Iterated Greedy Local Search for the Order Picking Problem Considering Storage Location and Order Batching Decisions

Preprint PISIS-RR23-01
Graduate Program in Systems Engineering, Department of Mechanical and Electrical Engineering
Universidad Autónoma de Nuevo León
San Nicolás de los Garza, NL, Mexico, January 2023

Diana L. Huerta-Muñoz
Universidad Autónoma de Nuevo León (UANL)
Graduate Program in Systems Engineering
dianahuerta@yalma.fime.uanl.mx

Roger Z. Ríos-Mercado
Universidad Autónoma de Nuevo León (UANL)
Graduate Program in Systems Engineering
roger.rios@uanl.edu.mx

Jesús Fabián López-Pérez
Universidad Autónoma de Nuevo León (UANL)
Business Management and Public Accounting Department
Texas A&M International University
College of Business
jesus.lopezpz@uanl.edu.mx

January 2023

Abstract

Order picking is the process of collecting products from a specific location to complete customer orders. It is the most costly activity inside of a warehouse with up to 65% of the incurred costs. On the other hand, two important and closely related problems to the order picking are, indeed, the storage location of products and the order batching, which may affect the routes performed by pickers if they are not optimized. In the first one, the decision to take into account is about where to place the items arriving at the warehouse considering that there are several and available locations with certain capacity and, in the second one, the decision is to determine how to group customer orders into batches, which must be assigned later to pickers to perform the corresponding routes. Although these three problems are commonly studied and solved independently, recent studies have shown that their integration may result in a greater improvement. In this paper, we study an integrated picking problem that considers these three subproblems simultaneously. The problem is motivated by a real-world application in a local warehouse. The aim of this integration is to obtain the best storage location of products, order batching, and picking sequence that minimize the total picking routing cost. We first present a mixed-integer linear programming model. Given the inherent computational complexity of the problem, we propose an iterated greedy local search metaheuristic that attempts to exploit some particular properties of the model. The proposed method is extensively assessed on random pseudo-real and real-world instances. The results indicate that instance sizes with up to 8197 order lines, 654 customer orders, and 4252 products, can be solved efficiently.

Keywords: Order picking; Storage location assignment; Order batching; Picking routing; Integer programming; Metaheuristics.

1 Introduction

In a supply chain and its corresponding logistic activities, one of the most important elements to consider is the warehousing process and other related activities in a distribution center (DC, which can be seen as a specific type of warehouse). These related activities performed in a DC have an important effect in the on-time deliveries to the final customers. Typical activities such as receiving products, allocating items at a corresponding storage location, grouping several order into batches, collecting products to complete customer orders, which are then packaged and delivered to the final customers, are some of the most important key elements for a successful DC performance. If all or some of them are not properly connected, they can generate costly logistic operations in the supply chain and create a poor perception for the final customers. From all these activities, there is one in particular that incurs in most of the warehouse costs, called the *Order Picking* (OP).

Order picking is the process of retrieving products from a specific location within a warehouse to complete customer orders. It is the most costly activity inside of a warehouse with up to 65% of the incurred costs (de Koster et al., 2007; Christophe et al., 2010). Although all the activities performed within a DC have an important effect that determines the level of efficiency of the order picking, there are two activities that are closely related and whose effect is direct and significant, *the storage location of products* and *the order batching*, which may affect the sequence of routes performed by pickers if they are not well-optimized, or if they are independently optimized, which is usually the case. In the storage location assignment, the decision to take into account is where to place the items arriving at the warehouse considering that there are several available locations or racks. In the order batching, the decision is to determine how to group customer orders, which must be assigned later to pickers to perform the corresponding routes. Although these three problems are commonly studied and solved independently, recent studies have shown that their integration may result in a greater improvement (Silva et al., 2020).

Focus of the paper. In this work, we aim to study the integration of the storage location of products, order batching, and routing decisions that minimize the total picking cost within a DC. The study is motivated by a real-world application in a local retail firm, which has several distributions centers that perform a manual picking activity each day.

Contribution. The contribution of this paper is twofold. First, we introduce a mixed-integer linear programming (MILP) model that integrates the storage location of products, order batching, and order picking decisions. Second, we develop an Iterated Greedy Local Search (IGLS) metaheuristic to obtain good quality feasible solutions efficiently to this integrated problem. The metaheuristic is fully assessed in both synthetic and real-world instances.

This work is organized as follows. First, in Section 2, the related literature is discussed. We present the problem under study and the proposed mathematical model in Section 3. The proposed solution method is described in Section 4. We provide an extensive computational experience in Section 5. Finally, some conclusions and future research lines are discussed in Section 6.

2 Literature review

According to de Koster et al. (2007); Christophe et al. (2010), the order picking, which is the operational activity of retrieving items from a specific storage location to complete customer orders delivered at a later stage, incurs from 55% to 65% of the warehouse costs. Some of the operational costs involved in this activity are related to three main components: *traveling time, time to pick an item,* and *time for remaining picking activities*; being the traveling time the most costly among them (Tompkins et al., 2010; Dukic and Opetuk, 2012). Several strategies proposed to improve the order picking efficiency with respect to this criterion are related to the integration of planning, tactical, and operational decisions, for example, layout design, the storage location of products, the order batching, and the sequence of routes that pickers must perform; as well as other criteria such as the size of the warehouse, order characteristics, zoning or replenishment decisions (van Gils et al., 2018a; Celik et al., 2022), which have not been studied deeply and can provide a significant order picking improvement in specific cases. Three main decisions can be identified because of the impact on the picking travel effort: Storage location of items, grouping customer orders to be performed in a single route, and the sequence of picking routes. In most of the literature, these decisions are studied separately or only two of them are combined. This work addresses the simultaneous integration of these three activities.

This section is divided into two parts. First, we briefly describe some aspects for each studied activity. Then, we provide a literature review related to the integration of two or more activities carried out in a warehouse in order the minimize picking costs.

2.1 The Storage location assignment, order batching, and order picking problems

Storage location assignment problem (SLAP): The Storage Location Assignment Problem (Hausman et al., 1976) is a tactical problem that decides where to place items into storage locations inside a warehouse, in order to make better use of the
physical space. A general review about the SLAP can be found in Rojas Reyes et al. (2019).

Since the SLAP is considered $\mathcal{NP}$-hard as it generalizes the Quadratic Assignment Problem (Loiola et al., 2007, QAP), it is common to identify several policies used to solve the problem easily. Some of the most common policies identified in the literature are the following (Silva et al., 2020; van Gils et al., 2018a,b):

- *Random*: In this policy, items are placed in any available storage location as long as capacities and dimensions of the locations allow it. It is considered a simple, fast, but inefficient heuristic for traditional commerce (it is more appropriated for e-commerce companies).
- *Dedicated*: Each item is assigned to a pre-defined and fixed storage location.
- *Rotation level*: This policy assigns high-turnover products to the best locations, those with the shortest distance to the depot (arrival/depart point).
- *Class-based*: In this policy, items are classified first, then they are assigned to a pre-defined area or zone, generally on a random basis.
- *By affinity*: Items that are often ordered together or have a certain level of affinity, are considered to be placed on the same storage location .
- *Closest space*: Items are located at locations closer to the depot.

Order batching problem (OBP): The Order Batching Problem is the problem of grouping customers orders into batches that will be collected later by pickers, in order to minimize routing costs. Order batching has a significant effect on picking costs as it reduces efforts by grouping several orders into one route, such that the better order assignment the better the picking solutions. This can be achieved by considering several orders with similar products or by applying a priority criterion. A survey of applications of the OBP can be found in Henn et al. (2012).

Since the OBP is also an $\mathcal{NP}$-hard problem (Gademann and Velde, 2005), solving to optimality real-world instances is not a simple task. Because of this, several simple rules or policies have been implemented. A common classification found in the literature is the one proposed by de Koster et al. (2007), which is as follows.

- *Basic methods*: Composed of constructive heuristics. One the most common example of this classification is the rule *First-Come-First-Serve* (FCFS), which is an strategy that groups orders into batches according to their arrival.
- *Seed methods*: A *seed order* is selected as a reference for other orders to be added in a specific batch. When a new order is added, it must not exceed the batch capacity, and the order to be selected from all the feasible ones, is the order that minimizes a measure of *closeness* to the *seed*. This process is repeated until there are no orders to add.
- *Savings methods*: These methods start by locating one order in one batch, then the number of batches is reduced by merging them as long as savings are found. This idea is similar to the one proposed by Clarke and Wright (1964), which correspond to one of the best-known heuristics used for vehicle routing problems.

Order picking problem (OPP): The third and the main activity of this study is the Order Picking Problem, which involves several tasks. The most important and time consuming of all of these tasks is to find the best design of routes, through a sequence of visited racks to retrieve the items, that minimizes the total traveling cost. This special task can be interpreted as a Vehicle Routing Problem (Braekers et al., 2016) or a Traveling Salesman Problem (Laporte, 1992) and, in both cases, it is an $\mathcal{NP}$-hard problem. The reader is referred to the following OPP survey for more details (van Gils et al., 2018b).

There are several ways to tackle the sequence of the order picking activity considering different policies proposed for the OPP. As it is done for the SLAP and the OBP, some of the most common policies used to obtain a sequence of the picking routes are (Petersen and Aase, 2004; van Gils et al., 2018a,b):

- *Return*: This policy establishes that pickers arrive and depart through the same point of a warehouse aisle (only if there are items to pick).
- *S-Shape*: Pickers enter in an aisle, traverse it until the end, and departs from that opposite point (in a S shape).
- *Mid-point*: Pickers enter and leave from the same point of the aisle, and only arrive as far as the middle point of it.
- *Largest gap*: Pickers enter and return to the same point of an aisle, but travel only as far as the larger gap between two adjacent items to be collected.
- *Optimal*: In this policy, pickers perform the optimal sequence (the one that provide the best solution cost), regardless the layout design and the location of items to collect.
- *Metaheuristic*: These routing policies use optimization heuristic procedures to compute the sequence of picking routes.

Current research works have studied and determined that integrating two or more of these decisions can result in better order picking solutions, although the complexity of the problem to be solved increases. Below, we provided a brief description of some relevant works found in the literature, which have shown that integrating some of these activities can provide significant improvements to the OPP.

2.2 Integrating tactical and operational activities for the OPP

The works presented in this section are an example of the effort and interest in improving the picking activity through the integration with other highly related decisions. These are briefly described below.

Hsieh and Huang (2011) develop a simulation to asses the integration of the storage location assignment (SLA) of products, order batching (OB), and picking routing (PR). Authors implement two known SLA and three PR policies from the literature, and proposed two new OB heuristics based on k-means and self-organization map algorithms (both correspond to clustering methods, where similar customer orders are grouped into one batch). Computational experiments were carried out on generated instances of an illustrative example with up to 300 orders and 20 SKUs (using a warehouse of 400 storage locations) to asses the effect of each applied strategy. Other works that consider simulation to evaluate the integration of SLA, OB, and PR policies on multi-block warehouses are proposed by Chen et al. (2010) and Chackelson et al. (2013). The first one carries out a Monte Carlo simulation to analyze the integration of five variants of the class-based SLA policy, two OB policies, and four well-known PR policies. The second one, applies simulation to analyze the effect of the integration of the SLA, OB, and PR (two policies per each activity) on a real-world application of a Spanish retail distributor. In all the works mentioned above, an ANOVA was performed considering the total distance traveled as one of the main objectives to evaluate, all of them concluding on the importance of integrating these three activities simultaneously.

Öncan (2015) studies the OBP combined with several routing policies for a rectangular layout. Since there are few studies on exact methods for the OBP, the author proposed several MILP formulations, one per each routing policy, and an Iterated Local Search Algorithm (ILS) with Tabu Thresholding to solve instances of larger size. The results indicate that their proposed ILS outperforms a general-purpose MILP solver and some other adaptations of savings heuristics implemented for the OBP. A previous work that also studies the optimization of the OBP in order to improve the OPP, is the proposed by Henn and Wäscher (2012), which provide a Tabu Search and an Attribute-based hill climber algorithm to evaluate the improvement in the total traveling time. Some well-known SLA and PR policies are used to obtain the storage location of products and the final picking routes.

