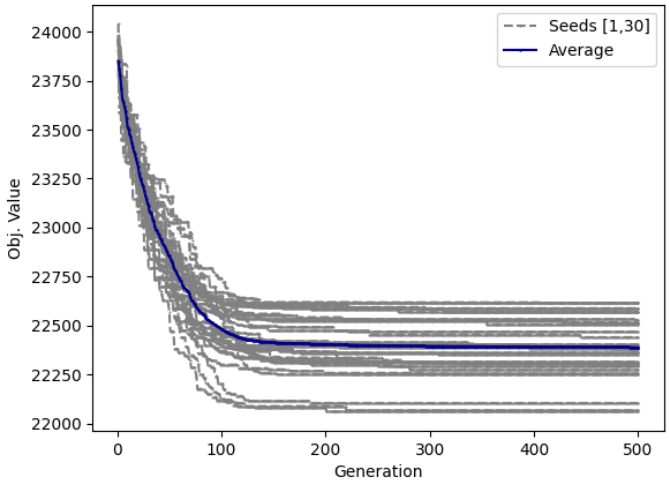

Figure 8.60 – Obj. Value over generations - instance ran2/53l-100-30. Source: the author.

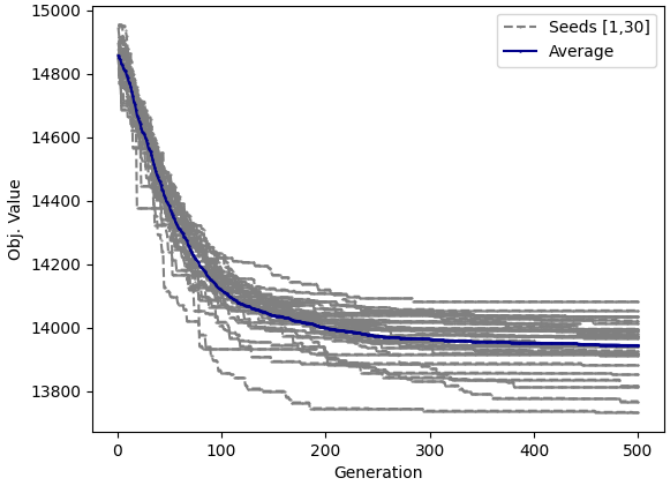

Figure 8.61 – Obj. Value over generations - instance ran2/54l-100-45. Source: the author.

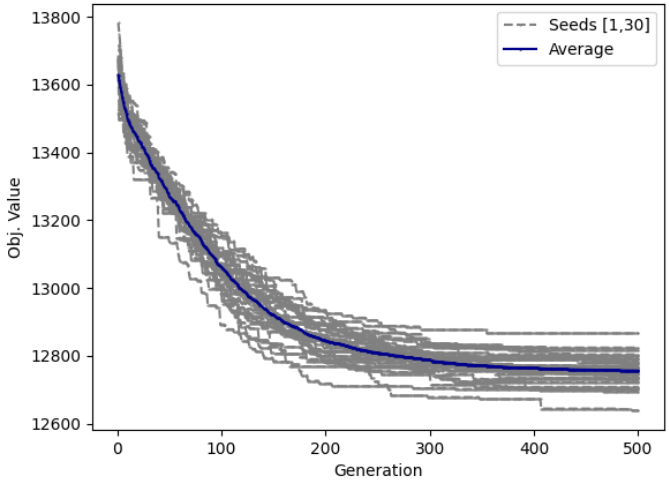

Figure 8.62 – Obj. Value over generations - instance ran2/55l-100-60. Source: the author.

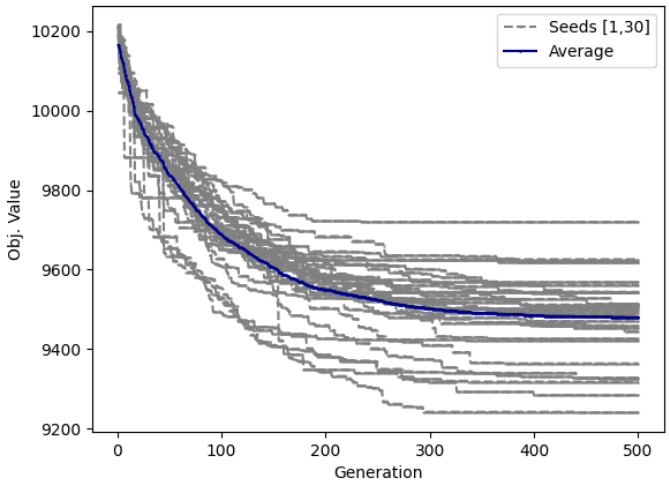

Figure 8.63 – Obj. Value over generations - instance ran2/56l-100-75. Source: the author.

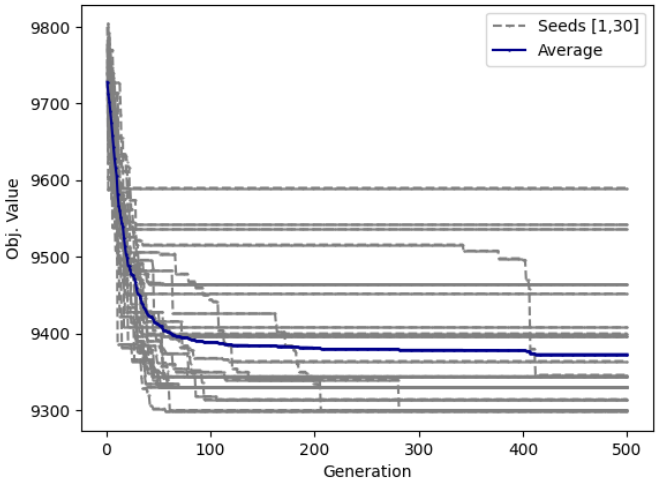

Figure 8.64 – Obj. Value over generations - instance ran2/91-40-30. Source: the author.

