INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

IFORS

Check for updates

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2025) 1–34
DOI: 10.1111/itor.70105

An iterated greedy-based metaheuristic with local search for the rank pricing problem

Herminia I. Calvete*, Carmen Galé, Aitor Hernández and José A. Iranzo

Departamento de Métodos Estadísticos, Instituto Universitario de Investigación de Matemáticas y Aplicaciones (IUMA), Universidad de Zaragoza, Pedro Cerbuna 12, Zaragoza 50009, Spain
E-mail: herminia@unizar.es[Calvete]; cgale@unizar.es[Galé]; aitor.hernandez@unizar.es[Hernández]; joseani@unizar.es[Iranzo]

Received 8 April 2025; received in revised form 9 September 2025; accepted 11 September 2025

Abstract

The rank pricing problem involves determining optimal prices for a set of products while accounting for customers' budgets and preferences. This study develops an iterated greedy-based metaheuristic to efficiently solve this problem. The core idea is to generate a sequence of solutions by iteratively applying destruction and reconstruction phases. In this process, some components of a solution are removed, yielding partial solutions from which complete solutions are reconstructed. A local search method with three neighborhood exploration strategies is then applied. Computational experiments demonstrate the effectiveness of the proposed algorithm by comparing its performance with exact and heuristic methods from the literature. It consistently finds optimal or near-optimal solutions for instances with known optima. For most cases where the optimal solution is unknown, the algorithm matches or outperforms the best-known solutions. Moreover, it achieves these results with significantly lower computational times, reinforcing its suitability for solving the rank pricing problem.

Keywords: rank pricing problem; iterated greedy; solution destruction; solution reconstruction; metaheuristic algorithms; bilevel optimization

1. Introduction

The rank pricing problem (RPP) is a pricing optimization problem, which consists of determining the optimal prices for a set of products offered by a company, while considering both the budgets and the preferences of a set of customers, each of whom is interested in purchasing a single unit of one product (Calvete et al., 2019). The problem assumes an unlimited availability of products. Each customer ranks the products in a strict order based on their characteristics and personal

*Correspondence author

Table 1
An instance of the RPP with nine customers and five products

Customers	Products					Budget
	i_1	i_2	i_3	i_4	i_5	
k_1	3	—	4	•5	—	72
k_2	2	4	1	•5	3	67
k_3	•5	—	3	4	—	66
k_4	3	2	•4	5	—	57
k_5	2	•5	4	—	3	54
k_6	4	—	•3	5	2	49
k_7	2	•5	3	—	4	48
k_8	2	4	—	5	•3	32
k_9	3	5	2	4	—	22
	Optimal prices					Revenue
	66	48	49	67	32	426

preferences; that is, ties in preferences are not allowed, meaning that two products cannot be equally preferred. On the one hand, customers aim to maximize their satisfaction by acquiring their most preferred product among the ones they can afford. On the other hand, the goal of the company is to maximize its revenue, which is computed as the total amount of money paid by the customers who make a purchase. A lower price may reduce revenue if customers were already willing to pay more, but it can also broaden the pool of potential buyers by making the product more affordable. Conversely, setting a higher price has the potential to increase revenue, though it risks discouraging purchases if the price exceeds what customers are able to pay.

Table 1 presents an instance of the RPP with nine customers and five products. The central part of the table reports the preference of each customer for every product (the larger the number, the stronger the preference), while the last column displays their budgets. If "high" prices are set, for instance, $p_1 = 66$, $p_2 = 67$, $p_3 = 57$, $p_4 = 72$, $p_5 = 54$, the purchasing decision of customers is $k_1 \rightarrow i_4$; $k_2 \rightarrow i_2$; $k_3 \rightarrow i_1$; $k_4 \rightarrow i_3$; $k_5 \rightarrow i_5$; customers k_6, k_7, k_8 and k_9 cannot purchase any product, and the company's revenue is 316. In contrast, if "low" prices are chosen, for instance $p_1 = 49$, $p_2 = 48$, $p_3 = 22$, $p_4 = 57$, $p_5 = 32$, the customers' selection would be $k_1, k_2, k_4 \rightarrow i_4$; $k_3, k_6 \rightarrow i_1$; $k_5, k_7 \rightarrow i_2$; $k_8 \rightarrow i_5$; $k_9 \rightarrow i_3$, and the company's revenue is 419. The optimal product prices appear in the last row of the table, and the products purchased by each customer in the optimal solution are indicated with a red dot in the preference matrix. In this solution, customers k_1, k_2, k_3, k_5, and k_7 purchase their most preferred product; customer k_4 purchases their second choice; customers k_6 and k_8 purchase their third choice; and customer k_9 cannot purchase any product. The total company revenue amounts to 426.

The RPP exhibits an underlying hierarchical structure, as customers react to the prices set by the company to make their purchasing decision. Therefore, bilevel optimization provides an appropriate modeling framework for this problem. The bilevel optimization model with one leader and one follower can be formulated as

$$
\begin{array}{ll}
\underset{x}{\text{“min”}} & F(x, y) \\
\text{subject to} & \\
 & G_j(x, y) \leqslant 0, j = 1, \dots, q, \\
\text{where, for every } x \text{ fixed, } y \text{ solves} & \\
\underset{y}{\min} & f(x, y) \\
\text{subject to} & \\
 & g_h(x, y) \leqslant 0, \quad h = 1, \dots, p,
\end{array}
$$

where $x \in \mathbb{R}^n$ are the upper level (UL) variables controlled by the leader and $y \in \mathbb{R}^m$ are the lower level (LL) variables controlled by the follower; $F, f : \mathbb{R}^{n+m} \longrightarrow \mathbb{R}$ are the UL and LL objective functions, respectively; and $G_j(x, y) \leqslant 0,\ j = 1, \dots, q$, and $g_h(x, y) \leqslant 0,\ h = 1, \dots, p$, indicate, in their respective cases, the constraints associated with the upper and lower levels. The quotation marks refer to the ambiguity that arises when the LL problem has multiple optimal solutions that lead to different values of the UL objective function. In such cases, the leader cannot anticipate which solution the follower will choose. Therefore, a selection rule must be established to select one of these optima to properly evaluate the UL objective. When the LL problem has a unique optimal solution for any given set of UL decision variables, the bilevel problem is said to be well-posed. In this case, the use of quotation marks is no longer necessary. A comprehensive overview of bilevel optimization, including optimality conditions, solution methods, applications, and related developments, can be found in Aussel et al. (2025), Bard (1998), Colson et al. (2007), Dempe (2002), Dempe and Zemkoho (2020), Kleinert et al. (2021), Dias Garcia et al. (2024), and Sinha et al. (2018) and the references therein. In the context of the RPP, the UL problem represents the company's pricing decisions, while the LL problem models the customers' purchasing behavior: each customer selects the most preferred product among those within their budget. This naturally leads to a bilevel formulation involving a single leader (the company) and multiple followers (the customers), each of whom independently solves their own decision problem.

Bilevel optimization problems are, in general, challenging to handle and solve to optimality (Dempe and Zemkoho, 2020). This complexity is particularly relevant when dealing with large-scale instances of the RPP (Calvete et al., 2019). However, the literature on metaheuristic algorithms for solving the RPP is limited, with existing approaches primarily focusing on evolutionary algorithms and variable neighborhood search (Calvete et al., 2024b; Jiménez-Cordero et al., 2025). This paper aims to fill this gap by proposing a novel iterated greedy-based metaheuristic with local search, designed to achieve high-quality solutions while significantly reducing computational time, particularly for large instances.

As described by Ruiz and Stützle (2007), iterated greedy (IG) is a stochastic local search method relying on a simple but effective idea. Starting from an initial solution, the method iteratively generates new solutions by applying two main phases: the partial destruction of the incumbent solution by removing some of its components, followed by its reconstruction using a greedy constructive heuristic. The acceptance criterion determines whether the new solution replaces the current one in the iterative process. IG has demonstrated its ability to provide good solutions in short computational times for combinatorial problems such as scheduling (Ruiz and Stützle, 2007; Ding et al., 2015; Fernandez-Viagas and Framinan, 2019; Feng et al., 2024) or routing problems (Karabulut and Tasgetiren, 2014; Nucamendi-Guillén et al., 2018; Wang et al., 2020; Dao et al., 2022). For a broader overview and more references on these applications, see Stützle and Ruiz (2018) and Ramalhinho and Stützle (2025).

The fundamental idea of destructing and reconstructing solutions has been widely applied across various fields, often under different names and interpretations, as will be noted in the literature review section. As highlighted by Stützle and Ruiz (2018), while the use of varying terminology and perspectives across the literature may cause some confusion, these approaches share a fundamental principle: the repeated use of constructive methods starting from intermediate or partial candidate solutions. The core idea behind the IG framework is to generate a sequence of solutions by iteratively applying destruction and reconstruction phases. In this process, components of a solution are removed, yielding partial solutions from which complete solutions are reconstructed. This iterative loop can be further enhanced with a local search phase, aimed at refining the reconstructed solutions.

The IG algorithm developed in this study consists of (1) a specialized greedy algorithm for computing the initial solution, in which each product is selected in random order and assigned the price that increases the company's revenue the most; (2) two procedures in the destruction phase for selecting the products whose prices will be removed, which differ in how the products, that is, the solution components to be destroyed, are selected; (3) a method for applying the specialized greedy algorithm in the reconstruction phase; (4) a local search method with three neighborhood exploration strategies; and (5) a non-standard acceptance criterion.

Building on the previous discussion, the main contributions of this study are the following:

- introducing the first IG algorithm to solve the RPP,
- developing a specialized greedy algorithm to construct/reconstruct a feasible solution,
- defining two specific procedures to partially destroy incumbent solutions throughout the execution of the algorithm,
- proposing several local search procedures to enhance the algorithm's performance, and
- conducting extensive computational experiments to demonstrate that the algorithm provides high-quality solutions within very short computational times.

The remainder of this paper is structured as follows. Section 2 offers a review of the literature pertinent to this study. To make this study self-contained, Section 3 presents the formulation of the RPP as a bilevel problem and two reformulations as single-level problems. Section 4 provides a comprehensive and detailed description of the IG algorithm introduced in this paper to solve the RPP, outlining its key components and main characteristics. The computational experiments conducted to assess the performance and effectiveness of the proposed algorithm are presented in Section 5. Specifically, these experiments focus on selecting the best configuration of the algorithm as well as comparing its performance with other exact and metaheuristic algorithms proposed in the literature. Finally, Section 6 summarizes the main findings of the paper and offers some concluding remarks.

2. Literature review

This section begins by examining existing methods for solving the RPP, along with recent variants proposed in the literature. Some references on other problems involving preferences or prices modeled using bilevel optimization have also been included. It concludes with an overview of general references on IG algorithms as well as a number of recent contributions that illustrate its continued application across different contexts.

Calvete et al. (2019) introduce the bilevel formulation of the RPP and propose two single-level formulations of the problem. The first one comes from reformulating the bilevel problem by leveraging the properties of the LL problems. The second one directly formulates the problem as a single-level model, in which a specific set of constraints captures the structure of customers' preferences. As both formulations are nonlinear (Calvete et al., 2019, p. 13), they propose two linearizations and a branch-and-cut algorithm to solve them. They conduct extensive computational experiments, including instances with 30, 50, 100, and 150 customers.

As mentioned in the previous section, to the best of the authors' knowledge, there are only two studies developing metaheuristic methods for solving the RPP. Calvete et al. (2024b) propose an evolutionary algorithm whose chromosomes are vectors of length equal to the number of products, and the vector components represent the prices set by the company for each product. Every chromosome can be directly linked to a feasible solution of the RPP by solving the LL problems associated with customers. The evolutionary algorithm applies the uniform crossover. The mutation operator switches some components of the chromosome to another possible price. The algorithm also includes a local search procedure, which is applied after the crossover and mutation operators. This procedure consists of examining the entire set of products in a random order, with the aim of identifying products (if any) such that changing their prices individually could improve the revenue. The extensive computational experiments conducted show that the algorithm delivers excellent performance with CPU times significantly lower than those required to solve the instances using an off-the-shelf optimization algorithm.

Jiménez-Cordero et al. (2025) propose two heuristic methods to solve the RPP. The first one consists of a variable neighborhood search algorithm working with vectors of prices. The second one takes the core principles of the evolutionary algorithm developed by Calvete et al. (2024b), excluding its local search procedure. In addition, they propose four local search procedures based on how some prices can be modified to improve revenue, depending on the relationship between product prices and customer budgets, which are applied in combination with both heuristic methods. Their computational experiments involve three instances showing that the use of the four local search procedures provides the best outcomes whatever the heuristic method used.

Subsequent research on the RPP has explored new problem variants, such as the rank pricing problem with ties (RPPT) and the capacitated rank pricing problem (CRPP). Domínguez et al. (2021) introduce the RPPT by allowing customers to express indifference among multiple products and thus, assigning them the same preference value. They propose different single-level formulations of the problem and develop exact algorithms to solve them. Building on this research, Calvete et al. (2024c) take advantage of the bilevel structure of this type of problem to propose a bilevel formulation for the RPPT, demonstrating its competitiveness in solving instances from previous studies. On the other hand, Domínguez et al. (2022) extend the RPP by incorporating capacity constraints on the number of copies available for sale for each product. Their study explores different approaches depending on whether envy among customers is permitted or not.