A statistical analysis to evaluate the significance among several activities inside of a warehouse, which affects the order picking, was provided by van Gils et al. (2018a). They applied a full-factorial ANOVA considering several policies for the order batching, storage location, zone picking, and picking routing. They found that all the hypothesis they provided were accepted, which means that combinations of these activities provide a significant effect in order picking costs. Later, van Gils et al. (2019b) extend this analysis by taking into account real-life features, e.g., picker blocking, safety constraints, and high level storage locations to analyze the effect of including these features on the traveling and waiting picking times. The authors conclude that it is necessary to include these characteristics during picking optimization to correctly evaluate the performance of the proposed methods and solutions.

A mathematical model and an Iterated Local Search to formulate and solve the integration of OB, RP, and picker scheduling (sequence of the batches assigned to available pickers) considering

order due times to increase the order picking efficiency were proposed by van Gils et al. (2019a). In this case, SLA decisions are not optimized, instead three SLA policies are used to obtain a variety of initial assignments. Computational experience and a full factorial design were carried out on synthetic and real-world instances of an automotive firm to compare exact and heuristic solutions as well as the ILS performance. Results show significant improvements when the integrated problem is considered.

Wang et al. (2020) study the optimization of the SLAP in order to improve the OP distance (considering an S-shape routing policy to compute it). Authors propose a data-based approach that first assigns items to aisles and then items to racks. Subsequently, the initial assignment is improved by swapping pairs of items. They assume that only one type of products must be assigned to each location (do not mix or partition product types in racks), therefore, the number of items must be less or equal than the number of storage locations. Also, only one order can be fulfilled per route (no order batching is considered), and the picking cart capacity is unlimited. Extensive computational experience was provided on real-world data, solving instances with at most 4,000 products. Results show that the proposed solution algorithm outperforms other methods proposed in the literature.

Silva et al. (2020) show that the integration of the SLAP and OP activities may incur in significant improvements. They propose several non-linear programming models for four cases of integration, which were linearized. The aim is to minimize the total routing costs considering each product is assigned to exactly one location and each location can have no more than one product. Computational results showed that it is not practical to solve even the small instances to optimality in less than 7200 seconds (3-5 picks, 10-60 slots). Thus, they developed a General Variable Neighborhood Search metaheuristic to obtain good quality solutions in better computing times. Authors do not assume any specific warehouse system and apply four routing policies in their solution method.

Kübler et al. (2020) propose an iterative solution method that solves the integration of a dynamic SLA, the OB, and the PR for the OPP. The dynamic SLA is obtained by solving a MILP, which takes into account multiple periods and forecasting techniques to evaluate the location of products in the current and future time periods. The OB and the PR are solved jointly through a proposed heuristic based on Particle Swarm Optimization (PSO) with evolutionary computation. Both approaches, the dynamic SLA and the PSO, are combined to solve the integrated problem. The relocation of products to storage locations for future time periods is carried out considering forecasted demands. Authors performed computational experiments on generated instances with a number of orders from 50 to 200 (2-10 items/order lines per each customer order) and 6000 products to be initially located (1 product per each storage location) in a multi-block warehouse of 7200 storage spots, showing a good performance on those cases. Another work that proposed an approach to optimize the integration of the SLA, OB, and PR was proposed in Scholz and Wäscher (2017), where

a mathematical model and an ILS to solve the OBP and the OPP simultaneously, is provided. Several well-known exact/heuristic PR policies were compared to evaluate the performance of the ILS on a two-block warehouse. The SLA is initially determined by using the random and class-based policies. Assumptions such as not partitioning batches and only one product can be assigned to each storage location are considered. Results show that the ILS combined with an exact method for PR outperforms the ILS with known (heuristic) routing policies; however, for very large instances the latter are recommended given the limitations on resources and large sizes of real-world cases.

Chen et al. (2022) propose a mathematical model and a two-phase algorithm that integrates order batching, picking sequence, and last-mile vehicle routing in order to minimize operations and penalty costs. For the first two decisions, they use an aisle similarity clustering method and the S-shape policy, respectively. For the last-mile delivery, authors solve a TSP with Time Windows. The results show that the integration achieves a positive impact in costs savings and good-quality indicators. It reinforces the benefit of solving integrated problems simultaneously during the optimization.

Celik et al. (2022) study the OPP considering storage replenishment operations to ensure available stock of products that will be retrieved during the order picking activity. Authors adapt mathematical models and solution methods of the Inventory Routing Problem to solve the newly defined problem. The aim is to determine the best replenishment plan that minimizes the total picking travel time and ensure the availability of items. They carried out extensive computational tests and provided several comparison with respect to other routing policies showing the effectiveness of their proposed methods.

Wagner and Mönch (2023) study the improvement of the picking activity when order batching decisions, considering an heterogeneous fleet of vehicles, are integrated. They proposed an MILP formulation and a Variable Neighborhood Search (VNS) metaheuristic to solve a case study for a company. Authors carry out extensive computational tests on large pseudo and real-world instances and compare their results with a current heuristic method that the company uses. They confirm that their proposed solution methods are efficient and, by integrating order batching decisions, companies can obtain better improvements in their warehouse operations. Another VNS provided in the literature that studies the integration of order batching decisions to improve the order picking costs is the one proposed by Albareda-Sambola et al. (2009), who made a comparison between their proposed VNS and several routing, storage location, and batching policies, providing an ANOVA statistic analysis considering these factors and four different warehouse designs.

In this work, we study an OPP that integrates, not only the storage location of products and the sequence of picker routes but also order batching decisions, simultaneously. We proposed a mathematical model and an IGLS metaheuristic to obtain good quality solutions of a real-world case study of a Mexican retail firm. The mathematical model is formulated as a mixed-integer linear program that can be used to obtain feasible solutions for small (but still real-world size) instances

using a general-purpose solver and, to the best of our knowledge, it can be considered the first mathematical formulation that integrates and optimize these three main decisions. On the other hand, the proposed IGLS provides a good alternative to solve the problem, without considerably reducing the quality of solutions, in reasonable computing times for large-scale instances. Moreover, most of the previous works mentioned above, specifically those that study the integration of these three decisions, consider several assumptions that are tackled in this work, e.g., that only one product type can be assigned to each storage location (we consider a maximum affinity value to determine what product types can be assigned together), the absence of due times for customer orders (our problem considers a maximum processing time for each one), and the use of an exact method to obtain each final picker route embedded in the proposed heuristic approach, which allows not to depend completely on the type of layout to be used to obtain good routes, giving the opportunity to use the same approach regardless the design of the warehouse.

3 Problem definition and mathematical model

3.1 Problem definition

Let L be the set of storage locations or racks. The problem can be represented by a directed graph $G = (N, A)$, where $N = L \cup \{0\}$ is the set of nodes including the depot (represented by the 0 node) from which pickers depart and return after collecting the customer orders; A is the set of available arcs between each pair of storage locations, where "arc" means any direct path, which exists between two adjacent or nearby racks, that does not involve passing through another intermediate rack. Let P, O, W, and B the set of products, customer orders, pickers, and batches, respectively. Each unit of product $p \in P$ has a volume V_p and a deterministic demand $D_{po} \geq 0$ required for a given order $o \in O$. Let Q^L and Q^W be the corresponding homogeneous capacity of racks and picker carts, respectively. All pairs of products $(p, q) \in P$, assigned to the same rack, must not exceed a value of $\gamma \in [0, 1]$, defined as the maximum affinity dissimilarity level between two selected products to be located in a rack. Furthermore, each order $o \in O$ must be collected before a maximum processing time $T_o^{\max}$. Finally, a cost c_e and a traveling time t_e, per each arc $e \in A$, are defined.

Then, the aim of the order picking problem, which considers storage location, order batching, and routing decisions, called from now on the PSB, is to determine where to locate products into racks, what orders to locate in a given batch, and in what sequence to perform the picking routes in such a way as to minimize the total picking cost.

Some other assumptions must be considered:

- Layout aisles are wide enough, so pickers carts can go back and forth through the aisles.

- No splitting of customer orders is allowed.
- The current warehouse layout used by the firm is a multi-block layout; however, the company is interested in analyzing the performance of the proposed algorithm considering a Fishbone layout (Dukic and Opetuk, 2012) as the firm seeks to migrate to this design in the future. Therefore, we decide to use an optimal policy to perform the picking routes independently of the layout design.

3.1.1 Product affinity

One important criterion for the company is the affinity of products to be located in the racks. This affinity can be measured in several manners (Kofler et al., 2014). However, the company has its own procedure to compute the affinity between two products p and q, which is composed of three main steps.

- **Step 1.** Each available product p has an affinity code assigned to it, which corresponds to an alphanumeric value for each product category defined by the company (see Table 1). Products (Stock Keeping Units, SKUs) from the same category have the same affinity code (column 2). These codes are sorted alphabetically assigning an ordinal value for each unique category in such a way that products with the same affinity code have the same ordinal value (column 3).

Table 1: Affinity values

Product	Affinity code	Value
P2	0A10800	1
P1	0A13400	2
P6	0A13401	3
P4	0A13401	3
P3	0F13300	4
P5	0G11200	5

- **Step 2.** Given the data provided above, the affinity value for any pair of products p and q, is computed as follows.

$$\texttt{Aff}(p,q) = \frac{|\texttt{Val}_p - \texttt{Val}_q|}{n_{cat}},$$

where $\texttt{Val}_p$ and $\texttt{Val}_q$ are the corresponding individual affinity values provided in the third column of Table 1, and n_{cat} is the number of product categories. Therefore, $\texttt{Aff}(p,q)$ resulted in a value [0,1], where 0 represents a pair of products of identical category and 1 represents

two completely unrelated product categories. This affinity value is computed for each pairwise (p, q) of products, and an affinity matrix is generated (Table 2).

Table 2: Affinity matrix

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
P1	0	**0.17**	0.33	**0.17**	0.50	**0.17**
P2	-	0	0.50	0.33	0.67	0.33
P3	-	-	0	**0.17**	**0.17**	**0.17**
P4	-	-	-	0	0.33	**0.00**
P5	-	-	-	-	0	0.33
P6	-	-	-	-	-	0

Considering this matrix it is possible to determine which products can be located in a given rack taken into account a maximum value of affinity (dissimilarity) allowed, γ.

3.2 Mathematical model

In this section, we introduce a mixed-integer linear programming model for the PSB problem. This model considers decision variables related to the main three activities, namely, product storage location (including location and quantity), order batching, and picker routing. The main data, sets, and variables of this model are the following.

Sets and parameters

- L, A, P, O, W, B: Set of storage locations (racks), available arcs (connections between racks), products, orders, pickers, and batches, respectively.
- $G = (N, A)$: Directed graph composed of the set of racks plus the depot, $N = L \cup \{0\}$ and the set of available arcs A.
- A_l^+ / A_l^-: Set of arcs that enter/exit to/from a given rack $l \in L \cup \{0\}$, where $A_l^+, A_l^- \subseteq A$.
- V_p: Volume of product $p \in P$
- γ: Maximum value [0,1] of dissimilarity (affinity) between products that can be located in the same rack.
- Q^L, Q^W: Capacity (volume) of racks and carts, respectively.
- D_{po}: Demand of product $p \in P$ for customer order $o \in O$.
- $T_o^{\max}$: Maximum processing time of an order $o \in O$.
- c_e, t_e: Picking cost and traveling time for each arc $e \in A$.

Decision variables for the storage location of products

- y_p^l, binary variable equal to 1 if product $p \in P$ is assigned to location $l \in L$; and equal to 0 otherwise.
- a_p^l, nonnegative integer variable representing the number of units of product $p \in P$ assigned to location $l \in L$.

Decision variables for the order batching

- b_{oi}^w, binary variable equal to 1 if order $o \in O$ is assigned to batch $i \in B$ and collected by picker $w \in W$; and equal to 0 otherwise.
- τ_{oi}^{ew}, binary variable equal to 1 if arc $e \in A$ is traversed by picker $w \in W$ servicing order $o \in O$ assigned to batch $i \in B$; and equal to 0 otherwise.