APPENDIX B - RESUMO EXPANDIDO

Tradicionalmente, administrar eficientemente grandes depósitos e armazéns não é uma tarefa fácil. A quantia de variáveis e processos envolvidos desde o momento em que um cliente realiza a compra de um único produto até o seu recebimento, são bastante consideráveis. Assim, suponha o caso de varejistas e atacadistas que vendem uma larga quantia de produtos e recebem milhares de pedidos diferentes diariamente, sendo que cada pedido demanda diferentes quantias e tipos de cada item: o problema torna-se um pouco mais complexo.

Nas últimas décadas, o crescimento de serviços digitais e internet impulsionou o e-commerce (vendas através de websites e aplicações), que vem se destacando como um modelo de negócio sustentável, escalável e rentável. Apenas no ano de 2021, estima-se que este mercado movimentou cerca de U$ 469.2 bilhões, apenas nos estados unidos (HOFSTETTERL, 2021). Estatísticas ainda apontam uma projeção para U$ 563.3 bilhões para 2025 (DEPARTMENT, 2022). Este grande mercado é baseado em preços atrativos, intuitividade e facilidade no momento das compras, e é claro, curtos períodos de entrega. Dessa forma, para otimizar todo o processo logístico, da venda ao recebimento, sistemas de gerenciamento de armazéns (WMS) eficientes, interconectados com sistemas integrados de gestão empresarial (ERP) tornam-se indispensáveis para lojistas que almejam sucesso financeiro em suas companhias (BOYSEN; de Koster; WEIDINGER, 2019).

Levando-se em consideração os WMS atuais, um desafio significativo dentro do contexto de armazéns é a dinâmica que envolve a coleta de produtos comprados pelos consumidores, uma vez que estudos indicam que certa de 55% dos custos operacionais nestes locais estão relacionados à este processo (PANSART; CATUSSE; CAMBAZARD, 2018). Assim, emergem problemas combinatórios práticos como o problema de coleta de pedidos (OPP) e o problema de loteamento de pedidos (OBP).

O OPP visa minimizar a distância viajada por um coletador (funcionário responsável por coletar os produtos das prateleiras dos armazéns) humano ou robô, durante a coleta de uma lista de produtos. O OBP é um problema interconectado, que tem por objetivo agrupar os pedidos dos clientes em lotes (que serão repassados aos coletadores). Os lotes devem respeitar um limite de capacidade (peso/volume). A ideia é encontrar uma distribuição dos pedidos em lotes, de forma que o somatório das distâncias percorridas pelos coletadores durante a retirada dos produtos requiridos seja minimizada. Tanto o OPP quanto o OBP são problemas que pertencem à classe de complexidade NP-Difícil

(GADEMANN; VELDE, 2005; KULAK; ŞAHIN; TANER, 2012).

Note que o OBP depende de alguma métrica de roteamento para encontrar possíveis lotes de pedidos. Quando este problema é tratado isoladamente, opções para lidar com as distâncias são usar políticas de roteamento (S-Shape, Largest Gap, Midpoint) ou heurísticas mais avançadas. Entretanto, uma alternativa é otimizar individualmente cada rota respondente à um lote durante o processo de loteamento, isso é, uma abordagem que execute o OBP simultaneamente com o OPP (SCHOLZ; WäSCHER, 2017). Esta estratégia é conhecida como problema de loteamento e coleta simultânea de pedidos (JOBPRP) (BRIANT et al., 2020).

Neste trabalho, apresentamos um novo modelo matemático e propomos uma heurística baseada em programação dinâmica e um algoritmo genético de agrupamento para o JOBPRP em armazéns com layout de bloco único. Nosso modelo funciona independentemente do layout do armazém (pode ser multi-bloco, multi-andar ou assumir outros formatos além do retangular) e pode ser usado para resolver instâncias pequenas.

A abordagem heurística aplica dois níveis de programação dinâmica: em um primeiro momento, executa um procedimento baseado na solução do problema da mochila para gerar uma configuração inicial de lotes; após, executa um algoritmo de programação dinâmica pra encontrar a rota ótima de cada lote. O valor objetivo é dado pelo somatório das distâncias percorridas em cada lote.

A abordagem metaheurística aplica uma heurística para gerar soluções iniciais, que atribui os pedidos mais similares ao mesmo lote. A distância percorrida em cada lote é calculada de forma ótima através de um algoritmo de programação dinâmica, e o procedimento de crossover utiliza uma técnica de transmissão controlada de genes.

Para avaliar nossas propostas, executamos experimentos computacionais com conjuntos de dados fornecidos pela literatura. O modelo matemático foi utilizado em um software solucionador de programas inteiros-mistos (Gurobi), onde se realizaram testes com pequenas instâncias para aferir a qualidade das soluções da nossa abordagem metaheurística. Nossos resultados computacionais evidenciaram alta estabilidade para todas as instâncias testadas e menores valores objetivo que os reportados previamente na literatura, mantendo um tempo de execução razoável.