Bilevel optimization has also been applied to model other problems involving preferences, prices, or both. In the context of facility location problems involving customer preferences, Hansen et al. (2004) consider the simple plant location problem with order introduced by Hanjoul and Peeters (1987), in which customers choose their most preferred facility based on individual ranked preference lists without ties. The problem is modeled as a pure binary linear bilevel optimization problem in which the leader decides which facilities to open, while the follower, representing all customers as a single decision-maker, selects the preferred assignments. The bilevel model is then reformulated as a single-level problem by replacing the LL optimization with a set of constraints that ensure its optimality. Vasilyev et al. (2009) and Vasilyev and Klimentova (2010) contribute to this problem by introducing valid inequalities and developing a branch-and-cut method. Camacho-Vallejo et al. (2014) propose an evolutionary algorithm designed to leverage the bilevel structure of the problem as an alternative to exact methods. Calvete et al. (2025) extend the problem to consider ties in the preference lists and model it as a bilevel optimization problem, which requires the pessimistic approach to handle the ties. Lin et al. (2024) add preferences to the *p*-median problem and develop two exact branch-and-cut solution methods. Calvete et al. (2020) introduce capacity constraints on the facilities, which may prevent all customers who wish to use a facility from actually being served by it. The paper analyzes the implications of two proposed problem approaches and adopts the bilevel optimization formulation. Additionally, the authors develop an effective metaheuristic algorithm based on a general framework inspired by evolutionary algorithms that takes advantage of the bilevel structure of the model.

Regarding pricing problems, Labbé and Violin (2016) and the references therein provide a review of studies up to that year involving price setting in hierarchical frameworks, where a leader sets taxes or prices on certain activities, and followers choose among taxed and untaxed options to minimize their operational costs. The authors focus on problems defined over network structures, where the leader owns a subset of arcs and the followers are users who travel between nodes in the network seeking minimum-cost paths. A bilevel approach has been taken by Kochetov et al. (2015) to address pricing within location decisions. They propose two hybrid local search-based algorithms: one using variable neighborhood descent and the other combining genetic search principles. Their effectiveness is evaluated against CPLEX, with results showing strong competitiveness. Additionally, Panin and Plyasunov (2012) analyze the computational complexity of the bilevel model, while Panin and Plyasunov (2023) offer a broader review of related facility location and pricing problems, summarizing the main contributions and results from their research over the years. López-Ramos et al. (2019) address transportation on congested roads through a bilevel model that integrates network pricing with hazmat routing and network design. The UL represents the operator, who sets tolls to maximize profit while accounting for construction and risk exposure, whereas the LL models vehicle routing decisions under congestion and toll charges. A reformulation combined with a tailored local search is proposed. Lin and Tian (2023) study a facility location problem where customers are responsible for travel costs, where prices are restricted to discrete values to reflect practical considerations. They model the problem as a bilevel program and propose two exact solution methods: a reformulation into a single-level mixed integer model using closed assignment constraints, and a branch-and-cut algorithm with tailored bilevel feasibility cuts. Computational results show the latter to be more efficient. Jacquet et al. (2024) address a profit-maximization problem for an electricity retailer who seeks to design a menu of contracts and propose a quadratically regularized model of customer response. Gao et al. (2025) address the optimization of the construction and operation of agricultural product origin warehouses. In their bilevel formulation, the UL supply chain enterprise decides on the location, capacity, and pricing of the warehouses, while the LL planters determine the allocation of their agricultural products among the warehouses to maximize profits. To solve the problem, the authors develop a hybrid adaptive large neighborhood search algorithm integrated with a heuristic rule-based mechanism. Benati et al. (2025) address a pricing problem in the financial industry, modeled as a bilevel optimization problem between a broker, who sets transaction fees, and an investor, who decides asset purchases in response. The broker's profit-maximization problem is formulated using an ordered median function, while the LL identifies the investor's optimal choice given the broker's pricing policy.

Finally, combining prices and preferences, Calvete et al. (2024a) study a facility location problem in which customers are responsible for travel costs, and preferences may include ties. The problem is formulated as a bilevel optimization model, where the presence of ties can lead to multiple optimal solutions in the LL problem. The assumption of rational customer behavior makes classical optimistic or pessimistic tie-breaking rules unsuitable. Therefore, each customer's decision is modeled as a lexicographic biobjective problem: first maximizing preference, then minimizing access cost to the selected facility. The resulting bilevel model is reformulated as a single-level mixed integer program using duality theory and bounded big-M constants. The computational study evaluates the reformulation's effectiveness and analyzes the impact of preference list length and facility opening costs.

The literature on IG algorithms is extensive, as it has been shown to be a highly efficient approach for addressing combinatorial problems, both on its own and in combination with other techniques. To avoid duplicating references already covered in other papers, three studies that offer numerous additional sources on the topic are highlighted. Stützle and Ruiz (2018) provide an overview of the method's principles, its applications, and its relationship with other approaches, offering an extensive list of references. In particular, their discussion of the relationships between IG and other algorithms, such as large neighborhood search (LNS) (Pisinger and Ropke, 2019), is of interest as both share a destruction/reconstruction structure. While both IG and LNS are designed to escape local optima and explore the solution space, they differ in key aspects of strategy and scope. IG typically removes a small, fixed portion of the current solution and reconstructs it using a greedy heuristic, following a relatively simple and focused trajectory through the search space. Its strength lies in its simplicity and efficiency, especially when paired with an effective construction heuristic. In contrast, LNS is designed to explore much larger and more diverse neighborhoods by employing more complex and often adaptive destroy and repair operators. These operators may be selected dynamically during the search process, allowing LNS to flexibly adapt to the problem structure and escape deep local optima more effectively. Summarizing, although both frameworks follow a similar conceptual structure, IG emphasizes efficiency and simplicity, whereas LNS prioritizes exploration power and flexibility. Missaoui et al. (2023) present a comprehensive review of the literature on variants of IG algorithms for combinatorial problems. Finally, Ramalhinho and Stützle (2025) present an up-to-date survey of IG and iterated local search algorithms, highlighting successful recent applications of both. They also emphasize that these frameworks appear in the literature under different names, reflecting their conceptual similarities. In addition, a number of recent studies (Demir, 2024; Mendoza-Gómez and Ríos-Mercado, 2024; Huerta-Muñoz et al., 2025;

Liu et al., 2025; Mousavia et al., 2025; Wang et al., 2025) highlight the growing interest and ongoing activity in this research area. Based on previous studies and searches in scientific databases, it can be concluded that the majority of papers on IG algorithms focus on scheduling problems because the nature of the solutions lends itself well to the process of destruction and subsequent greedy reconstruction. As far as the authors are aware, IG has never been applied to solve the RPP.

3. The bilevel formulation of the RPP

To formulate the RPP, let $K = \{1, \dots, |K|\}$ and $I = \{1, \dots, |I|\}$ be the sets of customers and products, respectively. For each customer $k \in K$, let b^k be their budget. Let $S^k \subseteq I$, with $S^k \neq \emptyset$, be the subset of products that the customer k is interested in, and let $\left\{s_i^k \in \mathbb{N} : i \in S^k\right\}$ represent the preference values assigned by the customer to the products of interest. Notice that the greater the preference value, the more preferred the product is. It is assumed that each product appears on at least one customer's preference list. If this is not the case, the product may be excluded from the study. In addition, $\{p_i \geqslant 0 : i \in I\}$ are the UL decision variables representing the product prices, and $\left\{x_i^k \in \{0, 1\} : k \in K, i \in S^k\right\}$ are the LL decision variables representing the customers' purchasing decisions ($x_i^k = 1$ if customer $k \in K$ buys product $i \in S^k$, and $x_i^k = 0$ otherwise).

Calvete et al. (2019) propose the following bilinear–linear bilevel mixed integer optimization formulation for the RPP with a single leader (the company) and multiple independent followers (each of the customers):

$$\max_{p} \quad \sum_{k \in K} \sum_{i \in S^k} p_i x_i^k \tag{1a}$$

subject to

$$p_i \geqslant 0 \quad i \in I, \tag{1b}$$

where, for each customer $k \in K$, the variables $\{x_i^k\}_{i \in S^k}$ solve:

$$\max_{x} \quad \sum_{i \in S^k} s_i^k x_i^k \tag{1c}$$

subject to (1c)

$$\sum_{i \in S^k} x_i^k \leqslant 1, \tag{1d}$$

$$\sum_{i \in S^k} p_i x_i^k \leqslant b^k, \tag{1e}$$

$$x_i^k \in \{0, 1\} \quad i \in S^k. \tag{1f}$$

The company aims to maximize its revenue (1a). For each customer $k \in K$, the corresponding LL problem maximizes the preference value obtained by the purchase made (1c), while ensuring that the customer buys at most one product (1d) and pays no more than their available budget (1e).

Note that the followers are independent, as each customer's LL problem involves only their own variables and those of the leader (Calvete and Galé, 2007). Moreover, as the customers' preferences are strict, meaning there are no ties, the LL problem corresponding to each customer has a unique optimal solution given the value of the UL decision variables, namely, the most preferred product they can afford. Thus, the bilevel problem is well-posed (Bard, 1998; Dempe, 2002) and, for this reason, quotation marks are not used in the formulation (1).

For the sake of clarity, next we briefly outline and streamline the approach proposed in Calvete et al. (2019) to reformulate problem (1a)–(1f) as a single-level nonlinear mixed integer optimization problem, which can be addressed using off-the-shelf solvers. When selecting prices, only distinct budgets need to be taken into account (Rusmevichientong et al., 2006). Let L represent the set of indices for the distinct budgets ordered from lowest to highest. Let σ be the function that maps customer k to the index l corresponding to the position of their budget within the ordered set of distinct budgets. Thus, in this section, $b^{\sigma(k)}$ refers to the budget of customer k.

For $i \in I$, $l \in L$, let v_i^l be an auxiliary binary variable which takes the value 1 if product i is priced at b^l; otherwise, v_i^l equals 0. Then, $p_i = \sum_{l=1}^{|L|} b^l v_i^l$, where $\sum_{l=1}^{|L|} v_i^l \leqslant 1$, since each product has at most one price, $i \in I$. Moreover, the constraint (1e) in each LL problem can be rewritten as $x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l$, $i \in S^k$.

Assume for the time being that the UL variables $\{v_i^l\}_{i\in I,\, l\in L}$ are known. Then, the customer k can afford the products in the set $I(k) = \{i \in S^k : \sum_{l=1}^{\sigma(k)} v_i^l = 1\}$. Hence, $x_i^k = 0$ for $i \in S^k \setminus I(k)$, and the optimal solution of the LL problem corresponding to the kth customer is the most preferred product $i \in I(k)$, that is, the product $i \in I(k)$ such that $\sum_{j\in I(k)} s_j^k x_j^k \geqslant s_i^k$. The above statements are derived under the assumption that the values of the UL variables are known. Consequently, only the LL variables corresponding to products in $I(k)$ are required, as the remaining variables are zero. To incorporate this structure into the UL problem, these constraints must be expressed in terms of the set S^k instead of $I(k)$, while ensuring that they are only active over the elements in $I(k)$. Constraints (2) ensure this fact:

$$\sum_{j\in S^k} s_j^k x_j^k \geqslant s_i^k \sum_{l=1}^{\sigma(k)} v_i^l \quad k \in K \quad i \in S^k. \tag{2}$$

Note that, if $i \in I(k)$ then $\sum_{l=1}^{\sigma(k)} v_i^l = 1$ and so constraints (2) are enforced. By contrast, if $i \in S^k \setminus I(k)$ then $\sum_{l=1}^{\sigma(k)} v_i^l = 0$ and the constraints are trivially satisfied.

Bringing together all the elements discussed above, problem (1a)–(1f) can be reformulated as the following single-level nonlinear problem:

$$\max_{v,x} \quad \sum_{k\in K} \sum_{i\in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_i^l \right) x_i^k \tag{3a}$$

subject to

$$\sum_{l=1}^{|L|} v_i^l \leqslant 1 \qquad i \in I, \tag{3b}$$

$$\sum_{i\in S^k} x_i^k \leqslant 1 \qquad k\in K, \tag{3c}$$

$$x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l \qquad k\in K \quad i\in S^k, \tag{3d}$$

$$\sum_{j\in S^k} s_j^k x_j^k \geqslant s_i^k \sum_{l=1}^{\sigma(k)} v_i^l \qquad k\in K \quad i\in S^k, \tag{3e}$$

$$v_i^l \in \{0,1\} \qquad i\in I \quad l\in L, \tag{3f}$$

$$x_i^k \in \{0,1\} \qquad k\in K \quad i\in S^k. \tag{3g}$$

Moreover, by introducing the nonnegative variables $\{z_i^k\}_{i\in I,k\in K}$, defined as the profit obtained by the company due to customer k purchasing product i, that is $z_i^k = \left(\sum_{l=1}^{\sigma(k)} b^l v_i^l\right) x_i^k$, problem (3a)–(3g) can be written as the following linear mixed integer optimization problem, explicitly formulated in this paper for the first time:

$$\max_{v,x,z} \quad \sum_{k\in K}\sum_{i\in S^k} z_i^k \tag{4a}$$

subject to

$$\sum_{l=1}^{|L|} v_i^l \leqslant 1 \qquad i\in I, \tag{4b}$$

$$\sum_{i\in S^k} x_i^k \leqslant 1 \qquad k\in K, \tag{4c}$$

$$x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l \qquad k\in K \quad i\in S^k, \tag{4d}$$

$$\sum_{j\in S^k} s_j^k x_j^k \geqslant s_i^k \sum_{l=1}^{\sigma(k)} v_i^l \qquad k\in K \quad i\in S^k, \tag{4e}$$

$$z_i^k \leqslant \left(\sum_{l=1}^{\sigma(k)} b^l v_i^l\right) \qquad k\in K \quad i\in S^k, \tag{4f}$$

$$z_i^k \leqslant b^{\sigma(k)} x_i^k \qquad k\in K \quad i\in S^k, \tag{4g}$$

$$v_i^l \in \{0,1\} \qquad i\in I \quad l\in L, \tag{4h}$$

$$x_i^k \in \{0,1\} \qquad k\in K \quad i\in S^k, \tag{4i}$$

$$z_i^k \geqslant 0 \qquad\qquad k \in K \quad i \in S^k. \tag{4j}$$

This linear mixed integer formulation will be used in Section 5 when the RPP is solved exactly using an optimization solver.