Decision variables for the order picking

- z_{poi}^{lw}, binary variable equal to 1 if product $p \in P$, located at rack l, which comes from order $o \in O$ assigned to batch $i \in B$, is serviced by picker $w \in W$; and equal to 0 otherwise.
- r_e^{iw}, binary variable equal to 1 if arc $e \in A$ is traversed by picker $w \in W$ to collect orders of batch $i \in B$; and equal to 0 otherwise.
- $f_e^{iw} \geq 0$, non-negative continuous variable representing the cart load used by picker $w \in W$ when traversing arc $e \in A$.
- q_{poi}^{lw}, number of units of product $p \in P$ collected by picker $w \in W$ at rack $l \in L$, for any order $o \in O$ that belongs to batch $i \in B$.

Then, the PSB is defined as follows.

$$\min \quad \sum_{i \in B} \sum_{w \in W} \sum_{e \in A} c_e r_e^{iw} \tag{1}$$

$$\text{s.t.} \quad \sum_{l \in L} y_p^l \leq \frac{\sum_{o \in O} V_p D_{po}}{Q^L} \qquad p \in P \tag{2}$$

$$\sum_{l \in L} a_p^l \geq \sum_{o \in O} D_{po} \qquad p \in P, \tag{3}$$

$$V_p a_p^l \leq Q^L y_p^l \qquad p \in P, l \in L \tag{4}$$

$$\sum_{p \in P} V_p a_p^l \leq Q^L \qquad l \in L \tag{5}$$

$$y_p^l + y_q^l \leq 1 \qquad p, q \in P | \texttt{Aff}(p,q) > \gamma, l \in L \tag{6}$$

$$z_{poi}^{lw} \leq y_p^l \qquad w \in W, p \in P, o \in O, i \in B, l \in L \tag{7}$$

$$V_p q_{poi}^{lw} \leq Q^W z_{poi}^{lw} \qquad p \in P, o \in O, i \in B, w \in W, l \in L \tag{8}$$

$$\sum_{l \in L} \sum_{o \in O} \sum_{p \in P} V_p q_{poi}^{lw} \leq Q^W \qquad i \in B, w \in W \tag{9}$$

$$\sum_{i \in B} \sum_{w \in W} \sum_{o \in O} q_{poi}^{lw} \leq a_p^l \qquad p \in P, \in L \tag{10}$$

$$\sum_{l \in L} \sum_{w \in W} \sum_{i \in B} q_{poi}^{lw} = D_{po} \qquad p \in P, o \in O \tag{11}$$

$$\sum_{e \in A_l^+} r_e^{iw} \leq 1 \qquad l \in L, i \in B, w \in W \tag{12}$$

$$\sum_{w \in W} \sum_{e \in A_0^+} r_e^{iw} \leq |W| \qquad i \in B \tag{13}$$

$$\sum_{e \in A_l^+} \sum_{i \in B} \sum_{w \in W} r_e^{iw} = \sum_{e \in A_l^-} \sum_{i \in B} \sum_{w \in W} r_e^{iw} \qquad l \in L \tag{14}$$

$$\sum_{e \in A_l^+} f_e^{iw} = \sum_{e \in A_l^-} f_e^{iw} - \sum_{o \in O} \sum_{p \in P} V_p q_{poi}^{lw} \qquad w \in W, i \in B, l \in L \tag{15}$$

$$f_e^{iw} \leq Q^W r_e^{iw} \qquad e \in A, i \in B, w \in W \tag{16}$$

$$\sum_{i \in B} \sum_{w \in W} \sum_{e \in A_0^+} f_e^{iw} = 0 \tag{17}$$

$$b_{oi}^w \leq \sum_{l \in L} \sum_{p \in P} z_{poi}^{lw} \leq M b_{oi}^w \qquad o \in O, i \in B, w \in W \tag{18}$$

$$\sum_{w \in W} \sum_{i \in B} b_{oi}^w = 1 \qquad o \in O \tag{19}$$

$$r_e^{iw} + b_{oi}^w - 1 \leq M \tau_{oi}^{ew} \qquad o \in O, w \in W, i \in B, e \in A \tag{20}$$

$$\tau_{oi}^{ew} \leq r_e^{iw} \qquad o \in O, w \in W, i \in B, e \in A \tag{21}$$

$$\tau_{oi}^{ew} \leq b_{oi}^w \qquad o \in O, w \in W, i \in B, e \in A \tag{22}$$

$$\sum_{i \in B} \sum_{e \in A} t_e \tau_{oi}^{ew} \leq T_o^{\max} \qquad o \in O, w \in W \tag{23}$$

$$y_p^l \in \{0,1\}, a_p^l \in \mathbb{Z}^+ \qquad p \in P, l \in L \tag{24}$$

$$z_{poi}^{lw} \in \{0,1\}, q_{poi}^{lw} \in \mathbb{Z}^+ \qquad p \in P, o \in O, w \in W, i \in B, l \in L \tag{25}$$

$$r_{ie}^w \in \{0,1\}, f_{ie}^w \geq 0 \qquad e \in A, i \in B, w \in W \tag{26}$$

$$b_{oi}^w \in \{0,1\} \qquad o \in O, i \in B, w \in W \tag{27}$$

$$\tau_{oi}^{ew} \in \{0,1\} \qquad o \in O, i \in B, w \in W, e \in A \tag{28}$$

$$h_o \in \{0,1\} \qquad o \in O \tag{29}$$

The objective function (1) minimizes the picking routing cost. Constraints (2) specify the maximum number of locations where a product type can be assigned. Constraints (3) determine that the quantity of product p assigned to locations must satisfy at least the demand of that product. Constraints (4)-(5) guarantee that the volume of the quantity of product delivered at each location must not exceed its capacity. Constraints (6) limit an affinity level between two product types

assigned to locations, that is, two products that are not affine enough cannot be placed in the same rack. Constraints (7) establish that if product p is not assigned to rack l there can not be any picker visit for any batch and any order Constraints (8)-(9) determine that the product volume assigned to a picker must not exceed the cart capacity volume. Constraints (10) limit the amount of product to pick up to at most the quantity available at location l. Constraints (11) establish that the total quantity of product p, for a given order, must be equal to the quantity required for that customer order. Constraints (12)-(14) are the node degree and the flow conservation constraints. Constraints (15)-(17) represent the load of vehicles at a given traversed arc e, this load must not exceed the cart capacity, and must be 0 when the picker return to the depot. Also, these constraints avoid subtours. Constraints (18)-(23) establish that the time required to collect an order must be less than a maximum processing time for that given order. Constraints (24) - (29) represent the domain of variables.

Computational complexity: The PSB is an order picking problem that considers assignment, batching, and routing decisions. We can demonstrate that the PSB is $\mathcal{NP}$-hard as follows. First, PSB is clearely in $\mathcal{NP}$ since feasibility can be verified in polynomial time. Now, we will show that the Traveling Salesman Problem (TSP), a well-known $\mathcal{NP}$-hard problem, is polynomially reducible to the PSB. For this, we consider the particular case of PSB when the number of available racks is equal to the number of products $|L| = |P|$ (all products have the same volume $V_p = 1$), there is only one customer order $|O| = 1$ that requires only 1 unit per product of the list $D_{po} = 1$, $p \in P$; one picker is available to perform the route and, since there is only one customer order and it is not possible to partition it, only one batch must be performed during the day, so $|W| = |B| = 1$; racks and carts capacities are $Q^L = 1$ and $Q^W = \infty$, respectively. There is no possibility to assign more than one product at each rack since $\gamma = 0.0$ and racks capacity does not allow it; also, $T_o^{\max} = \infty$ (there is enough time to process the order). For this special case, the PSB is reduced to find the minimum tour that visits all racks in $L \cup \{0\}$ (departing from the depot) resulting in a TSP, which in turn is $\mathcal{NP}$-hard (Gutin and Punnen, 2007). Therefore, the PSB is at least as hard as the TSP and, given the size of the real-world instances, we propose a metaheuristic to obtain good quality solutions in reasonable time.

4 The Iterated Greedy Local Search for the PSB

Since obtaining the optimal solution for real-world problems in this context can be a hard task for exact methods, we propose an Iterated Greedy Local Search (Lourenço et al., 2003) metaheuristic for the PSB (IGPSB). The proposed IGPSB (see Algorithm 1) starts with an initial feasible solution generated by solving an MILP subproblem for the storage location decisions, and applying a first-fit heuristic to obtain the batch of orders to be served by pickers. Finally, a TSP for each picker route (one batch per route) is solved. Subsequently, this initial solution is improved by two local search

(LS) procedures. Afterwards, a second phase is performed, where the initial solution is partially *destroyed* and greedily *reconstructed* to be, then, improved by the corresponding LS. Therefore, if the solution value $f(s)$, obtained after reconstructing solution s, is better than the one of the current solution s', then s replaces s'. Furthermore, if $f(s)$ is better than the incumbent value $f(s^*)$, then, the new incumbent is updated. Otherwise, if $f(s)$ is worse than $f(s')$, a procedure to evaluate if s should replace s', is performed. The IGPSB stops when a maximum number of iterations or a maximum computing time is met.

Algorithm 1 IGPBS

Input: : L (racks), P (products), W (pickers), O (orders), γ (max. affinity value), `%Dest` (% solution destruction), β (factor for worse solution acceptance).

Output: s^*: Best solution found.

▷ **Initial solution**

1: $s \leftarrow$ `getInitialSolution`(L, P, W, O, γ) ;
2: $s \leftarrow$ `LocalSearch`(s)

▷ **Iterative phase**

3: $s' \leftarrow s$, $s^* \leftarrow s$
4: **while** stopping criteria not met **do**
5: $s'' \leftarrow$ `Destruction`$(s,$ `%Dest`$)$
6: $s \leftarrow$ `reConstruction`$(s'',$ `%Dest`$)$
7: $s \leftarrow$ `LocalSearch`(s)
8: **if** $f(s) < f(s')$ **then**
9: $s' \leftarrow s$
10: **if** $f(s') < f(s^*)$ **then**
11: $s^* \leftarrow s'$
12: **end if**
13: **else**
14: $\{s, s'\} \leftarrow$ `evaluateWorseSolution`(s, s', s^*, β)
15: **end if**
16: **end while**
17: **return** s^*

We provide more details for the main IGPSB components below.

Phase 1: Generation of an initial solution

The first phase of the IGPSB requires to start with an initial feasible solution, and to obtain it, we propose a simple constructive heuristic composed of three steps. This initial solution is then

improved by applying two local searches we have developed for this problem.

Constructive heuristic

The constructive heuristic proposed for the IGPSB is composed of three main steps. In the first step, the storage location of products to racks is determined, while in the second step, the order batching is obtained. Once these two decisions are performed, the third step consist of solving a TSP for each picker route.

- *Step 1:* To obtain the storage location decisions, the following subproblem, which considers constraints (2)-(6) related to this activity in the original formulation, is solved.

  Let $\bar{y}_l \in \{0,1\} = 1$ if rack l contains any product p; otherwise, 0. Then, the subproblem that obtains a distribution plan to locate products into racks is as follows.

$$\min \quad \sum_{l \in L} c_{0l} \bar{y}_l \tag{30}$$

$$\text{s.t.} \quad \bar{y}_l \leq \sum_{p \in P} y_p^l \leq M \bar{y}_l \qquad l \in L \tag{31}$$

$$(2) - (6)$$

$$(24)$$

$$\bar{y}_l \in \{0,1\} \qquad l \in L \tag{32}$$

  The objective function (30) aims to minimize the distance between any used rack and the depot (a closest space policy). The new set of constraints (31) enables or disables the variables $\bar{y}_l$ depending on whether or not a product unit is assigned to each storage location l. Constraints (32) delimit the domain of the new decision variables.

- *Step 2:* We adapt a first-fit algorithm (Lodi et al., 2002) to obtain the order batching plan. The main steps are the following (see Figure 1):

  - Sort customer orders in a non-decreasing order according to their maximum processing time $T_o^{\max}$. The reasoning behind this is to locate in the first batches the orders that have less $T_o^{\max}$ available to be served as soon as possible by pickers (priority policy).
  - Select the first order of the list and assign it to the first batch. Then, select the second one and assign it to the first available batch where it can fit without exceeding the cart capacity or the processing time $T_o^{\max}$ for any order of the corresponding batch. Otherwise, *open* a new batch and assign the current order. Since each picker can only perform one batch per route; then, $|B| = |W|$.

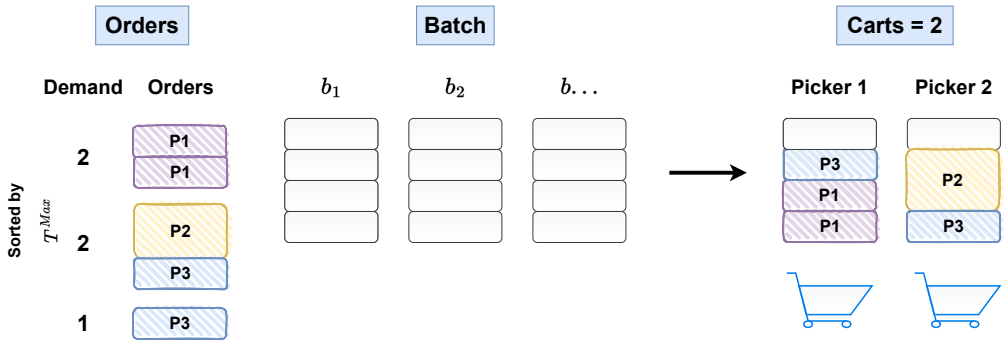


Figure 1: Offline First-Fit algorithm for the PSB

- *Step 3:* Solve a TSP for each picker route (batch) taken into account the current storage locations assignment. As we are working on an incomplete graph, i.e., there is no exists a direct path to connect several pair of locations (l_1, l_2), we simplify this by transforming this graph into a complete one using the the Floyd-Warshall algorithm (Floyd, 1962). Using this transformed graph, the final batching, and the storage location assignment found in previous steps, we apply the Concorde TSP solver to obtain the sequence of routes, which performs relatively fast per each batch.

Local search procedures

In order to improve a given solution s, we proposed two LS procedures. The first one improves the selection of storage locations and the second one attempt to improve the order assignment from the current batches. The LS procedures are explained below.

Storage location LS. The LS proposed for the storage location decisions, improves a solution s by applying a sequential neighborhood search scheme, which performs a 1-1 move that consist in selecting a rack l_1 from s and a rack l_2, which does not belong to s, and swap them (if feasible and if solution is improved), moving all product assigned from one rack to another (Figure 2).

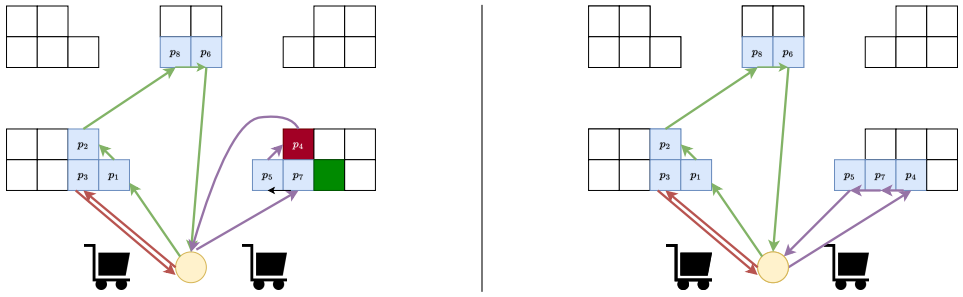


Figure 2: LS for storage location decision

Order batching LS. The second LS (Figure 3) improves the order batching by exploring a neighborhood where 1-1 moves are applied. In this LS, one order o_1 from batch b_1 is moved to another batch b_2 (not necessarily of a different picker). If solution remains feasible (cart capacity and processing times are not exceeded) and if its cost improves, the move is accepted.

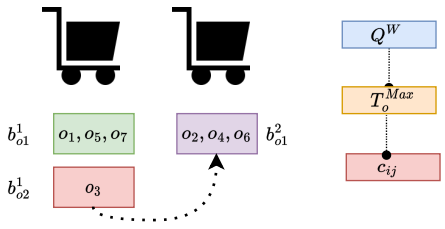
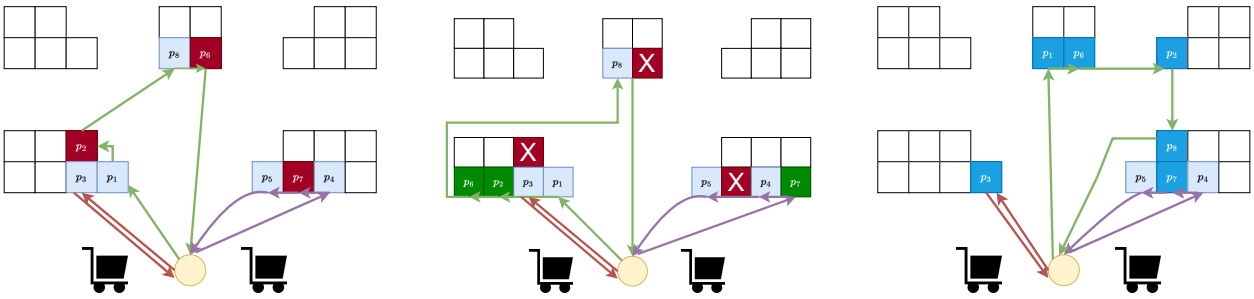


Figure 3: LS for order batching decision

Phase 2: Iterative phase

Once the initial solution is obtained, the second phase of the IGPSB starts by partially *destroying* the solution s, and then *reconstructing* it to obtain a new one, which will be improved by applying the LS procedures explained previously. If the reconstructed solution is better than the current one, this is accepted and replaces the current solution (and the incumbent one if it is the case). If the new solution is worse than the current accepted solution, a criterion of acceptance is evaluated to determine if it will replace the current one in the next iteration. The IGPSB stops when a stopping criteria is met. Figure 4 shows the main steps of this second phase of the IGPSB, and each component of this phase is explained with more detail below.



(a) Destruction (b) Reconstruction (c) Local Search

Figure 4: The main steps of the iterative phase of the IGPSB

Destruction-Reconstruction. During the destruction step, a given solution is perturbed by removing a proportion of it in order to be reconstructed greedily in the next step. For the studied problem, we focused the destruction on the selected racks where product is located. This procedure is as follows.

- **Destruction.** Select randomly a proportion (measured by parameter `%Dest`) of the total

number of racks assigned for s, and remove them resulting in a partial solution s''.

- **Reconstruction.** For each rack $l \in L \setminus s''$, compute the cost between depot and l. The rack l with the cheapest cost (shortest distance) is selected to be a new element of the current s''. Once the number of elements added corresponds to the proportion `%Dest`, the solution is complete.

Improvement. In this step, the solution reconstructed is improved by applying the proposed LS procedures mentioned above.

Selection. Once the solution s is improved, the algorithm must decide if this new improved solution will be accepted or not (Algorithm 1, lines 8-15). If the solution value $f(s)$ is better than the one of the current accepted solution s', then s replaces s'. Furthermore, if $f(s)$ is better than the incumbent s^*, then, the updated s' will replace s^*. On the other hand, if $f(s)$ is worse than $f(s')$, s must be evaluated to determine if it will be accepted (see Algorithm 2). For this, a *Gap* value is computed and multiplied by a factor β to determine a threshold τ, which will be used to decide if a worse solution will replace s' (increasing the margin of acceptance of poorer quality solutions in subsequent iterations). Also, a counter aw (naw) and a variable faw ($fnaw$), that accumulate the number of iterations and the solution value of the accepted (non-accepted) worse solutions (respectively), are defined to update the threshold τ in later iterations.

Algorithm 2 `evaluateWorseSolution`(s, s', s^*, β)

Input: : s, s', s^*, β ▷ New, current, and best solution, respectively; factor for poor-quality solution acceptance.

Output: s, s' ▷ Accepted/updated solutions.

$Gap \leftarrow \frac{|f(s)-f(s^*)|}{f(s^*)}$
if $iter = 0$ **then**
  $\tau \leftarrow \beta \times Gap$
end if
if $Gap \leq \tau$ and $Gap > 0$ **then**
  $s' \leftarrow s$ ▷ s is accepted, s' is updated.
  $faw \leftarrow faw + f(s)$
  $aw \leftarrow aw + 1$, $naw \leftarrow 0$
else
  $s \leftarrow s'$ ▷ s is not accepted.
  $fnaw \leftarrow fnaw + f(s)$
  $naw \leftarrow naw + 1$, $aw \leftarrow 0$
end if
if $naw = 0.1 \times iterIGLS_{\max}$ **then** ▷ After naw non-accepted worse solutions
  $Gap \leftarrow \frac{|\frac{fnaw}{naw}-f(s^*)|}{f(s^*)}$
  $\tau \leftarrow \beta \times Gap$
  $fnaw \leftarrow 0$, $naw \leftarrow 0$
end if
if $aw = 0.1 \times iterIGLS_{\max}$ **then** ▷ After aw accepted worse solutions
  $Gap \leftarrow \frac{|\frac{faw}{aw}-f(s^*)|}{f(s^*)}$
  $\tau \leftarrow \beta \times Gap$
  $faw \leftarrow 0$, $aw \leftarrow 0$
end if
return $\{s, s'\}$

In the following section, we perform several computational experiments in order to evaluate the proposed solution algorithms.

5 Computational experience

Computational experiments were conducted on a Workstation HP Intel(R)-Xeon(R) at 3.5GHz with 64 GB RAM (Win 10 Pro, 64 bits) and a processor with 6 cores. The solution algorithms have

been implemented in C++ and, in the case of the formulations, we use ILOG Concert Technology API (CPLEX 20.1), setting to only one thread per optimization.

For all tested instances, we evaluate the quality of the solution produced by a given method M, i.e. s^M, by computing the *Relative Percentage Deviation* (RPD) of its objective function value, $f(s^M)$, with respect to the best-known solution, $f(s^*)$. The RPD is defined as

$$\text{RPD} = \frac{f(s^M) - f(s^*)}{f(s^*)} \times 100\%. \tag{33}$$

The layout used for the tests is a fishbone layout with at most 241 storage locations (see Figure 5), where the black squared is the depot from where pickers depart. This layout is used for the real-world instances subset explained below. Other two reduced layouts with up to 58 and 121 racks (including depot), extracted from the main one, are used for pseudo-real instances.

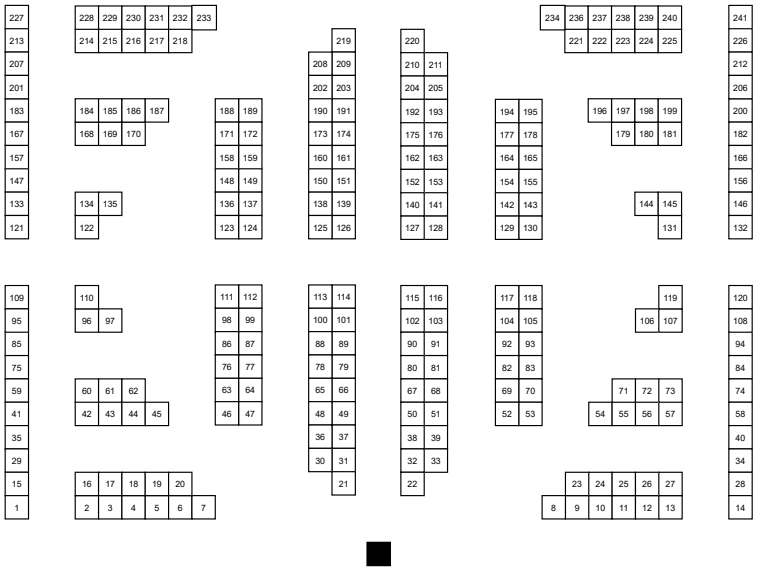


Figure 5: Fishbone layout

All tested instances, as well as any complementary material and obtained results, are publically available at https://github.com/DianaHuertaM/IGPSB.git.

This section is organized as follows. In Section 5.1, we provide a description of the generated instances. In Section 5.2, we provide a preliminary analysis of the impact of the maximum level of affinity γ for the storage location assignment decisions. Section 5.3 summarizes the results of the calibration tests for the IGPSB parameters. Finally, Section 5.4 describes the final performance.

5.1 Instances

We have created two sets of instances to asses the proposed solution method and determine its performance. The description is provided below.