Remark 1. It is worth mentioning at this point a useful property of the RPP that simplifies the computation of a bilevel feasible solution once the UL variables are fixed. Specifically, it allows the LL problem of each customer to be solved without the need for an optimization solver. This property, already exploited in Calvete et al. (2024b), will also be leveraged in the algorithm proposed in this paper.

Indeed, once the UL variables (i.e., product prices) have been fixed, obtaining a bilevel feasible solution reduces to solving the LL problem for each customer. This simply requires each customer to go through their preference list and select the most preferred product they can afford, given their budget. Since preference lists are maintained as ordered vector sets, this process can be handled efficiently without the need to invoke an optimization solver for the LL problems.

Let us return to the example displayed in Table 1 to illustrate this remark. Suppose the prices are set as follows: $p_1 = 22$, $p_2 = 54$, $p_3 = 48$, $p_4 = 72$, and $p_5 = 32$. The products purchased by the customers can be identified by examining the preference matrix row by row. For customer k_1, all products are affordable and thus the most preferred, i_4, is selected. For customer k_2, product i_4 is unaffordable, so the highest ranked affordable option is i_2. For customer k_3, the budget allows the acquisition of i_1, which coincides with their top preference. Proceeding in the same manner for the remaining customers, the purchases are $k_1 \to i_4$; $k_2, k_5 \to i_2$; $k_3, k_6, k_9 \to i_1$; $k_4 \to i_3$; $k_7, k_8 \to i_5$.

4. IG-RPP: An IG-based metaheuristic with local search for the RPP

As indicated in Section 1, the goal of the RPP is to set the prices of a set of products in order to maximize the company's revenue, considering that customers react to these prices by purchasing their most preferred product among those they can afford. According to Remark 1, once the product prices have been set, the LL problems can be efficiently solved without the need for an optimization solver. Thus, computing the value of the LL variables x_i^k amounts to directly checking each customer's preference list to determine whether they can purchase any product, and, if so, which one they will choose. Then, the UL objective function value, that is, the company's revenue, can be directly computed by summing the prices paid by all customers.

As a result, the proposed algorithm focuses solely on exploring the feasible space of the UL variables. From now on, a solution S refers to a vector of size $|I|$, where the ith component represents the price set by the company for product i, that is, the value of p_i. It is important to note that, each time product prices change in an iteration, the product selected by each customer may also change. Thus, both the customers' choices (i.e., the values of the LL variables x_i^k) and the UL objective function value must be updated accordingly. This update is always performed, although, to avoid repetition, it is not explicitly stated at each step in the algorithm description.

Remember that, as pointed out in Section 3, prices can be selected from the set of different budgets, and customers only purchase products that appear on their preference list. Thus, any product can only be assigned a price that matches the budget of any customer who has this product on their

Algorithm 1. Outline of an IG algorithm with Local Search

```
1: $S^* \leftarrow$ GenerateInitialSolution
2: repeat
3:     $S'_p \leftarrow$ Destruction($S^*$)
4:     $S' \leftarrow$ Reconstruction($S'_p$)
5:     $S' \leftarrow$ LocalSearch($S'$)
6:     $S^* \leftarrow$ AcceptanceCriterion($S^*, S'$)
7: until Stopping criterion met
```

preference list. Building upon these considerations, the set of possible prices for product $i \in I$ is defined as

$$\mathcal{P}_i = \left\{ b^k : k \in K \text{ and } i \in S^k \right\}. \tag{5}$$

The key features of an IG algorithm with local search are (1) the method used to generate the initial solution, usually a greedy algorithm; (2) the iteration process, specifically how the incumbent solution is partially destroyed and then rebuilt to generate a new solution; (3) the type of local search applied to explore neighborhoods of the new solution; and (4) the acceptance criterion for new solutions. The pseudocode in Algorithm 1 describes these general steps. Obviously, the UL objective function of the best solution ever obtained, along with the values of the UL and LL variables, is maintained throughout the iterations of the algorithm. In the next subsections, a detailed explanation of each step is provided.

4.1. Initialization

A greedy algorithm is used to generate the initial solution S^*. Partial solutions, which are solutions with some unpriced products, are successively constructed until all products have been assigned a price. To do this, the products are selected in random order and assigned the best possible price, that is, the one that gives the greatest increase in the value of the UL objective function for the current partial solution. Hence, after selecting a product i to be priced, every possible price in $\mathcal{P}_i$ is evaluated. For each candidate price, it is necessary to determine which products in the current partial solution would be purchased by the customers, and to compute the resulting company revenue. Then, as indicated above, the price for product i that yields the highest increase in the current revenue is selected.

Algorithm 2 displays the pseudocode of the process. The algorithm takes a list I^* of unpriced products as input, which includes all products when constructing the initial solution. It iterates over each product $i \in I^*$, determining the price that yields the highest profit (lines 2–16) by considering all possible prices for product i in decreasing order. The variable n keeps track of the number of customers who would purchase product i at a given price, while e stores the total amount of money these customers are paying in the current solution. Finally, the algorithm updates the solution by

Algorithm 2. Construction/Reconstruction Procedure

```
 1: procedure GREEDY( list $I^*$ of unpriced products)
 2:     for $i \in I^*$ do
 3:         $G \leftarrow -1$                    ▷ highest gain when evaluating different prices
 4:         $n \leftarrow 0$                     ▷ number of interested customers in product $i$
 5:         $e \leftarrow 0$                     ▷ accumulated expense
 6:         for all $k \in K$ such that $i \in S^k$, in decreasing order of $b^k$ do
 7:             $s \leftarrow 0, p \leftarrow 0$
 8:             if customer $k$ currently buys product $j$ then
 9:                 $s \leftarrow s_j^k$          ▷ current preference of the product $j$ purchased by customer $k$
10:                 $p \leftarrow p_j$            ▷ current price paid by customer $k$
11:             if $s < s_i^k$ then
12:                 $n \leftarrow n + 1$          ▷ update the number of interested customers
13:                 $e \leftarrow e + p$          ▷ update the accumulated expense
14:                 $g \leftarrow n \cdot b^k - e$    ▷ compute the gain when setting price $b^k$
15:                 if $g > G$ then
16:                     $G \leftarrow g$          ▷ update the highest gain when improved
17:         if $n > 0$ then
18:             Set the price of product $i$ to the value that results in a gain of $G$ units in the UL objective
                function and update the solution accordingly.
19:         else
20:             Set the price of product $i$ to $\max\{\mathcal{P}_i\}$.
```

setting the price that maximizes the profit (lines 17 and 18). If no customer is interested in the product at the price determined by the current solution, its price is set to the highest possible price, that is, to $\max\{\mathcal{P}_i\}$ (lines 19 and 20). The construction procedure has worst-case time complexity $\mathcal{O}(|I| \cdot |K|)$.

Given the central role of the construction phase, which serves as the foundation for the remaining procedures, we now present its detailed application to the example displayed in Table 1. The set of possible prices for each product is

$$\mathcal{P}_1 = \{72, 67, 66, 57, 54, 49, 48, 32, 22\},$$
$$\mathcal{P}_2 = \{67, 57, 54, 48, 32, 22\},$$
$$\mathcal{P}_3 = \{72, 67, 66, 57, 54, 49, 48, 22\},$$
$$\mathcal{P}_4 = \{72, 67, 66, 57, 49, 32, 22\},$$
$$\mathcal{P}_5 = \{67, 54, 49, 48, 32\}.$$

Let us assume that the product i_3 is randomly selected as the first one to be examined. If $p_3 = 72$, only customer k_1 can purchase this product: the gain is 72. If $p_3 = 67$, customers k_1 and k_2 can buy it, the gain is 134. We continue in this way, and the gains are 198, 228, 270, 294, 336, and 176, respectively, if p_3 is set to 66, 57, 54, 49, 48, and 22. Therefore, the highest gain is 336, p_3 is set to 48, and customers k_1 to k_7 buy this product. The current company's revenue is 336.

Now, let us randomly select product i_5 for examination. Looking at $\mathcal{P}_5$, setting $p_5 = 67$, customer k_2 prefers this product, so now selects it; there are no more changes, and the gain is $67 - 48 = 19$. Setting $p_5 = 54$ or $p_5 = 49$, as before, only customer k_2 changes their selection, and the gain is $54 - 48 = 6$ or $49 - 48 = 1$. Setting $p_5 = 48$, customers k_2 and k_7 change their selection and the gain is $2(48 - 48) = 0$. Finally, setting $p_5 = 32$, customers k_2 and k_7 change their selection and customer k_8 can afford this product, so the gain is $2(32 - 48) + 32 = 0$. Therefore, the highest gain is 19, and so $p_5 = 67$. In the current partial solution, k_1, k_3 to k_7 purchase i_3, and k_2 purchases i_5. The current company's revenue is 355.

Let i_4 be the following product to be examined. Setting $p_4 = 72$, then customer k_1 changes their selection because the customer prefers i_4 over i_3. Anything else changes, the gain is $72 - 48 = 24$. Setting $p_4 = 67$, k_1 and k_2 now select i_4, the gain is $2 \times 67 - 48 - 67 = 19$. Setting $p_4 = 66$, k_1, k_2, and k_3 prefer i_4, the remaining partial solution remains without changes, and the gain is $3 \times 66 - 48 - 67 - 48 = 35$. Setting $p_4 = 57$, customers k_1 to k_4 now select i_4, and the gain is $4 \times 57 - 48 - 67 - 48 - 48 = 17$. Setting $p_4 = 49$, customers k_1 to k_4 and k_6 now select i_4, and the gain is $5 \times 49 - 48 - 67 - 48 - 48 - 48 = -14$. Setting $p_4 = 32$, customers k_1 to k_4 and k_6, as well as k_8 select i_4, and the gain is $6 \times 32 - 48 - 67 - 48 - 48 - 48 = -67$. The highest gain is 35, and so $p_4 = 66$. In the current partial solution, k_1, k_2, and k_3 buy i_4, and customers k_4 to k_7 purchase i_3. The current company's revenue is 390.

Let i_1 be the following product to be examined. Setting either $p_1 = 72$ or $p_1 = 67$ does not change the current purchasing decisions. Setting $p_1 = 66$, customer k_3 changes their selection since the customer prefers product i_1 to product i_4, and the gain is $66 - 66 = 0$. Setting p_1 at 57 and 54 provides negative gains since only customer k_3 would change their selection and would pay less than they are currently paying. Setting either $p_1 = 49$ or $p_1 = 48$, customers k_3 and k_6 change their selection, and the gain is either $2 \times 49 - 66 - 48 = -16$ or $2 \times 48 - 66 - 48 = -18$. Setting $p_1 = 32$ would also allow customer k_8 to buy this product, yielding a gain of $3 \times 32 - 66 - 48 = -18$. Finally, setting $p_1 = 22$ would also allow customers k_8 and k_9 to buy this product, yielding a gain of $4 \times 22 - 66 - 48 = -26$. Hence, according to line 20 in Algorithm 2, $p_1 = 72$. In the partial current solution, the customers' purchasing decisions have not changed. The current company's revenue is 390.

The last product to be examined is i_2. Setting p_2 at 67 or 57 does not change the current purchasing decisions. Setting $p_2 = 54$, customer k_5 changes their selection to product i_2, and the gain is $54 - 48 = 6$. Setting $p_2 = 48$, customers k_5 and k_7 select i_2, and the gain is $2 \times 48 - 48 - 48 = 0$. Setting $p_2 = 32$ would also allow customer k_8 to buy this product, resulting in a gain of $3 \times 32 - 48 - 48 = 0$. Setting $p_2 = 22$ would also allow customers k_8 and k_9 to buy this product, resulting in a gain of $4 \times 22 - 48 - 48 = -8$. Hence, the highest gain is 6, and so $p_2 = 54$. The current company's revenue is 396.

The constructed solution is $p_1 = 72$, $p_2 = 54$, $p_3 = 48$, $p_4 = 66$, and $p_5 = 67$. Regarding the customers, k_5 purchases i_2; k_4, k_6, and k_7 purchase i_3; k_1, k_2, and k_3 purchase i_4; and k_9 does not purchase any product.

Algorithm 3. Destruction Procedure

```
1: procedure DESTROY(list $I^*$ of products)
2:     for all product $i \in I^*$ do
3:         Remove price of product $i$
4:         for all customer $k \in K$ such that $x_i^k = 1$ do
5:             $x_i^k \leftarrow 0$
6:             for all product $j \in S^k$ such that $j \notin I^*$, in decreasing order of $s_j^k$ do
7:                 if $b^k \geqslant p_j$ then
8:                     $x_j^k \leftarrow 1$
9:                     break
```

4.2. Destruction

Let S^* be the incumbent solution. In this step, the price of a percentage %D of the products is removed, thus partially destroying the solution. In fact, the number of products whose price is removed is set to $\left\lceil \frac{|I|\,\%\mathrm{D}}{100} \right\rceil$. To select the products whose price is destroyed, two methods are proposed:

- *Uniform selection (US)*: Products are randomly selected with the same probability. Therefore, each time this process is applied, the products are shuffled, and then the first $\left\lceil \frac{|I|\,\%\mathrm{D}}{100} \right\rceil$ products in the shuffled list are selected.
- *Based on price variability selection (PVS)*: This method selects products for removal based on the variability of their prices in the best solutions so far encountered. The underlying idea is that products with lower variability have presumably already reached their best price, and, therefore, their price may not need to be changed. To measure such variability, a number of available solutions are needed. Therefore, this method is applied after 1000 iterations of the algorithm have been completed. Prior to this, US is always applied. After 1000 iterations have been completed, the best 100 distinct solutions are taken. Then, for each product, the variance of the collection of assigned prices in these solutions is computed. After sorting the products from lowest to highest variance, the products are divided into five blocks. The first four blocks contain $\left\lfloor \frac{|I|}{5} \right\rfloor$ products each, and the fifth block contains the remaining products. Then, a weight of 1 is assigned to each product in the first block, a weight of 2 to each in the second block, and so on, until a weight of 5 is assigned to each product in the fifth block. Products are selected at random, with a probability of being chosen directly proportional to their weight. The update of these weights is only performed every 1000 iterations.

The process of removing product prices is carried out using the destruction procedure detailed in Algorithm 3. It ensures that once a product's price is removed, affected customers reallocate their purchases to the best available alternative according to their preferences and budget constraints.

The destruction procedure has worst-case time complexity $\mathcal{O}(|I^*| \cdot |I| \cdot |K|)$, where I^* denotes the list of products whose prices are to be removed.

4.3. Reconstruction

In this step, Algorithm 2 described in Section 4.1 is applied, considering as the list I^* of unpriced products those whose prices were removed during the application of the Destruction procedure. In particular, during the reconstruction step, the products are processed in the order in which they were chosen when destroying the incumbent solution S^*. The reconstruction procedure has worst-case time complexity $\mathcal{O}(|I^*| \cdot |K|)$.

4.4. Local search

After the reconstruction step of a solution is completed, several local search procedures are considered. Three different local search techniques are designed to effectively explore the neighborhood of the most promising solutions. They are described in the following paragraphs along with the moment they are applied.

4.4.1. Local search 1 (LS1)

LS1 examines all the products one by one in a random order and, for each one, it determines the best possible price for it among the possible prices, looking at the change in the UL objective function. If it is strictly improved, the current solution is updated to the new solution. Essentially, for each product separately, what is done is to destroy the solution for that product, that is, remove its price, and to reconstruct it by selecting its best possible price using Algorithm 2 with this product as the only unpriced one.

After examining all the products, it is guaranteed that the current solution is at least as good as the one with which the local search began, in terms of the value of the UL objective function provided. If this value is the same, the LS1 procedure stops; otherwise, it iterates, considering the current solution, until no further improvement is made. Each iteration of the LS1 procedure has worst-case time complexity $\mathcal{O}(|I|^2 \cdot |K|)$.

4.4.2. Local search 2 (LS2)

LS2 examines all the pairs of products. For a given pair of products, their prices are swapped. Note that this could result in a product having a price which is not possible for that product. Whether this happens or not, each product in the pair is then examined individually in order to assign it the best possible price, looking at how the UL objective function changes. The procedure works as follows. For each product $i \in I$, it is sequentially paired with every other product from $i + 1$ up to the last one. Suppose product i is now paired with product j. Their prices are exchanged, and then the best price for product i is determined, taking into account the updated price of j. Next, product j is considered, and its best price is determined, taking into account the current price of i. This process involves destroying and rebuilding each product in the pair one by one. If doing so, an improved value of the UL objective function is found, the current solution is updated, and the LS2 procedure continues by examining another pair of products.

Once all pairs of products have been examined, it is guaranteed that the current solution is at least as good as the one with which the local search began in terms of the value of the UL objective function provided. If this value remains the same, the LS2 procedure terminates; otherwise, it continues iterating, considering the current solution, until no further improvements are achieved. Each iteration of the LS2 procedure has worst-case time complexity $\mathcal{O}(|I|^3 \cdot |K|)$.

4.4.3. Local search 3 (LS3)

LS3 aims to analyze the consequences of changing the price of a given product i to other possible price for that product. Once the price is changed, the products different from i are examined one by one to determine their best possible price, looking at how the UL objective function changes. The current solution is updated right after a change brings an improvement in the UL objective function.

The change of the price of the product i to other possible price can be done in different ways but the IG-RPP algorithm looks at three of them:

LS3-1 Changing the price of product i to its maximum possible value.
LS3-2 Increasing the price of product i to the next higher possible value, if any.
LS3-3 Decreasing the price of product i to the next lower possible value, if any.

Therefore, three versions of LS3 are available. Once all products have been examined with the version of the LS3 selected, it is guaranteed that the current solution will either match or exceed the value of the UL objective function of the solution with which the local search started. If the value remains unchanged, the version of LS3 which is being applied halts; otherwise, it continues iterating, considering the current solution, until no additional improvements are found. Each iteration of each version of the LS3 procedure has worst-case time complexity $\mathcal{O}(|I|^3 \cdot |K|)$.

4.4.4. When to apply the local search?

Based on preliminary tests, it was determined that incorporating local search into the algorithm yields a clear improvement in solution quality. However, a balance was sought between computational effort and solution performance. Therefore, local search is not applied to every solution but only to selected ones. The local search procedures are applied to “promising” solutions. To determine if a solution is promising or not, the average UL objective function value of a pool containing the last 100 solutions resulting from the reconstruction step is maintained. If the UL objective function of the current solution is better than the average, the current solution is passed to the LS1 procedure. After LS1 is applied, the UL objective function value of the obtained solution is compared to the best available UL objective function value. If it does not improve this value, the local search stops. Otherwise, LS2 and the three versions of LS3 in numerical order are applied as a block. Afterward, the current solution will have an equal or improved UL objective function value. If it is the same, the local search procedure stops; otherwise, LS1, LS2, and the three versions of LS3 in numerical order are applied until no further improvement is achieved. Moreover, in order to avoid unnecessary computations, an archive of the solutions to which the local search has been applied is maintained.

4.5. Acceptance criterion

Most IG algorithms have an acceptance criterion that strikes a balance between diversification and intensification. The extremes would be either accepting every solution regardless of its quality, or only accepting a solution if it improves upon the incumbent. The IG-RPP algorithm adopts a simpler compromise acceptance criterion that enables this balance. To implement this criterion, a parameter #IT is established. The current solution is accepted only if it improves the incumbent solution S^*, unless the number of iterations without improvement reaches #IT. In that case, the current solution is accepted anyway.

4.6. Stopping criterion

The stopping criterion (SC) of the IG-RPP algorithm is defined based on the number of iterations without improvement in the objective function.

5. Computational experiments

This section presents the results of the computational experiments conducted to assess the performance of the IG-RPP algorithm. These experiments were performed on a PC equipped with a 13th Gen Intel Core i9-13900F processor. The system has 64 GB of RAM and runs Windows 11 64-bit as the operating system. The IG-RPP algorithm has been implemented in C++ and compiled with Microsoft Visual C++ 19.33.

The set of instances used for testing the IG-RPP algorithm was randomly generated by Calvete et al. (2024b) and can be downloaded from https://zenodo.org/records/15082777. In addition, the code used for generating the instances is also included. The number of customers, $|K|$, takes values from the set $\{50, 100, 150, 200\}$ and the number of products, $|I|$, is determined by the number of customers. It can take values $0.1\,|K|$ and $0.5\,|K|$. These two parameters lead to eight possible combinations of customer–product sizes. For each combination, the length of the customer preference lists, $\left|S^k\right|$ for each customer $k \in K$, depends on the number of products and takes values from the set $\{0.2\,|I|\,, 0.4\,|I|\,, 0.6\,|I|\,, 0.8\,|I|\,, |I|\}$. Finally, customer budgets fall within one of the two intervals, $[1, 2\,|K|]$ and $[|K|\,, 2\,|K|]$. For each combination of these four factors, three instances are available, resulting in a collection of 240 instances of the RPP being tested. Furthermore, the preprocessing procedure developed by Calvete et al. (2019) is applied to the instances before solving them. This procedure seeks to identify, for each customer, products that they will certainly not purchase in an optimal solution. Such products can be removed from customer preference lists before solving the problem, leading to a sparser matrix of preferences.

Since the IG-RPP algorithm depends on several tunable parameters, multiple configurations of this algorithm are possible. Therefore, as a first stage, the impact of these parameters on the solution quality and the computational time invested is analyzed to determine the best configuration. Then, a second stage consists of assessing the performance of the IG-RPP algorithm in terms of both the solution's quality and the computational times required to obtain it. For this purpose, the results provided by the IG-RPP algorithm are compared against those provided by the exact solver Gurobi, and the metaheuristics proposed by Calvete et al. (2024b) and Jiménez-Cordero et al. (2025).

5.1. Selecting the algorithm configuration

Selecting the algorithm configuration consists of fixing the parameter values that yield the best results when assessing the quality of the algorithm based on the UL objective function values and the computational time. For this purpose, the levels considered for each parameter are

- Percentage of destruction (%D): 10, 20, 30, 40, 50.
- Selection of products whose price is destroyed (SPD): US, PVS.
- Acceptance criterion (#IT): 0, 250, 500, 750.
- Stopping criterion (SC): 5000, 10,000.

Each combination of these four parameter values results in a configuration of the IG-RPP algorithm. Therefore, a total of $2 \times 5 \times 2 \times 4 = 80$ configurations are available.

In order to select the best configuration of the IG-RPP algorithm, two measures are computed for each configuration: the success rate ($SRate$) and the percentage of gap ($PGap$). The $SRate$ of a given configuration is computed as the number of successes for that configuration divided by the total number of instances, which is 240, and multiplied by 100. A configuration having a success on a given instance means that the configuration reaches IG_{best}, the best value obtained for such an instance by any of the available configurations. Table 2 shows the number of successes for each

Table 2
Number of successes for each configuration

			%D				
SC	SPD	#IT	10	20	30	40	50
5000	US	0	120	150	159	159	157
		250	118	165	195	189	182
		500	110	159	190	187	188
		750	107	160	183	186	186
	PVS	0	121	151	160	165	156
		250	123	167	198	188	186
		500	112	165	199	189	182
		750	120	162	191	187	185
10000	US	0	125	156	164	169	162
		250	136	180	203	195	189
		500	123	177	204	195	191
		750	122	176	206	200	192
	PVS	0	131	162	165	166	163
		250	144	189	207	196	188
		500	130	189	210	196	187
		750	130	179	207	197	188