- **Calibration set:** This set is composed of 40 pseudo-real instances with size $r = 30$, 60, 100, and 500 order lines (where an order line represents the information of a specific item required by a given customer); the number of orders, pickers, and products corresponds to $|O| = |W| = 5$ and $|P| = 30$ (for $r = 30/60$ instances), and $|O| = |W| = 10, 20$ and $|P| = 50$ (for sizes $r = 100/500$, respectively). Products and their volume (measured in cm^3) were selected randomly from a general list provided by the company; the number of units ordered for a specific product p in a given order o is set to $D_{po} = 1$ units. Vehicle and rack capacities are set to $Q^W = 126000$ cm^3 and $Q^L = 216000$ cm^3, respectively. The maximum processing time required for a given customer order X_o is computed as $T_o^{\max} = \rho|X_o|$, where $|X_o|$ corresponds to the number of products that belong to that order and ρ determines an approximate number of seconds needed to handling each product during the picking activity; for the tests, we use a value $\rho = 100$ seconds.

- **Performance set:** This set is composed of 60 pseudo-real instances (subset A) and 14 real-world instances (subset B). We explain both sets in more detail. Vehicle and rack capacities are set to $Q^W = 1680000$ cm^3 and $Q^L = 3192000$ cm^3, respectively.

  - **Subset A:** This subset corresponds to 30 small instances (10 per size) with $r = 30$, 60, and 100 order lines; number of orders, pickers, and products of $|O| = |W| = 3, 5$ and $|P| = 20$; and 30 large instances (10 per size) with $r = 1000$, 3000, and 5000 order lines; number of orders, pickers, and products of $|O| = |W| = 150, 250$ and $|P| = 100$. The number of units ordered for a specific product was generated using a uniform distribution on the interval $\sim U[1, 3]$, and the products for each order were selected randomly for a given list of products.
  - **Subset B:** This set is composed of 13 instances obtained from the data provided by the company. The size of them varies from 362 to 8197 order lines, number of pickers $|W| = 1-44$, number of orders $|O| = 1-654$, and number of products of $|P| = 106-4252$. The $T_o^{\max}$ value for a given order $o \in O$ was obtained by computing the following formula:

    $$T_o^{\max} = t_{\text{tot}} + t_{\text{rem}}.$$

    where $t_{\text{tot}} = t_{\text{fin}} - t_{\text{ini}}$ (difference between the initial and final picking time in the real-word case), and $t_{\text{rem}} = 1800 - (t_{\text{tot}} \mod 1800)$, which is the remaining time to complete the nearest half hour.

To easily identify the main characteristics of both sets, we have named each instance as Inst$\mathbb{A}$r$\mathbb{B}$o$\mathbb{C}$p$\mathbb{D}$, where $\mathbb{A}$ is the distribution center ID from which the information was extracted, $\mathbb{B}$ is the number of order lines (instance size), $\mathbb{C}$ represents the number of orders, and $\mathbb{D}$ the number

of products.

In the next section, we analyze the impact of the parameter γ in the efficiency of the proposed IGPSB.

5.2 The impact of the affinity value γ

The γ value, which represents the maximum affinity (or dissimilarity) allowed among product categories assigned to a given rack, is a grouping criterion that may reduce significantly the number of required racks resulting in a potential improvement in picking costs since it decreases the number of places to be visited by the pickers.

As a preliminary analysis to determine its impact in the solution, we evaluate several values of γ. We set these values to $\gamma = 0.0, 0.10, 0.20, 0.30$, where a value of 0.0 corresponds to allow *only identical item categories* in each rack; a value near 1.0, indicates that *items assigned to racks may differ as much as categories* exist. For these tests, we use the *calibration set* and solve only the initial phase of Algorithm 1 (lines 1–2) as this parameter is only used in the MILP for the storage location assignment (SLA) subproblem of the `getInitialSolution` procedure. This MILP (30)-(32) is solved using the CPLEX solver considering a maximum optimization time of 600 s. The final results are presented in Figures 6-8.

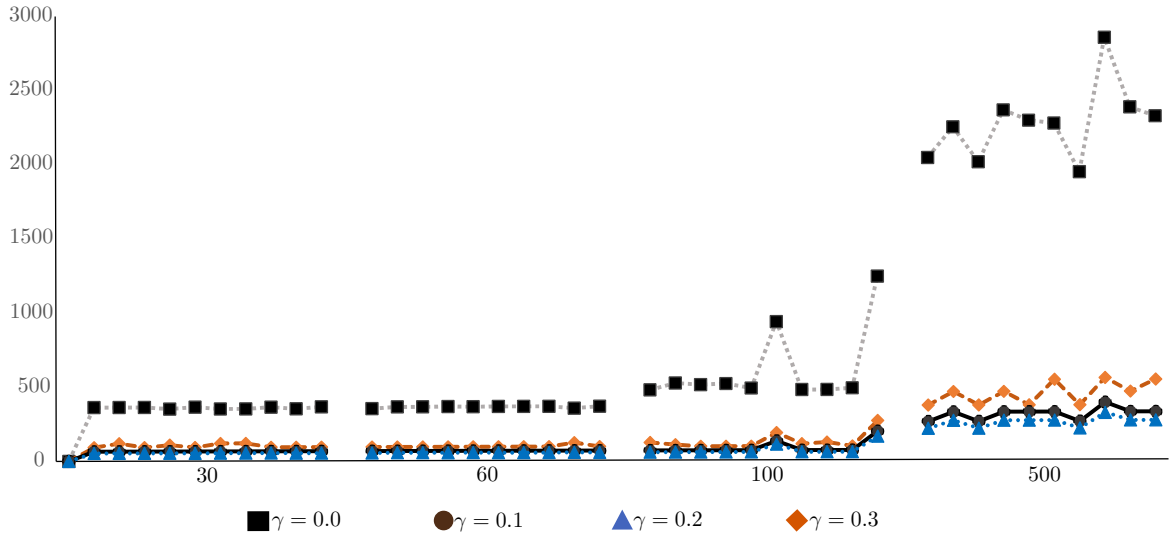


Figure 6: Picking cost per affinity level

Figure 6 shows the picking cost per each instance and each affinity value. We can observe that, when only identical product categories can be assigned in a same rack ($\gamma = 0.0$), the picking cost increases, and it is more notorious for larger instances. The reason of this is because instances consider a list of 30 different products for small sizes and 50 products for large instances, and most

of these products belong to a different category (only some of them belong to the same category). Then, it is required to use more racks in order to locate all the items in a specific storage location as it is not allowed to place other product categories in a same rack.

In Figure 7, we can observe that the number of racks used to assign items is greater when $\gamma = 0$ (only allowed identical product categories are allowed). On the other hand, when γ increases, the number of required storage locations decreases and the picking costs decrease significantly as this activity requires less number of racks to visit during the routing.

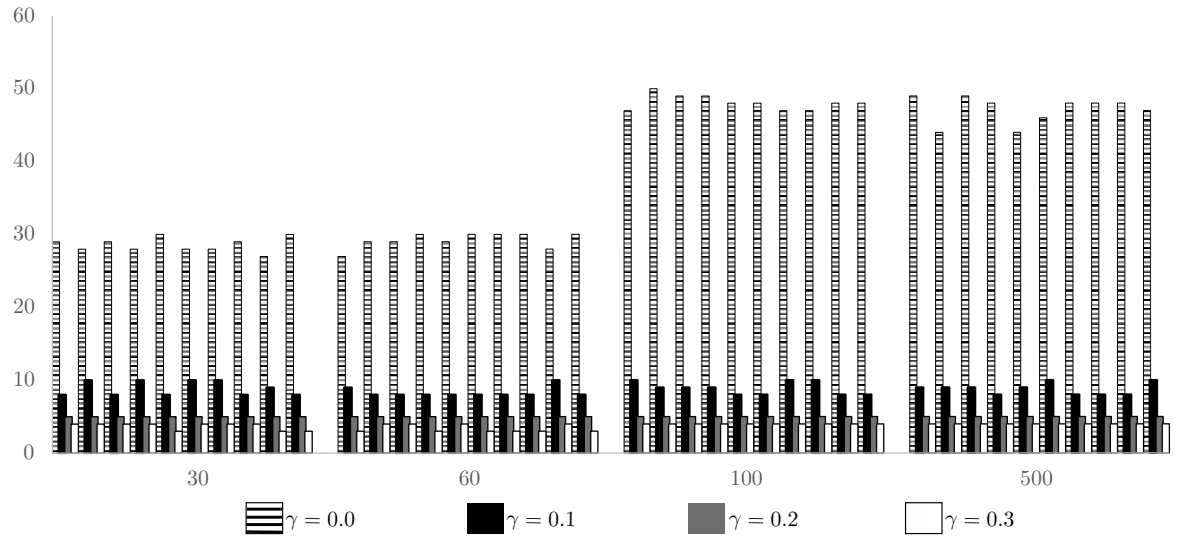


Figure 7: Number of racks required per each affinity level

Finally, in Figure 8, the proportion of computational time required for the constructive heuristic to find the best feasible solution is reported. It can be observed that the most time consuming option (in all the cases) is presented when $\gamma = 0.10$. Comparing the results shown in Figures 6-8, $\gamma = 0.20$ outperforms the remaining options as it obtained a better trade-off between picking costs and computing times. Furthermore, the similarity between products located at the the same rack is not affected more than the necessary.

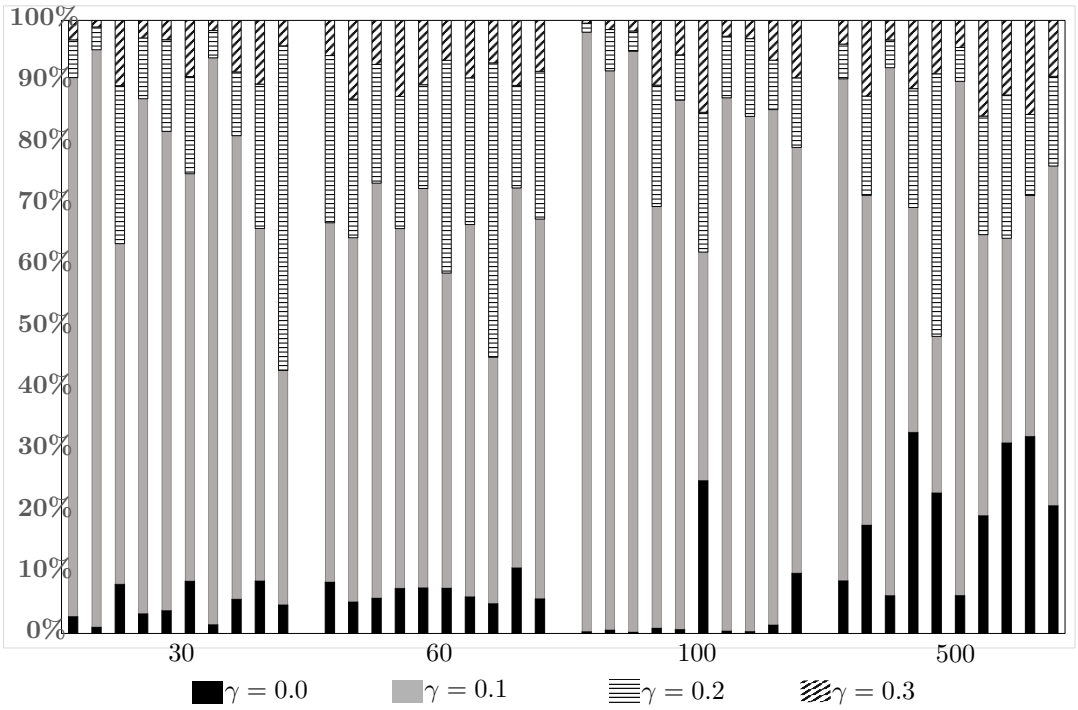


Figure 8: Proportion of computing times required using different affinity levels

We can briefly conclude that by allowing flexibility when placing products on the racks, order picking costs can be reduced considerably and a better use of racks space is obtained without modifying significantly the affinity between products assigned to them. Therefore, we propose to use a value of $\gamma = 0.2$ for the remaining experiments of the next sections.