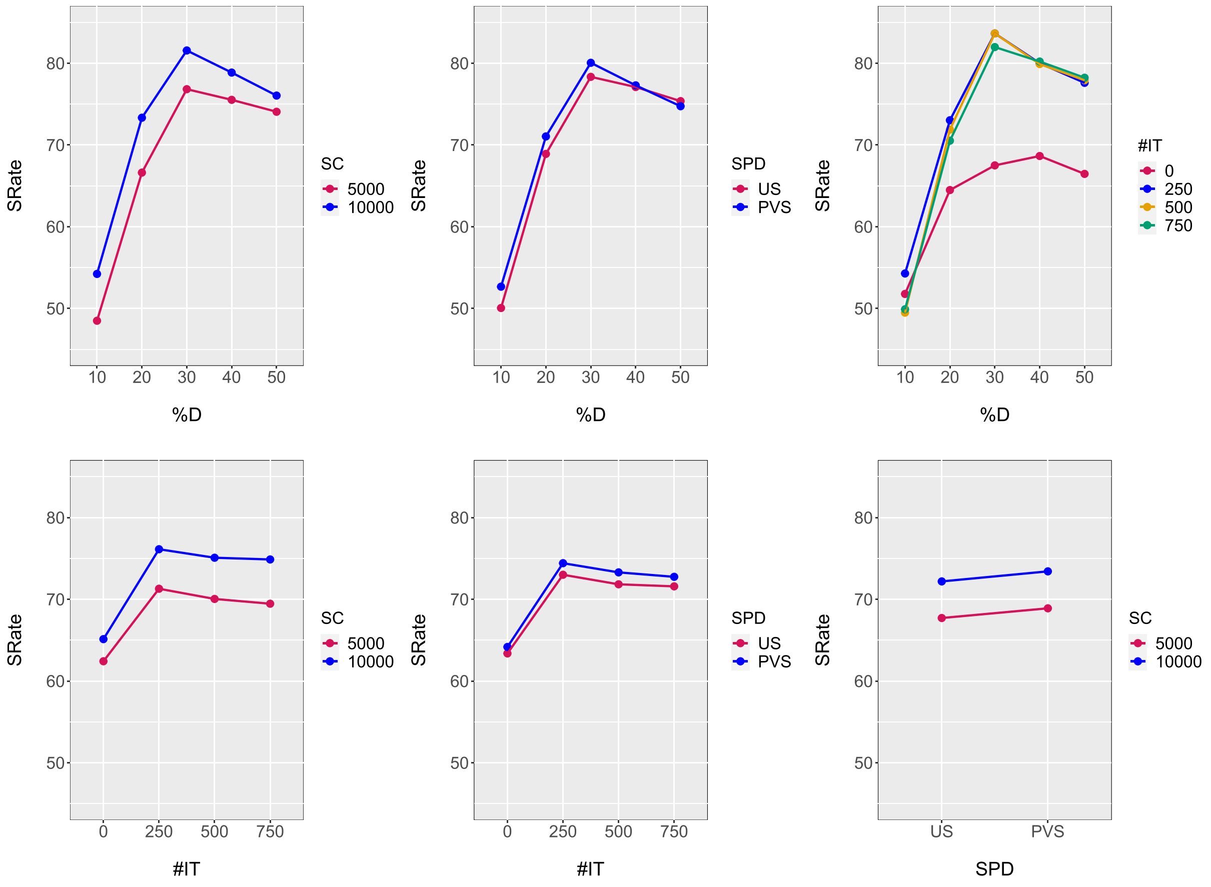

Fig. 1. Interaction effects plots for *SRate*.

configuration. Each cell in columns four to eight, rows three to eighteen, contains the value of the number of successes for the configuration described by the corresponding values in the row of the first, second, and third columns (SC, SPD, and #IT) and the corresponding column (%D).

Figure 1 shows the interaction effects plots for *SRate*, which illustrate how the effect of one parameter depends on the value of another parameter. The three top plots depict the behavior of *SRate* as %D varies from 10 to 50, with the highest values observed for %D = 30, regardless of the values of SC, SPD, or #IT from 250 to 750. In the three plots where SC defines the groups, a value of 10,000 results in a better *SRate* than 5000. Regarding SPD, there are almost no differences in *SRate* between its two values. Concerning #IT, it is worth noting that #IT = 0 is much worse than the other values.

Previous information on the influence of the parameters of the IG-RPP algorithm in *SRate* is complemented by Fig. 2, which displays a heatmap of *SRate* across the 80 configurations. Heatmaps are an effective tool for the graphical representation of data, where darker or more intense colors represent higher values, while lighter colors indicate lower values. The number inside each cell represents *SRate*. From this information, configurations with an *SRate* greater than 80% are selected as promising configurations. This results in 16 out of 80 configurations.

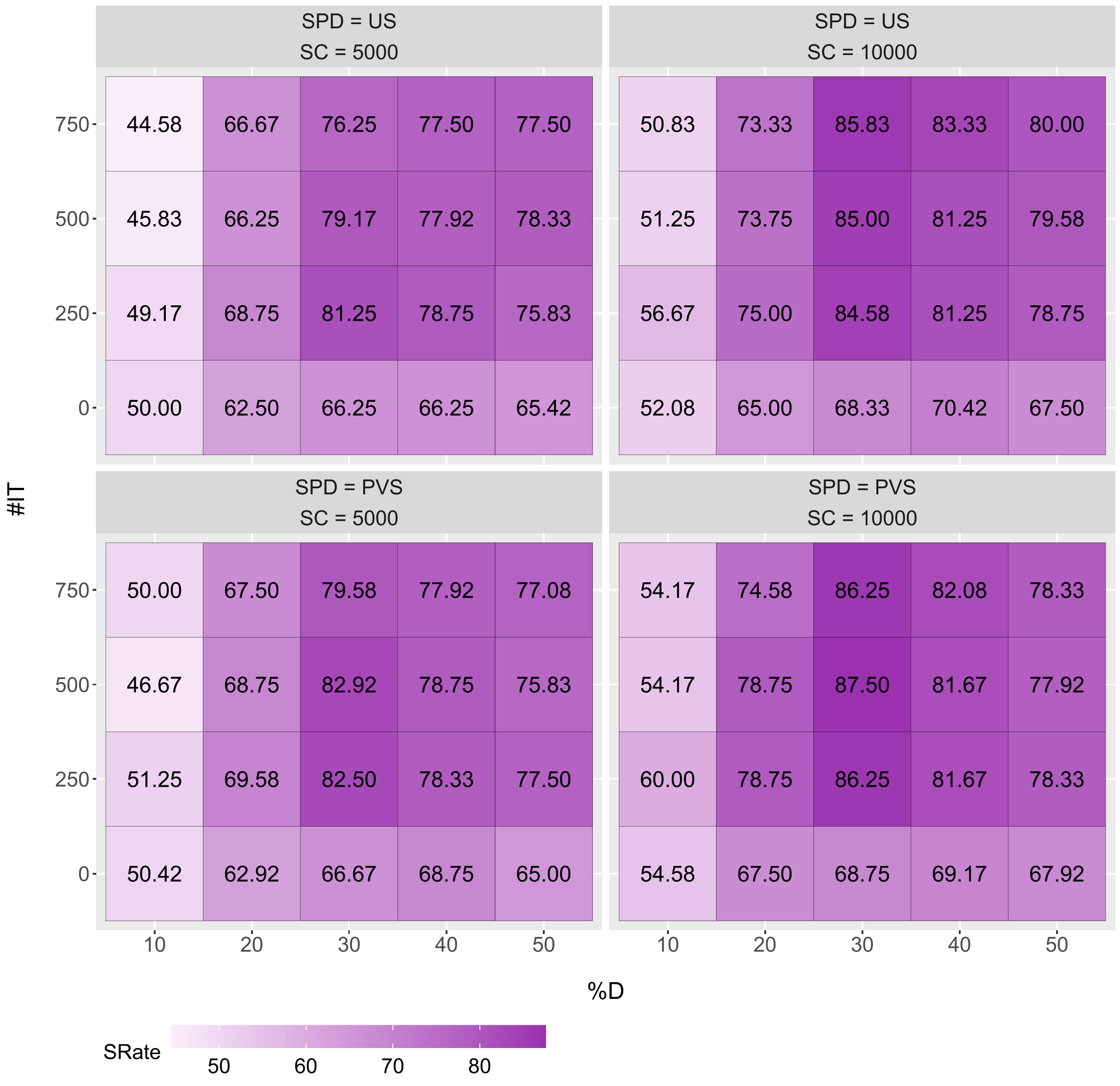

Fig. 2. Heatmap of the *SRate* for each configuration.

To assess the distance of the UL objective function found by a configuration of the IG-RPP algorithm and IG_{best}, $PGap$ is computed for each instance and each configuration as

$$PGap = \frac{IG_{best} - Value}{IG_{best}} \times 100,$$

where $Value$ denotes the UL objective function of the instance found by the corresponding configuration. Table 3 and Fig. 3 follow a similar structure to previously described in Table 2 and Fig. 2,

Table 3
Average *PGap* for each configuration. The value corresponding to the 16 pre-selected configurations is highlighted in bold

			%D				
SC	SPD	#IT	10	20	30	40	50
5000	US	0	0.158	0.100	0.043	0.048	0.050
		250	0.160	0.076	**0.007**	0.008	0.015
		500	0.169	0.083	0.013	0.012	0.011
		750	0.178	0.083	0.021	0.017	0.012
	PVS	0	0.146	0.094	0.047	0.044	0.053
		250	0.153	0.070	**0.008**	0.014	0.017
		500	0.156	0.075	**0.009**	0.013	0.016
		750	0.170	0.086	0.015	0.015	0.018
10000	US	0	0.146	0.093	0.040	0.036	0.045
		250	0.128	0.061	**0.005**	**0.006**	0.011
		500	0.141	0.067	**0.005**	**0.007**	0.010
		750	0.151	0.069	**0.005**	**0.006**	**0.009**
	PVS	0	0.140	0.087	0.040	0.036	0.047
		250	0.117	0.057	**0.004**	**0.008**	0.014
		500	0.132	0.060	**0.003**	**0.006**	0.014
		750	0.152	0.071	**0.006**	**0.008**	0.015

but refer to the average and maximum *PGap*, respectively. From Table 3, we conclude that the average does not help in deciding whether to discard any of the previously selected 16 configurations, as they all have very similar average values, under 0.009%. Regarding Fig. 3, the focus is on configurations with small maximum *PGap*s, under 0.13%. Then, four configurations combining a high *SRate* with a low maximum *PGap* are selected. These configurations are presented in Table 4. Note that all four configurations have %D = 30% and SC = 10,000. Additionally, they combine both SPD methods with #IT = 250 and #IT = 500.

Next, the focus is on the computational time invested by the IG-RPP algorithm. The reported computational time corresponds to the total time required to execute the entire algorithm, including initialization, destruction, and reconstruction phases, local searches, and the evaluation of solutions. It is worth noting that, regardless of the 80 configurations or the instance considered, the computational time ranges from 0.037 to 40.196 seconds. Figure 4 displays a boxplot of the computational time invested by the four configurations in Table 4, with instances grouped based on the number of customers. As expected, the computational time increases as the number of customers grows. Nevertheless, the computational time is always less than 38 seconds and the 75th percentile is below 20 seconds. It is noticeable that the four configurations exhibit a similar behavior within each group of instances. Thus, the first conclusion is that, in terms of computational time, the IG-RPP algorithm is highly efficient, regardless of the selected configuration. Therefore, to choose a configuration for the next stage of the computational experiments, *SRate* is prioritized, and the selected configuration is %D = 30, SPD = PVS, #IT = 500, and SC = 10,000. From now on, IG-RPP always refers to such a configuration. Tables 5 and 6 report the UL objective function values and the computational times for the 240 instances under the selected configuration of the algorithm.

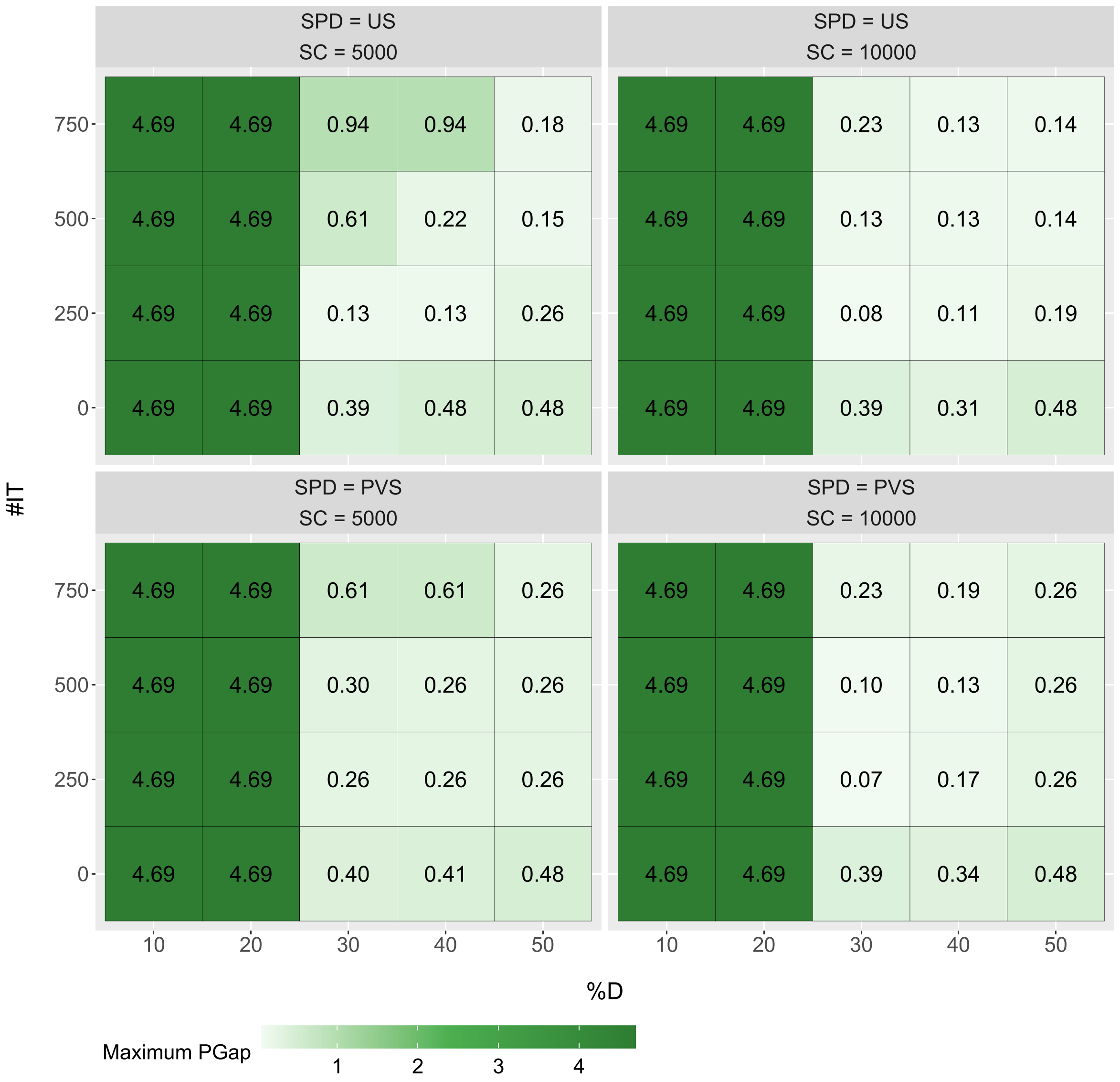

Fig. 3. Heatmap of the maximum *PGap* across the 240 instances.

Table 4
Configurations with a good balance between high *SRate* and low maximum *PGap*

Configuration parameters					
%D	SPD	#IT	SC	*SRate*	Maximum *PGap*
30	US	250	10000	84.58%	0.08
30	US	500	10000	85.00%	0.13
30	PVS	250	10000	86.25%	0.07
30	PVS	500	10000	87.50%	0.10

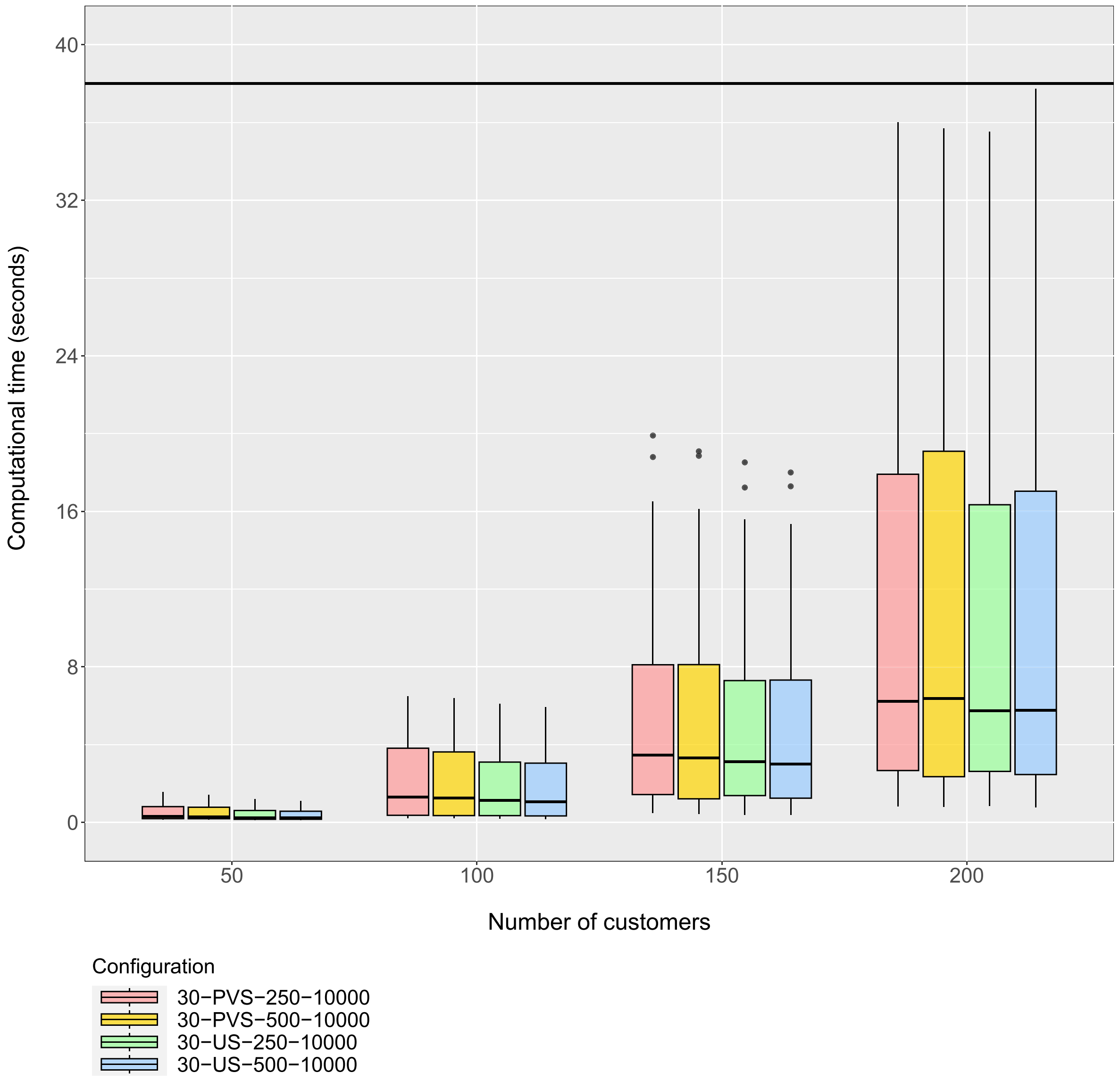

Fig. 4. Boxplot of the computational time of the four configurations in Table 4, according to the number of customers in the instances.

5.2. Evaluating the algorithm performance

In this part of the computational study, the solutions found by the IG-RPP algorithm are compared with those provided by the exact resolution with Gurobi (GRB) and by the evolutionary algorithm developed by Calvete et al. (2024b), from now on named EV-RPP algorithm. In addition, the IG-RPP algorithm is compared with the heuristic resolution proposed by Jiménez-Cordero et al. (2025) using their instances. The reported computational time for IG-RPP and EV-RPP corresponds to

Table 5
UL objective function values and computational times in seconds for the instances 1–120 under the selected algorithm configuration (%D = 30, SPD = PVS, #IT = 500, and SC = 10,000). A and B refer to the budget intervals $[1, 2|K|]$ and $[|K|, 2|K|]$, respectively

$\|K\| = 50, \|I\| = 5$					$\|K\| = 50, \|I\| = 25$					$\|K\| = 100, \|I\| = 10$					$\|K\| = 100, \|I\| = 50$				
Ins.	$\|S^k\|$	b^k	UL obj	CPU	Ins.	$\|S^k\|$	b^k	UL obj	CPU	Ins.	$\|S^k\|$	b^k	UL obj	CPU	Ins.	$\|S^k\|$	b^k	UL obj	CPU
1	1	A	1715	0.16	31	5	A	2131	0.35	61	2	A	7717	0.22	91	10	A	9616	1.94
2	1	A	1788	0.14	32	5	A	2467	0.42	62	2	A	6470	0.22	92	10	A	9196	2.54
3	1	A	1645	0.15	33	5	A	1956	0.38	63	2	A	6272	0.21	93	10	A	8503	2.15
4	1	B	2588	0.15	34	5	B	3663	0.47	64	2	B	12550	0.26	94	10	B	14827	3.05
5	1	B	2778	0.15	35	5	B	3634	0.76	65	2	B	11905	0.25	95	10	B	14491	3.25
6	1	B	2579	0.15	36	5	B	3526	0.57	66	2	B	12014	0.24	96	10	B	14474	3.36
7	2	A	1702	0.15	37	10	A	2362	0.46	67	4	A	7562	0.28	97	20	A	8561	2.83
8	2	A	1550	0.14	38	10	A	2219	0.39	68	4	A	7773	0.27	98	20	A	8552	2.57
9	2	A	2165	0.17	39	10	A	2272	0.51	69	4	A	7681	0.29	99	20	A	9487	2.36
10	2	B	3037	0.17	40	10	B	3466	1.10	70	4	B	12855	0.33	100	20	B	15046	4.12
11	2	B	3105	0.17	41	10	B	3531	0.86	71	4	B	12707	0.32	101	20	B	14274	4.29
12	2	B	2998	0.17	42	10	B	3595	1.02	72	4	B	12968	0.33	102	20	B	14866	4.30
13	3	A	1568	0.16	43	15	A	2386	0.72	73	6	A	7591	0.33	103	30	A	9255	3.35
14	3	A	1619	0.18	44	15	A	2327	0.74	74	6	A	8422	0.34	104	30	A	10207	2.97
15	3	A	1833	0.18	45	15	A	2467	0.50	75	6	A	7897	0.34	105	30	A	9544	2.73
16	3	B	3068	0.20	46	15	B	3685	1.25	76	6	B	12860	0.40	106	30	B	14393	4.67
17	3	B	3119	0.21	47	15	B	3766	0.99	77	6	B	13070	0.41	107	30	B	14786	4.22
18	3	B	3132	0.19	48	15	B	3683	0.89	78	6	B	12740	0.46	108	30	B	14692	4.48
19	4	A	2229	0.19	49	20	A	2176	0.69	79	8	A	7565	0.35	109	40	A	10200	3.39
20	4	A	1996	0.20	50	20	A	2682	0.81	80	8	A	7741	0.42	110	40	A	10110	3.57
21	4	A	1635	0.21	51	20	A	2266	0.77	81	8	A	7621	0.37	111	40	A	9983	3.42
22	4	B	3177	0.24	52	20	B	3611	1.34	82	8	B	13559	0.48	112	40	B	14956	4.97
23	4	B	3253	0.21	53	20	B	3551	1.21	83	8	B	13530	0.47	113	40	B	14516	4.62
24	4	B	3198	0.21	54	20	B	3568	1.16	84	8	B	13420	0.47	114	40	B	14901	5.14
25	5	A	1739	0.22	55	25	A	2235	0.84	85	10	A	7666	0.44	115	50	A	9969	3.78
26	5	A	2012	0.21	56	25	A	2299	0.93	86	10	A	8037	0.43	116	50	A	9909	4.04
27	5	A	1929	0.22	57	25	A	2340	0.71	87	10	A	8067	0.40	117	50	A	9248	3.96
28	5	B	3237	0.22	58	25	B	3723	1.42	88	10	B	13304	0.53	118	50	B	14661	6.40
29	5	B	3177	0.24	59	25	B	3528	1.24	89	10	B	13485	0.58	119	50	B	15081	6.33
30	5	B	3129	0.22	60	25	B	3687	1.29	90	10	B	13630	0.54	120	50	B	14428	6.13

Table 6
UL objective function values and computational times in seconds for the instances 120–240 under the selected algorithm configuration (%D = 30, SPD = PVS, #IT = 500, and SC = 10,000). A and B refer to the budget intervals $[1, 2|K|]$ and $[|K|, 2|K|]$, respectively

$\|K\| = 150, \|I\| = 15$					$\|K\| = 150, \|I\| = 75$					$\|K\| = 200, \|I\| = 20$					$\|K\| = 200, \|I\| = 100$				
Ins.	$\|S^k\|$	b^k	UL obj	CPU	Ins.	$\|S^k\|$	b^k	UL obj	CPU	Ins.	$\|S^k\|$	b^k	UL obj	CPU	Ins.	$\|S^k\|$	b^k	UL obj	CPU
121	3	A	15688	0.46	151	15	A	19743	4.11	181	4	A	28276	0.91	211	20	A	36003	7.42
122	3	A	17015	0.45	152	15	A	21114	4.43	182	4	A	29084	0.91	212	20	A	40844	7.21
123	3	A	16014	0.52	153	15	A	21035	4.82	183	4	A	28616	0.79	213	20	A	38429	7.27
124	3	B	28383	0.57	154	15	B	33915	6.41	184	4	B	51077	1.32	214	20	B	58676	11.06
125	3	B	28037	0.66	155	15	B	32140	6.43	185	4	B	50555	1.89	215	20	B	58020	10.84
126	3	B	28704	0.60	156	15	B	32854	6.30	186	4	B	49957	1.31	216	20	B	58733	12.10
127	6	A	18777	0.64	157	30	A	21797	5.76	187	8	A	30305	1.51	217	40	A	36819	9.63
128	6	A	16107	0.91	158	30	A	20231	5.91	188	8	A	31887	2.17	218	40	A	38245	8.81
129	6	A	16600	0.78	159	30	A	19936	5.16	189	8	A	27845	1.65	219	40	A	39422	12.09
130	6	B	29367	0.97	160	30	B	32580	8.70	190	8	B	52529	2.36	220	40	B	59134	16.47
131	6	B	29285	1.34	161	30	B	33424	9.49	191	8	B	52914	2.22	221	40	B	59691	19.77
132	6	B	29556	1.18	162	30	B	32388	8.04	192	8	B	53750	1.95	222	40	B	59766	17.29
133	9	A	18555	1.13	163	45	A	21048	6.79	193	12	A	31541	1.87	223	60	A	38028	14.06
134	9	A	17540	0.83	164	45	A	21825	6.91	194	12	A	33744	2.24	224	60	A	40721	15.48
135	9	A	16963	1.08	165	45	A	21052	6.77	195	12	A	32837	1.47	225	60	A	38770	15.51
136	9	B	29707	1.61	166	45	B	33192	10.53	196	12	B	54179	3.28	226	60	B	59836	20.28
137	9	B	30771	1.46	167	45	B	33215	12.56	197	12	B	53411	3.39	227	60	B	59979	21.68
138	9	B	30543	1.31	168	45	B	33479	10.63	198	12	B	54751	2.93	228	60	B	58700	21.88
139	12	A	18562	1.10	169	60	A	20871	7.68	199	16	A	33775	2.33	229	80	A	37164	18.96
140	12	A	17452	1.34	170	60	A	20975	8.35	200	16	A	33623	2.49	230	80	A	42978	25.92
141	12	A	18790	1.48	171	60	A	20718	7.99	201	16	A	32655	2.93	231	80	A	38271	20.41
142	12	B	30618	2.14	172	60	B	32669	15.23	202	16	B	56119	4.45	232	80	B	59049	28.51
143	12	B	30944	1.49	173	60	B	33057	12.58	203	16	B	54890	4.82	233	80	B	59023	26.83
144	12	B	29841	2.09	174	60	B	32943	15.61	204	16	B	54593	3.62	234	80	B	59571	27.51
145	15	A	17882	1.56	175	75	A	20215	9.21	205	20	A	33694	2.63	235	100	A	37897	19.78
146	15	A	18452	2.07	176	75	A	21686	10.13	206	20	A	31970	3.09	236	100	A	40755	21.33
147	15	A	16767	1.22	177	75	A	19260	11.15	207	20	A	32078	2.93	237	100	A	38844	19.47
148	15	B	30885	1.79	178	75	B	32828	18.86	208	20	B	55329	3.84	238	100	B	58863	35.70
149	15	B	31654	2.52	179	75	B	33661	19.09	209	20	B	55064	5.51	239	100	B	59732	34.96
150	15	B	30864	2.14	180	75	B	32704	16.12	210	20	B	54401	4.78	240	100	B	59275	35.40