5.3 Calibration tests

In order to better exploit the efficiency of the algorithm, we have performed two required parameter calibrations for the IGPSB: the percentage of elements to be *destroyed* (`%Dest`) from the solution and the value of the parameter β, which is used to compute the threshold of acceptance of poorer quality solutions. For these experiments, we use the corresponding set of calibration instances and fix the affinity value to $\gamma = 0.20$ (as it obtained better results in the preliminary tests), the time limit to solve the MILP used to construct an initial feasible SLA is set to 600 s, and the stopping criteria for the IGPSB are fix to $\texttt{iterIG}_{\max} = 100$ iterations and `MaxOptTime` = 3600 s.

5.3.1 Calibration of percentage of solution destruction parameter

The first parameter to calibrate is `%Dest`, the number of elements to remove from a solution during the `Destruction` step (line 5, Algorithm 1). We consider three factor levels, `%Dest` = 30, 50, and

70, which mean that, for a given solution s, it is required to remove (randomly) 30/50/70% of the total number of selected racks, which later will be reassigned (greedily) in the same proportion during the `reConstruction` procedure. For these specific experiments, we also fix the parameter $\beta = 0$, which corresponds to the option of not accepting poor-quality solutions during the IGPSB. Since the `Destruction` procedure applies a random selection of solution elements to remove, we consider 4 repetitions of the IGPSB for each tested instance.

Table 3: Average RPD and computing times by removing a % of the solution in the IGPSB

Instance	`%Dest`	**RPD**			**Time(s)**		
		Min	**Avg**	**Max**	**Min**	**Avg**	**Max**
r30	**30**	0.00%	0.00%	0.00%	200.5	272.1	363.6
r60		0.00%	0.00%	0.00%	205.0	276.9	369.9
r100		0.00%	0.02%	0.08%	339.7	441.0	533.4
r500		0.00%	5.18%	16.60%	1855.4	2384.6	2940.7
Avg		**0.00%**	**1.30%**	**4.17%**	**650.1**	**843.7**	**1051.9**
r30	**50**	0.00%	0.00%	0.00%	323.9	444.5	646.1
r60		0.00%	0.00%	0.00%	341.0	428.3	554.2
r100		0.00%	1.11%	4.35%	419.5	551.4	676.5
r500		4.65%	16.42%	35.24%	2364.5	2977.8	3399.2
Avg		1.16%	4.38%	9.90%	862.2	1100.5	1319.0
r30	**70**	0.00%	0.00%	0.00%	363.6	646.1	511.9
r60		0.00%	0.00%	0.00%	369.9	554.2	610.7
r100		0.08%	4.35%	0.08%	533.4	676.5	533.4
r500		16.6%	35.24%	23.63%	2940.7	3399.2	3432.7
Avg		4.17%	9.90%	5.93%	1051.9	1319.0	1272.2

Table 3 shows the results obtained per each `%Dest` and instance size. The first column is assigned to the instance size; in the second one, the `%Dest` values to be evaluated are shown; and, in column blocks RPD and Time(s), we provide the minimum (Min), the average (Avg), and the maximum (Max) ARP/Time obtained per each group size of the calibration set. It is observed for small instances ($r = 30, 60$) that a local optimum was found and the perturbations applied to the solutions were enough to find a good result in few seconds providing no difference among the `%Dest` options in terms of solution quality; however, in terms of computing times, removing a 30% of the solution requires less time than removing 50 or 70%. On the other hand, for large instances ($r = 100, 500$), it is easier to identify the options that obtain the best results providing more information about the algorithm performance when the value of `%Dest` varies. We can observe that the best RPD values and computing times are obtained when the percentage of the elements to remove is 30%, obtaining 1.30% of ARPD in 843.7s on average (the maximum RPD is also small for this case). Therefore, the value of `%Dest` $= 30$ is definitely the percentage to be destroyed of a given solution s in the IGPSB as it outperforms the other options.

5.3.2 Calibration of β

The second parameter to calibrate corresponds to β, which is a value in the [0,1] range used to calculate the threshold of acceptance of poor-quality solutions during the IGPSB (see Algorithm 2). The smaller the β value, the tighter the acceptance threshold and fewer solutions with worse objective value will be accepted. A value near 1 allows to accept any solution that falls within the current gap computed between the best solution found so far and the last (average) solution obtained in the current iteration. For these tests, we set this parameter to $\beta = 0.0, 0.1, 0.2$ and 0.3. Since it is important that the algorithm has a mechanism for escaping from local optima, the parameter $\beta = 0.0$ will be used only for comparative terms on the degradation of the quality of the solution when considering greater values of this parameter. The aim is to determine the $\beta > 0.0$ that obtains the best trade-off between the average RPD and the average computing times. We consider all the parameters mentioned in the previous tests including `%Dest` = 30. Four repetitions of the IGPSB are carried out for each tested instance.

Table 4: RPD and computing times considering different β values in the IGPSB

Instance	β	#Best	RPD			Time(s)		
			Min	Avg	Max	Min	Avg	Max
r30	**0.0**	10/10	0.00%	0.00%	0.00%	200.50	272.09	363.64
r60		10/10	0.00%	0.00%	0.00%	204.96	276.92	369.85
r100		10/10	0.00%	0.00%	0.00%	339.67	440.97	533.36
r500		10/10	0.00%	4.32%	10.49%	1855.42	2384.62	2940.68
	Avg		**0.0%**	**1.1%**	**2.6%**	**650.14**	**843.65**	**1051.88**
r30	**0.1**	10/10	0.00%	0.00%	0.00%	76.80	81.15	85.98
r60		10/10	0.00%	0.00%	0.00%	82.07	87.28	92.72
r100		10/10	0.00%	1.11%	4.35%	271.52	310.00	385.08
r500		8/10	4.35%	11.54%	24.30%	1564.97	1659.98	1764.41
	Avg		1.1%	3.2%	7.2%	498.84	534.60	582.05
r30	**0.2**	10/10	0.00%	0.00%	0.00%	78.92	83.86	88.82
r60		10/10	0.00%	0.00%	0.00%	81.23	88.47	96.56
r100		10/10	0.00%	1.11%	4.35%	265.02	309.73	376.41
r500		8/10	4.35%	12.95%	27.60%	1594.46	1724.32	1910.12
	Avg		1.1%	3.5%	8.0%	504.91	551.59	617.98
r30	**0.3**	10/10	0.00%	0.00%	0.00%	78.72	83.21	88.69
r60		10/10	0.00%	0.00%	0.00%	81.08	88.59	97.04
r100		10/10	0.00%	1.11%	4.35%	269.22	309.13	371.56
r500		9/10	1.20%	9.66%	23.04%	1575.94	1668.64	1761.99
	Avg		**0.3%**	**2.7%**	**6.8%**	**501.24**	**537.39**	**579.82**

The description of Table 4 is similar to that of Table 3, but now the comparison is with respect to β. According to the results shown in Table 4, there is a clear option that provides better results on average, which corresponds to $\beta = 0.3$. We can observe that, although the average RPD has been slightly better when $\beta = 0.0$, this option does not allow to escape from local optimal eliminating any possibility of improving the solution. Furthermore, the number of best solutions found and the average computing times are better when $\beta = 0.3$, the latter giving the opportunity to further

explore the solution space (if the stopping criteria allow it) in order to find better solutions at the end of the optimization. Therefore, $\beta = 0.3$ is the value that provides the best trade-off among the evaluated options.

5.4 IGPSB performance

This section is devoted to assessing the performance of the proposed solution method. First, we compare the solutions obtained by the IGPSB with respect to those found by solving the MILP formulation (1)-(29) using a general-purpose solver and with respect to the initial feasible solution (improved by the LS) constructed as an input for the IGPSB. Second, since it has been observed that the MILP of the SL assignment subproblem (30)-(32) does not perform efficiently when the number of products is greater than 300, a simple but fast heuristic method is proposed to replace the corresponding MILP (especially for instances in which it does not perform well) conducting several tests to compare both alternatives in the IGPSB.

For all the tests of this section, we consider the parameter setting from the previous section, i.e., a maximum affinity value of $\gamma = 0.20$; the proportion of elements to be removed from a solution of %Dest = 30, and a $\beta = 0.3$. The stopping criteria for the IGPSB are set to $\texttt{iterIG}_{\max} = 100$ iterations and $\texttt{MaxOptTime} = 3600$ s. We use ILOG Concert Technology API (CPLEX 20.1) with 3600 s. as time limit to solve the formulation proposed for the problem; and a time limit of 600 s. to solve the SL assignment subproblem to obtain the initial feasible solution. The experiments were conducted on the *Performance set* of instances, considering four repetitions of the IGPSB for each of them. We report the best results of each solution method and the best solution values obtained by the IGPSB.

5.5 Comparison among solution methods

We now evaluate the results obtained by solving the proposed formulation (1)-(29) as well as the initial and the best solutions found after solving the first and second phases of the IGPSB, respectively. This comparison is shown in Tables 5-6 (for pseudo-real and real-world instances, respectively), where the first column is assigned for the instance name; the next block of columns (`A - MILP`), corresponds to the costs of the best solutions found, the MIP-Gap, and the computing time (seconds) required by CPLEX to solve the proposed formulation; in block (`B - IG-Ph1`), we show the solution costs and computing times required to obtain the (improved) initial feasible solution constructed in the first phase of the IGPSB; on the other hand, column block (`C - IG-Ph2`), provides the solution costs, computing times, and the final number of pickers $|W_f|$ needed to perform the routes at the end of the second phase of the IGPSB. Finally, the last column block (`Improvement`) shows the percentage of improvement between each pair of solution methods (`M1-M2`), where a negative value corresponds to an improvement obtained by method `M1` with

respect to M2.