Table 7
Summary of the number of instances with a solution equal to or worse than the optimal solution. The symbol "–" refers to 0. CPU time is measured in seconds

	$\|K\|$	# Ins	*RD* Min.	*RD* Mean	*RD* Max.	CPU time IG-RPP Min.	CPU time IG-RPP Mean	CPU time IG-RPP Max.	CPU time GRB Min.	CPU time GRB Mean	CPU time GRB Max.
IG-RPP = GRB	50	60	—	—	—	0.14	0.51	1.42	0.00	2.43	21.88
	100	52	—	—	—	0.21	1.70	5.14	0.08	343.40	2042.43
	150	32	—	—	—	0.45	1.92	6.30	1.91	957.54	3363.36
	200	8	—	—	—	0.79	1.29	1.89	39.50	844.26	2979.48
	All	152	—	—	—	0.14	1.25	6.30	0.00	364.46	3363.36
IG-RPP < GRB	50	—	—	—	—	—	—	—	—	—	—
	100	3	0.01	0.01	0.02	3.25	3.32	3.36	12.59	60.57	153.25
	150	3	<0.01	0.02	0.03	5.16	6.00	6.43	304.89	1223.14	2425.19
	200	3	<0.01	0.03	0.09	7.21	7.30	7.42	384.64	979.43	1372.66
	All	9	<0.01	0.02	0.09	3.25	5.54	7.42	12.59	754.38	2425.19

the total time required to execute each algorithm. For Gurobi, it refers to the total time spent, with a time limit of 3600 seconds.

5.2.1. Comparison with the exact resolution

The exact resolution of the instances was performed by solving formulation (4a)–(4j) using Python 3.10 and the off-the-shelf commercial solver Gurobi 10.0.3. All Gurobi parameters were left at their default values, except for Threads set to one, TimeLimit set to 3600 seconds, and the Gurobi parameters MIPGapAbs and MIPGap adjusted to 0.999 and 10^{-5}, respectively. When the Gurobi run is interrupted upon reaching the time limit, the best solution available at that moment is saved.

To better measure the quality of the solution found by the IG-RPP algorithm, the relative difference (*RD*) is defined as

$$RD = \begin{cases} \dfrac{Z_{GRB} - Z_{IG}}{Z_{GRB}} \times 100 & \text{if } Z_{GRB} > Z_{IG} \\ \dfrac{Z_{IG} - Z_{GRB}}{Z_{GRB}} \times 100 & \text{if } Z_{GRB} < Z_{IG} \end{cases}, \tag{6}$$

where Z_{IG} represents the UL objective function value of the solution found by the IG-RPP algorithm and Z_{GRB} denotes the UL objective function value obtained from GRB. Obviously, if $Z_{GRB} = Z_{IG}$, $RD = 0$.

GRB is able to solve 161 out of 240 instances to optimality. Table 7 summarizes the comparison between IG-RPP and GRB for the instances in which GRB provides the optimal solution. The first column shows if IG-RPP reaches the optimal solution or not. The second column indicates the number of customers. The third column gives the number of instances. The fourth, fifth, and sixth columns show the minimum, the average, and the maximum values of the *RD*, respectively. The seventh, eighth, and ninth columns present the minimum, the average, and the maximum values of the CPU time in seconds used by IG-RPP to solve the instances, respectively. Lastly, the final three columns provide the same information for the CPU time spent by GRB. The symbol "–" means 0.

Table 8
Summary of the number of instances for which GRB stops due to the stopping criterion. The symbol "–" refers to 0. CPU time is measured in seconds. CPU time invested by GRB is not included as it is always equal to 3600 s

	$\|K\|$	# Ins	*RD*			CPU time IG-RPP		
			Min.	Mean	Max.	Min.	Mean	Max.
IG-RPP > GRB	100	3	0.01	0.05	0.10	4.97	5.83	6.40
	150	13	0.02	0.12	0.58	8.35	12.57	19.09
	200	31	0.01	0.21	0.79	1.47	16.58	35.70
	All	47	0.01	0.18	0.79	1.47	14.79	35.70
IG-RPP = GRB	100	1	—	—	—	3.96	3.96	3.96
	150	11	—	—	—	1.34	6.14	16.12
	200	15	—	—	—	1.87	3.49	12.09
	All	27	—	—	—	1.34	4.58	16.12
IG-RPP < GRB	100	1	0.01	0.01	0.01	6.33	6.33	6.33
	150	1	<0.01	<0.01	<0.01	8.70	8.70	8.70
	200	3	0.02	0.04	0.05	10.84	14.24	19.77
	All	5	<0.01	0.02	0.05	6.33	11.55	19.77

IG-RPP finds the optimal solution in 152 out of 161 instances, with computational times never exceeding 7 seconds, while GRB, in the worst case, requires nearly 3000 seconds. Furthermore, in the nine instances where IG-RPP does not reach the optimal solution, the *RD* is very small, with a worst-case value of just 0.09%. In terms of computational time, IG-RPP requires significantly less time than GRB, taking no more than 8 seconds in the worst case, compared to GRB's maximum time of almost 2500 seconds.

Table 8 presents similar information, but for the instances where GRB terminates due to the stopping criterion, meaning the optimal solution is not guaranteed. This table does not display the CPU time used by GRB, as it is always 3600 seconds. Additionally, since all instances with 50 customers are solved to optimality, this case is also omitted. Three categories can be identified: instances where IG-RPP outperforms GRB ($Z_{IG-RPP} > Z_{GRB}$), which account for 47 out of 79; instances where both algorithms yield the same objective function value ($Z_{GRB} = Z_{IG-RPP}$), comprising 27 out of 79; and instances where GRB performs better than IG-RPP ($Z_{GRB} > Z_{IG-RPP}$), which make up 5 out of 79. Thus, IG-RPP provides the best UL objective value in a significantly higher number of instances. Additionally, when IG-RPP outperforms GRB, *RD* is higher compared to the reverse scenario, thus proving that the improvement is greater. Furthermore, it is worth mentioning that the longest computational time for IG-RPP is under 36 seconds, compared to 3600 seconds for GRB, further confirming IG-RPP's superior performance over GRB.

5.2.2. Comparison with EV-RPP

This section compares the performance of IG-RPP and EV-RPP. *RD* is defined as in (6), substituting Z_{GRB} by Z_{EV}, the objective function value of the solution provided by EV-RPP. Table 9 is similar to Table 8, but includes all 240 instances. Again, IG-RPP clearly outperforms EV-RPP. IG-RPP outperforms EV-RPP in 39 out of 260 instances, performs equally in 190, and is worse in 11. Additionally, when IG-RPP outperforms EV-RPP, the *RD* is higher than in the reverse case.

Table 9
Summary of the comparison between IG-RPP and EV-RPP. The symbol "–" refers to 0. CPU time is measured in seconds

			RD			CPU time IG-RPP			CPU time EV-RPP		
	$\|K\|$	# Ins	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
IG-RPP > EV-RPP	50	1	0.22	0.22	0.22	1.25	1.25	1.25	18.83	18.83	18.83
	100	8	0.01	0.09	0.34	0.22	3.76	6.40	16.41	68.98	109.78
	150	12	0.01	0.07	0.18	6.30	11.62	19.09	89.45	206.94	341.65
	200	18	<0.01	0.05	0.11	1.89	18.11	35.70	63.65	332.65	611.85
	All	39	<0.01	0.07	0.34	0.22	12.74	35.70	16.41	231.84	611.85
IG-RPP = EV-RPP	50	59	—	—	—	0.14	0.49	1.42	2.81	13.88	30.90
	100	52	—	—	—	0.21	1.84	6.33	16.55	50.18	149.08
	150	46	—	—	—	0.45	3.61	16.12	30.37	104.37	575.90
	200	33	—	—	—	0.79	4.43	35.40	48.55	126.40	444.93
	All	190	—	—	—	0.14	2.30	35.40	2.81	65.27	575.90
IG-RPP < EV-RPP	50	—	—	—	—	—	—	—	—	—	—
	100	—	—	—	—	—	—	—	—	—	—
	150	2	<0.01	0.01	0.02	5.16	6.93	8.70	118.85	119.32	119.80
	200	9	<0.01	0.03	0.09	7.27	18.79	34.96	146.74	325.37	556.25
	All	11	<0.01	0.03	0.09	5.16	16.63	34.96	118.85	287.91	556.25

Moreover, the longest computational time for IG-RPP is under 36 seconds, while EV-RPP requires up to 612 seconds.

5.2.3. *Comparison with the algorithms in Jiménez-Cordero et al. (2025)*

The numerical experiments in Jiménez-Cordero et al. (2025) involve only three small instances where $(|K|, |I|) \in \{(30, 5), (30, 25), (60, 50)\}$. These instances can be downloaded from https://github.com/groupoasys/RPP_VNS_data. Since the authors do not specify whether the preprocessing step was applied, we present our results using the unpreprocessed instances to ensure a fair comparison. Additionally, since they solved each instance 1000 times, we have also solved each instance 1000 times using IG-RPP and EV-RPP. In addition, we have solved the instances using GRB. It is important to note that Jiménez-Cordero et al. (2025) indicate that GRB, using formulation (3a)–(3g), takes 10 seconds to solve the smallest instance and cannot solve the two other instances in under 600 seconds. In their computational setup, they mention that their experiments were run on a Linux-based server with CPUs clocking at 2.6 GHz, 1 thread, and 8 GB of RAM, using Python 3.11.4 and Gurobi 10.0.3. When we solved the three instances using formulation (4a)–(4j), GRB found the optimal solution in 0.41, 0.53, and 6.40 seconds, respectively. The optimal objective function values were 807, 1042, and 2017, respectively.

Table 10 presents the results of this part of the experiment. The first column shows the number of customers and products in each instance, while the second column indicates the number of runs in which the optimal solution was achieved. The fourth and fifth columns display the minimum and maximum objective function values across the 1000 runs. The sixth and seventh columns show the minimum and maximum computational time invested throughout those runs.

When $(|K|, |I|) = (30, 5)$, the best version of the algorithms proposed in Jiménez-Cordero et al. (2025) is able to find the optimal solution in all runs, although no computational times are provided. When $(|K|, |I|) = (30, 25)$, the best version reaches the optimal solution in all runs from the second

Table 10
Summary of the results obtained in the instances of Jiménez-Cordero et al. (2025). CPU time is measured in seconds

		IG-RPP			
		Z_{IG}		CPU time	
Instance	Optimal runs	Min.	Max.	Min.	Max.
(30, 5)	1000	807	807	0.15	0.17
(30, 25)	1000	1042	1042	0.48	0.63
(60, 50)	1000	2017	2017	2.66	2.99
		EV-RPP			
		Z_{EV}		CPU time	
Instance	Optimal runs	Min.	Max.	Min.	Max.
(30, 5)	1000	807	807	5.37	5.79
(30, 25)	1000	1042	1042	10.90	14.30
(60, 50)	828	2016	2017	38.90	81.30

50. Finally, when $(|K|, |I|) = (60, 50)$, even the best approach is unable to find the optimal solution in all runs when a 600-second time limit is used as the stopping criterion, obtaining the optimal solution just as an outlier. Given these results, it is clear that IG-RPP outperforms all the algorithms proposed in Jiménez-Cordero et al. (2025), as it consistently finds the optimal solution in every run, and the computational times are very small, almost negligible.

Concerning the EV-RPP algorithm, although it is not the focus of this paper, it is clear that EV-RPP also outperforms the algorithms in Jiménez-Cordero et al. (2025) both in terms of solution quality and computational time. It consistently finds the optimal solution in all 1000 runs for the two smallest instances and finds the optimal solution in 828 out of 1000 runs for the larger instance. In the other 172 runs, it provides a solution that is only one unit worse in the UL objective function value.