Table 5: Performance comparison among solution methods on pseudo-real instances

Instance	A - MILP			B - IG-Ph1		C - IG-Ph2			Improvement		
	Sol	**MGap**	**Time**	**Sol**	**Time**	**Sol**	**Time**	$\|W_f\|$	**(B-A)**	**(C-A)**	**(C-B)**
Inst1r30o3p20	52.5	0.0	**9755.31**	52.5	1.0	52.5	33.6	1	0.0%	0.0%	0.0%
Inst2r30o3p20	52.5	0.0	2994.7	52.5	0.8	52.5	39.8	1	0.0%	0.0%	0.0%
Inst3r30o3p20	64.0	29.6	**14397.2**	68.7	1.1	63.9	41.1	1	7.4%	0.0%	-6.9%
Inst4r30o3p20	52.5	0.0	429.328	52.5	0.9	52.5	41.3	1	0.0%	0.0%	0.0%
Inst5r30o3p20	52.5	26.5	**14396.6**	52.5	1.1	52.5	40.2	1	0.0%	0.0%	0.0%
Inst1r30o5p20	89.6	48.4	**14400.3**	68.7	1.0	63.9	42.8	1	-23.3%	-28.6%	-6.9%
Inst2r30o5p20	52.5	0.0	3808.33	52.5	0.8	52.5	41.5	1	0.0%	0.0%	0.0%
Inst3r30o5p20	64.0	0.0	1431.77	68.7	1.1	63.9	43.7	1	7.4%	0.0%	-6.9%
Inst4r30o5p20	64.0	17.0	**14398.5**	68.7	1.2	63.9	42.9	1	7.4%	0.0%	-6.9%
Inst5r30o5p20	64.0	0.0	**8267.23**	68.7	0.9	63.9	46.6	1	7.4%	0.0%	-6.9%
Inst1r60o3p30	272.5	84.0	**14397.5**	137.4	2.8	127.9	201.7	2	-49.6%	-53.1%	-6.9%
Inst2r60o3p30	127.9	25.0	**14400**	137.4	4.3	127.9	87.9	2	7.4%	0.0%	-6.9%
Inst3r60o3p30	64.0	0.0	3086.08	68.7	3.7	63.9	48.8	1	7.4%	0.0%	-6.9%
Inst4r60o3p30	64.0	0.0	**1483.55**	68.7	2.9	63.9	50.1	1	7.4%	0.0%	-6.9%
Inst5r60o3p30	64.0	0.0	843.578	68.7	4.9	63.9	44.5	1	7.4%	0.0%	-6.9%
Inst1r60o5p30	137.4	57.4	**14400.5**	137.4	4.2	113.6	323.0	2	0.0%	-17.3%	-17.3%
Inst2r60o5p30	109.3	51.2	**14402.3**	137.4	5.5	102.7	280.1	2	25.7%	-6.1%	-25.3%
Inst3r60o5p30	64.0	0.0	2241.64	68.7	4.2	63.9	41.7	1	7.4%	0.0%	-6.9%
Inst4r60o5p30	137.4	57.0	**14402**	137.4	2.4	116.5	288.5	2	0.0%	-15.2%	-15.2%
Inst5r60o5p30	64.0	0.0	1202.8	68.7	2.5	63.9	37.2	1	7.4%	0.0%	-6.9%
Inst1r100o3p50	127.93	43.736	14399.2	137.4	22.0	127.9	114.5	2	7.4%	0.0%	-6.92%
Inst2r100o3p50	127.93	40.539	**14397.9**	137.4	21.5	127.9	92.1	2	7.4%	0.0%	-6.92%
Inst3r100o3p50	137.39	38.819	**14398.2**	137.4	18.9	127.9	86.1	2	0.0%	-6.9%	-6.92%
Inst4r100o3p50	127.93	22.246	**14400.4**	137.4	8.5	127.9	87.9	2	7.4%	0.0%	-6.92%
Inst5r100o3p50	127.93	25.796	**14400.4**	137.4	9.5	127.9	81.9	2	7.4%	0.0%	-6.92%
Inst1r100o5p50	379.49	85.13	**14400.5**	137.4	21.1	127.9	331.9	2	-63.8%	-66.3%	-6.92%
Inst2r100o5p50	127.93	49.014	**14402**	137.4	28.3	127.9	277.7	2	7.4%	0.0%	-6.92%
Inst3r100o5p50	888.64	92.778	**14403.7**	137.4	8.8	127.9	80.9	2	-84.5%	-85.6%	-6.92%
Inst4r100o5p50	—	—	**14406**	137.4	55.0	127.9	198.0	2	—	—	-6.92%
Inst5r100o5p50	179.33	66.262	**14403.7**	137.4	23.1	127.9	159.1	2	-23.4%	-28.7%	-6.92%
Inst1r1000o150p100	—	—	—	169.7	601.3	127.9	3648.9	2	—	—	-24.64%
Inst2r1000o150p100	—	—	—	137.4	601.2	127.9	3647.7	2	—	—	-6.92%
Inst3r1000o150p100	—	—	—	137.4	601.2	127.9	3718.2	2	—	—	-6.92%
Inst4r1000o150p100	—	—	—	169.7	601.5	127.9	3716.9	2	—	—	-24.64%
Inst5r1000o150p100	—	—	—	305.0	601.8	127.9	3762.9	2	—	—	-58.07%
Inst1r1000o250p100	—	—	—	169.7	601.7	127.9	3948.6	2	—	—	-24.64%
Inst2r1000o250p100	—	—	—	150.8	602.5	127.9	4013.1	2	—	—	-15.18%
Inst3r1000o250p100	—	—	—	137.4	601.7	127.9	3638.7	2	—	—	-6.92%
Inst4r1000o250p100	—	—	—	137.4	601.1	127.9	3689.2	2	—	—	-6.92%
Inst5r1000o250p100	—	—	—						—	—	
Inst1r3000o150p100	—	—	—	274.8	601.4	255.8	3668.5	4	—	—	-6.92%
Inst2r3000o150p100	—	—	—	385.1	601.4	301.4	3660.7	4	—	—	-21.72%
Inst3r3000o150p100	—	—	—	274.8	601.9	255.8	3928.9	4	—	—	-6.92%
Inst4r3000o150p100	—	—	—	457.7	602.4	319.7	3893.4	5	—	—	-30.15%
Inst5r3000o150p100	—	—	—	339.4	602.5	255.8	3922.7	4	—	—	-24.64%
Inst1r3000o250p100	—	—	—	274.8	602.8	255.8	3807.7	4	—	—	-6.92%
Inst2r3000o250p100	—	—	—	274.8	601.8	255.8	3847.8	4	—	—	-6.92%
Inst3r3000o250p100	—	—	—	274.8	463.0	255.8	3711.2	4	—	—	-6.92%
Inst4r3000o250p100	—	—	—	339.4	601.2	255.8	3655.2	4	—	—	-24.64%
Inst5r3000o250p100	—	—	—	343.5	229.1	319.7	3753.4	5	—	—	-6.92%
Inst1r5000o150p100	—	—	—	464.2	568.5	447.6	3819.5	7	—	—	-3.59%
Inst2r5000o150p100	—	—	—	480.8	298.1	447.6	3811.6	7	—	—	-6.92%
Inst3r5000o150p100	—	—	—	640.8	602.6	447.6	4032.3	7	—	—	-30.15%
Inst4r5000o150p100	—	—	—	480.8	603.0	447.6	3913.8	7	—	—	-6.92%
Inst5r5000o150p100	—	—	—	480.8	603.5	447.6	4064.6	7	—	—	-6.92%
Inst1r5000o250p100	—	—	—	754.0	603.8	527.6	4305.6	7	—	—	-30.03%
Inst2r5000o250p100	—	—	—	412.1	602.0	383.6	3639.7	6	—	—	-6.92%
Inst3r5000o250p100	—	—	—	480.8	506.8	447.6	3733.0	7	—	—	-6.92%
Inst4r5000o250p100	—	—	—	594.0	601.5	447.6	3859.2	7	—	—	-24.64%
Inst5r5000o250p100	—	—	—	549.5	601.4	511.5	3775.5	8	—	—	-6.92%

It can be observed in Table 5 that CPLEX (`A - MILP`) was able to find the optimal solution in 11 out of 30 instances with 30-100 order lines. Instances that required more than 3600s of computing time were re-optimized considering a time limit of 14400s in an attempt to find a better near-optimum solution. It can be noticed that, even for the smallest instances (30 order lines), CPLEX was not able to find an optimal solution for four of them in 4 hours, much less for the largest ones for which no feasible solution was found. On the other hand, the solution algorithm `B` (first phase of the IGPSB), obtained good initial feasible solutions efficiently for the pseudo-real instances requiring at most 604s for the largest ones (most of the computation time was consumed by the SL assignment subproblem); while the solution method `C` (second IGPSB phase) improved most of those initial solutions from `B`, reporting improvements from 6.92% up to 58.10% in approximately 1h of optimization outperforming the previous alternatives in terms of solution quality. Considering that the company is interested in knowing the smallest number of pickers required to perform the routes to complete customer orders, we can conclude that for small instances where the number of available pickers is equal to the number of customer orders ($|W| = |O| = 3, 5$), it is needed at most 2 pickers; whereas, for large instances, where $|W| = |O| = 150, 250$, only required from 4 to 8 pickers.

Table 6: Performance comparison among solution methods on real-world instances

Instance	A - MILP			B - IG-Ph1		C - IG-Ph2			Improvement		
	Sol	MGap	Time	Sol	Time	Sol	Time	$\|W_f\|$	(B-A)	(C-A)	(C-B)
Int23r101o5p98	89.7	42.4	3603.1	107.7	601.0	63.9	647.0	1	20.1%	-28.7%	-40.6%
Int85r107o1p107	129.9	63.0	3599.2	80.1	601.1	63.9	642.0	1	-38.3%	-50.8%	-20.2%
Inst53r110o8p107	103.0	48.7	3609.3	261.8	601.2	63.9	638.9	1	154.2%	-37.9%	-75.6%
Inst58r111o7p106	141.3	61.3	3607.1	201.5	601.4	107.7	863.9	1	42.6%	-23.8%	-46.5%
Inst98r111o9p106	75.4	36.5	13691.9	91.5	601.1	63.9	637.7	1	21.4%	-15.2%	-30.2%
Inst22r362o30p325	—	—	—	119.1	606.9	80.1	665.4	1	—	—	-32.8%
Inst22r457o44p376	—	—	—	815.5	708.5	702.9	1623.1	1	—	—	-13.8%
Int121r463o36p411	—	—	—	1033.0	683.1	810.5	2229.3	1	—	—	-21.5%
Inst93r467o32p414	—	—	—	125.8	610.7	125.8	668.6	1	—	—	0.0%
Inst154r477o33p420	—	—	—	782.3	630.3	568.0	2017.5	1	—	—	-27.4%
Inst134r5064o384p2923	—	—	—	—	—	—	—	—	—	—	—
Inst2r5528o399p3128	—	—	—	—	—	—	—	—	—	—	—
Inst22r7643o654p3944	—	—	—	—	—	—	—	—	—	—	—
Inst108r8197o596p4252	—	—	—	—	—	—	—	—	—	—	—

Table 6 shows a similar description but now considering the Subset B of instances (real-world). For this set, CPLEX was not able to find the optimal solution, even though the number of order lines is around 100 and customer orders does not exceed 44. An important characteristic of this set is that the number of products is similar than the number order lines, which is an important difference from the pseudo-real instances and has an important effect on the efficiency of the proposed algorithms. It can be noticed for sizes larger than 5000, no feasible solution could be found for any solution method, the reason for this is due to the fact that loading the instance to CPLEX could not be

completed and, since all the methods require to solve an MILP, all of them were affected. In response to this, we have developed a simple heuristic method to address this issue in the IGPSB, which is described and discussed in the following section.

5.6 IGPSB performance using a SLA heuristic

As we can observed in the previous section for real-world instances, where the number of products is greater than 2000, it is difficult to introduce the problem into CPLEX to be solved. Therefore, no solution for the SL assignment subproblem can be created, showing a limitation to perform the second phase of the IGPSB. As an alternative to address this issue, we have proposed a simple heuristic to create an initial assignment of products into racks and thus continue with the batching and routing decisions to generate a first initial feasible solution for the IGPSB. The proposed heuristic, shown in Algorithm 3, starts by sorting the set of products and available racks according to their affinity and their closeness to the depot, respectively. Then, in a sequential order, the first rack l is *open* and the first product p is assigned to it if there exists available space to locate (at least) a proportion of its total demand and its affinity value, with respect to the first product $\bar{p}$ assigned to l, is less than γ; otherwise, the current rack is no longer considered and another one must be open to locate the products that are not yet assigned. Once the total units of products p are place into racks, the product is removed from the non-assigned set. Finally, a feasible assignment is provided if all products were located in a given storage location.

The heuristic was embedded in the first phase of the IGPSB replacing the MILP that solves the initial SL assignment. Then, the IGPSB was performed (four repetitions per each instance) and results are briefly summarized in the Figures 9a-9b, where the solution costs and computing times provided by the IGPSB, considering the MILP and the SLA heuristic, are compared. It is observed in Figure 9a that, for the pseudo-real instances (30-5000 order lines), both procedures perform similar; in this case, the SLA heuristic (dashed red lines) obtained improvements of up to 18% in five of the large instances of this set. On the other hand, for small and large real-world instances (RWS, RWL, respectively), significant differences are found (up to 91% of improvement); in fact, as it has been pointed, the MILP for the SLA subproblem has complications in carrying out the load of RWL instances and no feasible solutions can be found to proceed; so, the proposed heuristic can solve this problem. In terms of computing times, Figure 9b provides the required time for each phase of the IGPSB considering both alternatives. For the first IGPSB phase (obtaining an improved initial solution), the SLA heuristic (dashed red lines) outperforms the MILP alternative (solid red lines) in most of the cases, especially for larger instances. For the second phase, the time required for the IGPSB is similar in both cases on pseudo-real instances; however, the most noticeable results, in which the performance of the heuristic is superior, correspond to those provided for real-world instances.

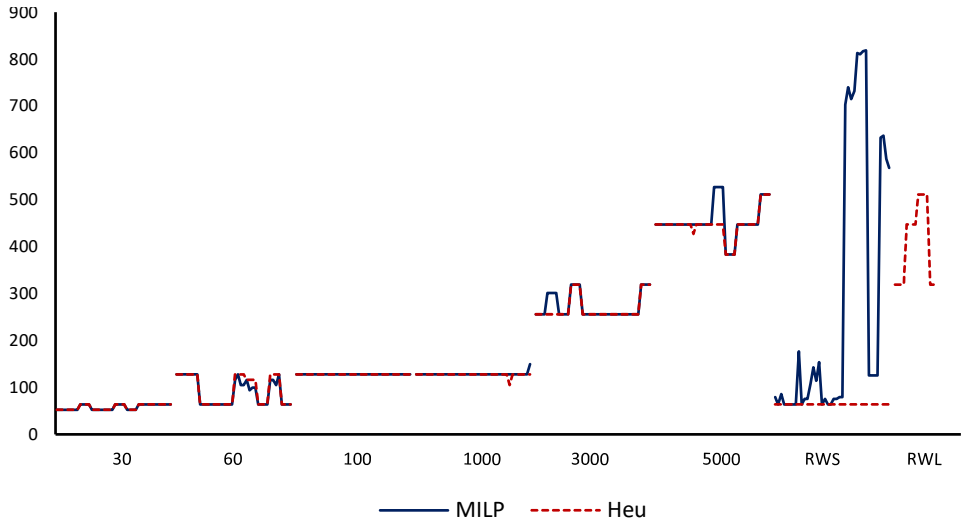
Algorithm 3 `SLAHeuristic()`.

Input: : L, P, D_p^T, Q^L ▷ Set of racks, products, total demand per product, rack capacity

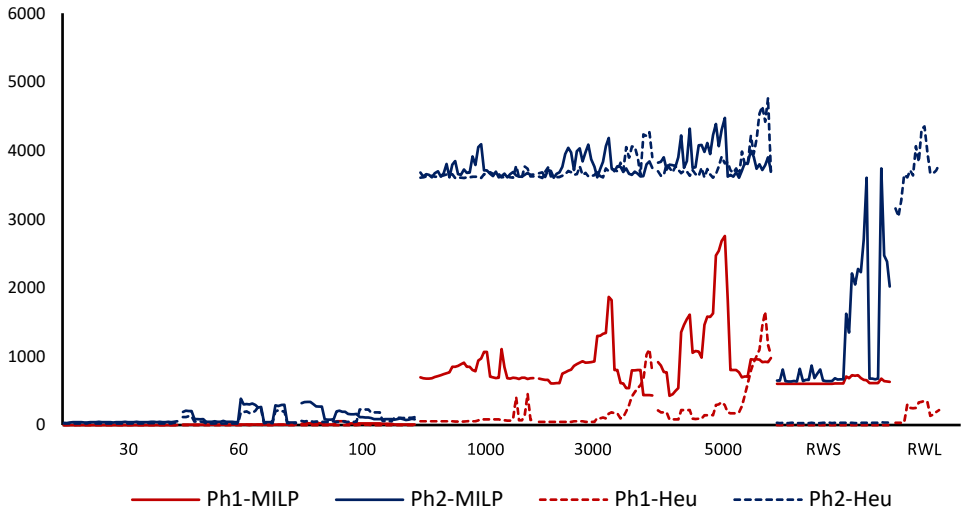
Output: $As(l, p, q)$ ▷ Final assignment (rack, product, quantity)

1: $P^s \leftarrow$ `sortByAffinity`(P) ▷ P^s is sorted P
2: $L^s \leftarrow$ `sortByClosenessToDepot`(L) ▷ L^s is sorted L
3: $Q^L_{rem} \leftarrow Q^L$ ▷ Remaining capacity
4: **while** $P^s \neq \emptyset$ and $L^s \neq \emptyset$ **do**
5: Select the first rack $l \in L^s$
6: Select the first item $p \in P^s$
7: Set $\bar{p}$ the first item p assigned to rack l
8: $D_{cov} \leftarrow \min(D_p^T, Q^L_{rem})$ ▷ Covered demand
9: **if** $D_{cov} > 0$ and $\texttt{Aff}_{\bar{p}p} \leq \gamma$ **then**
10: $As(l, p, D_{cov})$: Assign D_{cov} units of p to l
11: $Q^L_{rem} \leftarrow Q^L_{rem} - D_{cov}$
12: $D_p^T \leftarrow D_p^T - D_{cov}$
13: **else**
14: Remove l from L^s
15: Remove p from P^s if D_p^T is equal to 0
16: $Q^L_{rem} \leftarrow Q^L$
17: **end if**
18: **end while**
19: **if** $P^s \neq \emptyset$ **then**
20: $As(l, p, q) \leftarrow \emptyset$ ▷ Infeasible, no assignment found
21: **end if**
22: **return** $As(l, p, q)$

Both alternatives, solving directly the MILP with the branch and bound of the solver and the heuristic implemented for the SLA subproblem, seem to perform similar for small and medium instance sizes. We suggest to solve directly the MILP with branch and bound only for instances with less than 5000 order lines, 250 customer orders, and 100 available products as we have observed that, when the number of products increases, the MILP is harder to solve by a general-purpose solver (even if the number of order lines is small). For the remaining cases, the proposed SLA heuristic is used to obtain the initial SLA for the initial solution of the PSB.



(a) Final IGPSB solution costs



(b) Total computing times for phase 1 and 2 of the IGPSB

Figure 9: Comparison of final solution costs and computing times using the SLA MILP vs the pure heuristic in the IGPSB

6 Conclusions

This paper addresses an order picking problem, which integrates storage location, batching, and routing decisions. The aim is to find the allocation of products to racks, the grouping of customer orders, and the sequence of routes performed by pickers that minimize the order picking cost. A mixed-integer linear programming model is introduced and an Iterated Greedy Local Search algorithm (IGPSB) is proposed to solve the problem. Comprehensive computational experiments have been carried out to calibrate the main parameters of the IGPSB and to assess the performance of the proposed solution methods on pseudo-real and real-world instances focused on the basis of a Mexican retail firm. During the performance tests, we have observed that the commercial branch-and-bound method applied directly to the MILP model, used to solve the SLA subproblem when the initial feasible solution is constructed, was able to solve all the pseudo-real instances. However, for the real-world instances, the branch-and-bound method started to present problems, especially for larger sizes where it was not able to load the problem into the solver. Therefore, a simple but fast heuristic method was developed to obtain the initial location assignment of products obtaining initial solutions very quickly. With this new proposal, final results have shown that the IGPSB can obtain feasible solutions in reasonable computing times for instances with up to 8197 order lines, 654 customer orders, and up to 4252 products. On the other hand, the proposed model solved by a commercial state-of-the-art solver can only achieve optimal solutions for a few small size instances, not being able to solve those of larger size.

Future research lines identified during the development of this work are devoted to strengthening the proposed formulation by developing other valid inequalities or the introduction of cuts during the optimization process to reduce the symmetries in batching and routing decisions. Another research direction is to extend the problem scope by considering the integration of other decisions such as zoning, packaging, or last mile operations. Also, it may be interesting to study the use of other routing, batching and storage location policies to make a comparison with the strategies proposed in this work; or use other criteria in the objective function, for example, consider a penalization for due orders or introduce a decision that considers inventory levels in order to know when to replenish product in racks. In terms of robustness, it would be interesting to make a complete study of the proposed algorithms considering other layouts, heterogeneous fleet of picker carts, or integrate dynamic and stochastic elements.

Acknowledgments: The research of Diana Huerta-Muñoz has been supported by the Mexican Council for Science and Technology (CONACYT) through a postdoctoral fellowship. The research of Roger Ríos-Mercado has been partially funded by CONACYT (grant FC-2016-2/194), and UANL (grants PAICYT CE1837-21 and 241-CE-2022). The research of Jesús Fabián López-Pérez has been partially supported by UANL. This support is gratefully acknowledged.

References

Albareda-Sambola, M., Alonso-Ayuso, A., Molina, E., and De Blas, C. S. (2009). Variable neighborhood search for order batching warehouse. *Asia-Pacific Journal of Operational Research*, 26(05):655–683.

Braekers, K., Ramaekers, K., and Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313.

Celik, M., Archetti, C., and Sural, H. (2022). Inventory routing in a warehouse: The storage replenishment routing problem. *European Journal of Operational Research*, 301(3):1117–1132.

Chackelson, C., Errasti, A., Ciprés, D., and Lahoz, F. (2013). Evaluating order picking performance trade-offs by configuring main operating strategies in a retail distributor: A design of experiments approach. *International Journal of Production Research*, 51(20):6097–6109.

Chen, C.-M., Gong, Y., de Koster, R. B. M., and van Nunen, J. A. E. E. (2010). A flexible evaluative framework for order picking systems. *Production and Operations Management*, 19(1):70–82.

Chen, W., Zhang, Y., and Zhou, Y. (2022). Integrated scheduling of zone picking and vehicle routing problem with time windows in the front warehouse mode. *Computers & Industrial Engineering*, 163:107823.

Christophe, T., Olli, B., Wout, D., and Birger, R. (2010). Using a TSP heuristic for routing order pickers in warehouses. *European Journal of Operational Research*, 200(3):755–763.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(04):568–581.

de Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501.

Dukic, G. and Opetuk, T. (2012). Warehouse layouts. In Manzini, R., editor, *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems*, chapter 3, pages 55–69. Springer, London, UK.

Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.

Gademann, N. and Velde, V. D. S. (2005). Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE Transactions*, 37(1):63–75.

Gutin, G. and Punnen, A. P. (2007). *The Traveling Salesman Problem and Its Variations*. Springer, New York.

Hausman, W. H., Schwarz, L. B., and Graves, S. C. (1976). Optimal storage assignment in automatic warehousing systems. *Management Science*, 22(6):629–638.

Henn, S., Koch, S., and Wäscher, G. (2012). Order batching in order picking warehouses: A survey of solution approaches. In Manzini, R., editor, *Warehousing in the Global Supply Chain*, chapter 6, pages 105–137. Springer, London, UK.

Henn, S. and Wäscher, G. (2012). Tabu search heuristics for the order batching problem in manual order picking systems. *European Journal of Operational Research*, 222(3):484–494.

Hsieh, L.-F. and Huang, Y.-C. (2011). New batch construction heuristics to optimise the performance of order picking systems. *International Journal of Production Economics*, 131(2):618–630.

Kofler, M., Beham, A., Wagner, S., and Affenzeller, M. (2014). Affinity based slotting in warehouses with dynamic order patterns. In Klempous, R., Nikodem, J., Jacak, W., and Chaczko, Z., editors, *Advanced Methods and Applications in Computational Intelligence*, volume 6 of *Topics in Intelligent Engineering and Informatics*, chapter 7, pages 123–143. Springer, Cham, Switzerland.

Kübler, P., Glock, C. H., and Bauernhansl, T. (2020). A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses. *Computers & Industrial Engineering*, 147:106645.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

Lodi, A., Martello, S., and Vigo, D. (2002). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1):379–396.

Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., and Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 11, pages 321–353. Springer, Boston.

Petersen, C. G. and Aase, G. (2004). A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19.

Rojas Reyes, J. J., Solano-Charris, E. L., and Montoya-Torres, J. R. (2019). The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10(2):199–224.

Scholz, A. and Wäscher, G. (2017). Order batching and picker routing in manual order picking systems: the benefits of integrated routing. *Central European Journal of Operations Research*, 2017(25):491–520.

Silva, A., Coelho, L. C., Darvish, M., and Renaud, J. (2020). Integrating storage location and order picking problems in warehouse planning. *Transportation Research Part E: Logistics and Transportation Review*, 140:102003.

Tompkins, J. A., White, J. A., Bozer, Y. A., Frazelle, E. H., and Tanchoco, J. M. A. (2010). *Facilities Planning*. Wiley, New York.

van Gils, T., Caris, A., Ramaekers, K., and Braekers, K. (2019a). Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. *European Journal of Operational Research*, 277(3):814–830.

van Gils, T., Caris, A., Ramaekers, K., Braekers, K., and de Koster, R. B. (2019b). Designing efficient order picking systems: The effect of real-life features on the relationship among planning problems. *Transportation Research Part E: Logistics and Transportation Review*, 125:47–73.

van Gils, T., Ramaekers, K., Braekers, K., Depaire, B., and Caris, A. (2018a). Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions. *International Journal of Production Economics*, 197:243–261.

van Gils, T., Ramaekers, K., Caris, A., and de Koster, R. B. (2018b). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15.

Wagner, S. and Mönch, L. (2023). A variable neighborhood search approach to solve the order batching problem with heterogeneous pick devices. *European Journal of Operational Research*, 304(2):461–475.

Wang, M., Zhang, R.-Q., and Fan, K. (2020). Improving order-picking operation through efficient storage location assignment: A new approach. *Computers & Industrial Engineering*, 139:106186.

Öncan, T. (2015). MILP formulations and an iterated local search algorithm with tabu thresholding for the order batching problem. *European Journal of Operational Research*, 243(1):142–155.