5.2.4. Best UL objective function

To enable future comparisons with other algorithms developed to solve the RPP, the Table A.1 in the Appendix provides the best UL objective function value ever achieved for each of the 240 instances tested. The term best solution refers to the one obtained using any of the 80 configurations of the IG-RPP algorithm, the EV-RPP algorithm, or the best result provided by the Gurobi solver.

6. Conclusions

This paper addresses the RPP and develops IG-RPP, an IG-based metaheuristic algorithm with local search, which proves to be a suitable method to solve the RPP, particularly to tackle large-scale instances of the problem, where exact methods become impractical. IG-RPP follows the general structure of an IG algorithm: the initialization, which in this case consists of generating an initial solution using a greedy algorithm, the iteration process, which involves the partial construction and reconstruction of the incumbent solution, the application of a local search phase, which in this particular case includes several local searches specifically designed for the RPP, and, finally, the acceptance criterion to decide whether a new solution is admitted or not.

In the algorithm design, four parameters have been tuned, resulting in a total of 80 configurations being evaluated. The computational experiments have highlighted the effect of the percentage of destruction on the quality of the obtained solutions, which should be at least 30%. Additionally, configurations with a maximum number of iterations set to 10,000 have shown better performance in terms of solution quality. The algorithm design includes a method more suited to the RPP than the random selection of components to destroy from the solution. However, the results do not show a significant difference between the two methods. Regarding the proposed new acceptance criterion based on #IT, computational experiments showed that there was a need to consider a number of iterations strictly greater than 0.

Moreover, to assess its performance, IG-RPP has been compared with existing resolution methods in the literature by solving a large number of instances. Computational experiments confirm the effectiveness of IG-RPP in terms of both solution quality and computational efficiency. In fact, the proposed algorithm clearly outperforms all previously published resolution methods.

Future research could explore extensions of the IG approach to tackle additional problem variants, such as the RPPT or the CRPP. Another future research direction could explore applying ad hoc modifications of this algorithm to other problems beyond the RPP that share a similar structure of prices and customer preferences.

Acknowledgments

This research has been funded by Agencia Estatal de Investigación del Ministerio de Ciencia, Innovación y Universidades (Gobierno de España) under grants PID2022-139543OB-C43 and TED2021-130961B-I00, and by Gobierno de Aragón under grant E41-23R.

References

Aussel, D., Egea, C., Schmidt, M., 2025. A tutorial on solving single-leader-multi-follower problems using SOS1 reformulations. *International Transactions in Operational Research* 32, 3, 1227–1250.

Bard, J.F., 1998. *Practical Bilevel Optimization. Algorithms and Applications*. Kluwer Academic Publishers, Dordrecht.

Benati, S., Leal, M., Puerto, J., 2025. Bilevel portfolio optimization with ordered pricing models. *Omega* 133, 103274.

Calvete, H.I., Domínguez, C., Galé, C., Labbé, M., Marín, A., 2019. The rank pricing problem: models and branch-and-cut algorithms. *Computers and Operations Research* 105, 12–31.

Calvete, H.I., Galé, C., 2007. Linear bilevel multi-follower programming with independent followers. *Journal of Global Optimization* 39, 409–417.

Calvete, H.I., Galé, C., Hernández, A., Iranzo, J.A., 2024a. A bilevel approach to the facility location problem with customer preferences under a mill pricing policy. *Mathematics* 12, 22, 3459.

Calvete, H.I., Galé, C., Hernández, A., Iranzo, J.A., 2024b. An evolutionary algorithm for the rank pricing problem. In Sevaux, M., Olteanu, A.L., Pardo, E.G., Sifaleras, A. and Makboul, S. (eds), *Metaheuristics. MIC 2024*. Lecture Notes in Computer Science, vol. 14754. Springer, Cham, pp. 360–366.

Calvete, H.I., Galé, C., Hernández, A., Iranzo, J.A., 2024c. A novel approach to pessimistic bilevel problems. An application to the rank pricing problem with ties. *Optimization*. 74(12): 2823–2856.

Calvete, H.I., Galé, C., Hernández, A., Iranzo, J.A., 2025. The facility location problem with ties in customer preferences: a pessimistic bilevel approach. In *Monografías Matemáticas García de Galdeano. Universidad de Zaragoza*. To appear.

Calvete, H.I., Galé, C., Iranzo, J.A., Camacho-Vallejo, J.F., Casas-Ramírez, M.S., 2020. A matheuristic for solving the bilevel approach of the facility location problem with cardinality constraints and preferences. *Computers and Operations Research* 124, 105066.

Camacho-Vallejo, J.F., Cordero-Franco, A.E., González-Ramírez, R.G., 2014. Solving the bilevel facility location problem under preferences by a Stackelberg-evolutionary algorithm. *Mathematical Problems in Engineering* 2014, 430243.

Colson, B., Marcotte, P., Savard, G., 2007. An overview of bilevel optimization. *Annals of Operations Research* 153, 235–256.

Dao, S.D., Mallégol, A., Meyer, P., Mohammadi, M., Loyer, S., 2022. A hybrid iterated greedy algorithm for hydrographic survey routing problem. *Marine Geodesy* 45, 1, 75–100.

Demir, Y., 2024. An iterated greedy algorithm for the planning of yarn-dyeing boilers. *International Transactions in Operational Research* 31, 115–139.

Dempe, S., 2002. *Foundations of Bilevel Programming*. Kluwer Academic Publishers, Dordrecht.

Dempe, S., Zemkoho, A., 2020. *Bilevel Optimization. Advances and Next Challenges*. Springer, Cham.

Dias Garcia, J., Bodin, G., Street, A., 2024. Bileveljump.jl: modeling and solving bilevel optimization problems in Julia. *INFORMS Journal on Computing* 36, 2, 327–335.

Ding, J.Y., Song, S., Gupta, J.N., Zhang, R., Chiong, R., Wu, C., 2015. An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Applied Soft Computing* 30, 604–613.

Domínguez, C., Labbé, M., Marín, A., 2021. The rank pricing problem with ties. *European Journal of Operational Research* 294, 2, 492–506.

Domínguez, C., Labbé, M., Marín, A., 2022. Mixed-integer formulations for the capacitated rank pricing problem with envy. *Computers and Operations Research* 140, 105664.

Feng, X., Zhao, F., Jiang, G., Tao, T., Mei, X., 2024. A tabu memory based iterated greedy algorithm for the distributed heterogeneous permutation flowshop scheduling problem with the total tardiness criterion. *Expert Systems with Applications* 238, 121790.

Fernandez-Viagas, V., Framinan, J.M., 2019. A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. *Computers and Operations Research* 112, 104767.

Gao, Y., Wang, X., Yang, K., Ma, J., Jiang, L., Luo, Q., 2025. A bilevel programming model for optimizing location, capacity, and pricing decisions of origin warehouses in an agricultural supply chain. *Applied Mathematical Modelling* 145, 116145.

Hanjoul, P., Peeters, D., 1987. A facility location problem with clients' preference orderings. *Regional Science and Urban Economics* 17, 3, 451–473.

Hansen, P., Kochetov, Y., Mladenović, N., 2004. Lower bounds for the uncapacitated facility location problem with user preferences. Les Cahiers du GERAD, G-2004-24..

Huerta-Muñoz, D.L., Ríos-Mercado, R.Z., López-Pérez, J.F., 2025. Iterated greedy local search for the order picking problem considering storage location and order batching decisions. *Transportation Research Part E: Logistics and Transportation Review* 201, 104157.

Jacquet, Q., van Ackooij, W., Alasseur, C., Gaubert, S., 2024. Quadratic regularization of bilevel pricing problems and application to electricity retail markets. *European Journal of Operational Research* 313, 3, 841–857.

Jiménez-Cordero, A., Pineda, S., Morales, J.M., 2025. An enhanced heuristic framework for solving the rank pricing problem. *Expert Systems with Applications* 279, 127122.

Karabulut, K., Tasgetiren, M.F., 2014. A variable iterated greedy algorithm for the traveling salesman problem with time windows. *Information Sciences* 279, 383–395.

Kleinert, T., Labbé, M., Ljubić, I., Schmidt, M., 2021. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal on Computational Optimization* 9, 100007.

Kochetov, Y.A., Panin, A.A., Plyasunov, A.V., 2015. Comparison of metaheuristics for the bilevel facility location and mill pricing problem. *Journal of Applied and Industrial Mathematics* 9, 3, 392–401.

Labbé, M., Violin, A., 2016. Bilevel programming and price setting problems. *Annals of Operations Research* 240, 141–169.

Lin, Y.H., Tian, Q., 2023. Facility location and pricing problem: discretized mill price and exact algorithms. *European Journal of Operations Research* 308, 2, 568–580.

Lin, Y.H., Tian, Q., He, D., Wang, Y., 2024. Revisiting Stackelberg p-median problem with user preferences. *Computers and Operations Research* 161, 106429.

Liu, D., Zou, Z., Liang, X., 2025. A multi-strategy iterated greedy algorithm with three-phase for hybrid flow shop scheduling in distributed factory. *Applied Soft Computing* 181, 113475.

López-Ramos, F., Nasini, S., Guarnaschelli, A., 2019. Road network pricing and design for ordinary and hazmat vehicles: Integrated model and specialized local search. *Computers and Operations Research* 109, 170–187.

Mendoza-Gómez, R., Ríos-Mercado, R.Z., 2024. Regionalization of primary health care units: an iterated greedy algorithm for large-scale instances. *Expert Systems with Applications* 250, 123924.

Missaoui, A., Ozturk, C., O'Sullivan, B., 2023. Iterated greedy algorithms for combinatorial optimization: a systematic literature review. In *2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*. IEEE, Piscataway, NJ, pp. 1–7.

Mousavia, S., Bhambarb, S., Englandb, M., 2025. An iterated greedy algorithm with variable reconstruction size for the obnoxious p-median problem. *International Transactions in Operational Research* 32, 1, 144–175.

Nucamendi-Guillén, S., Angel-Bello, F., Martínez-Salazar, I., Cordero-Franco, A.E., 2018. The cumulative capacitated vehicle routing problem: new formulations and iterated greedy algorithms. *Expert Systems with Applications* 113, 315–327.

Panin, A.A., Plyasunov, A.V., 2012. On complexity of the bilevel location and pricing problems. *Journal of Applied and Industrial Mathematics* 8, 4, 574–581.

Panin, A.A., Plyasunov, A.V., 2023. The multilevel facility location and pricing problems: the computational complexity and the stability analysis. *Optimization Letters* 17, 1295–1315.

Pisinger, D., Ropke, S., 2019. Large neighborhood search. In Gendreau, M. and Potvin, J.Y. (eds), *Handbook of Metaheuristics*, 3rd edn. Springer, Berlin, pp. 99–127.

Ramalhinho, H., Stützle, T., 2025. Iterated local search, iterated greedy and applications. *TOP* 33, 229–261.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research* 177, 3, 2033–2049.

Rusmevichientong, P., Van Roy, B., Glynn, P.W., 2006. A nonparametric approach to multiproduct pricing. *Operations Research* 54, 1, 82–98.

Sinha, A., Malo, P., Deb, K., 2018. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation* 22, 2, 276–295.

Stützle, T., Ruiz, R., 2018. Iterated greedy. In Martí, R., Pardalos, P.M. and Resende, M.G.C. (eds), *Handbook of Heuristics*. Springer, Berlin, pp. 547–577.

Vasilyev, I.L., Klimentova, K.B., 2010. The branch and cut method for the facility location problem with client's preferences. *Journal of Applied and Industrial Mathematics* 4, 441–454.

Vasilyev, I.L., Klimentova, K.B., Kochetov, Y.A., 2009. New lower bounds for the facility location problem with client's preferences. *Computational Mathematics and Mathematical Physics* 49, 1010–1020.

Wang, H., Ruiz, R., Villa, F., Vallada, E., 2025. Iterated greedy for the yard crane scheduling problem with input/output assignment. *European Journal of Operational Research* 327, 1, 84–94.

Wang, X., Wang, S., Wang, L., Zheng, H., Hao, J., He, R., Sun, Z., 2020. An effective iterated greedy algorithm for online route planning problem. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, Piscataway, NJ, pp. 1–8.

Appendix

This appendix includes the best UL objective function value ever obtained for each of the 240 instances tested. Table A.1 provides for each instance the number of the instance and the best objective function value achieved, Z.

Table A.1
Best available UL objective function value (in bold when optimality is ensured)

Instance	Z	Instance	Z	Instance	Z	Instance	Z	Instance	Z
1	**1715**	49	**2176**	97	**8561**	145	17882	193	31541
2	**1788**	50	**2682**	98	**8552**	146	**18452**	194	33744
3	**1645**	51	**2266**	99	**9487**	147	**16767**	195	32837
4	**2588**	52	**3611**	100	**15046**	148	30885	196	54179
5	**2778**	53	**3551**	101	**14274**	149	31654	197	53411
6	**2579**	54	**3568**	102	**14866**	150	**30864**	198	54751
7	**1702**	55	**2235**	103	**9257**	151	**19743**	199	33775
8	**1550**	56	**2299**	104	**10207**	152	**21114**	200	33623
9	**2165**	57	**2340**	105	**9544**	153	**21035**	201	32655
10	**3037**	58	**3723**	106	**14393**	154	**33926**	202	56119
11	**3105**	59	**3528**	107	**14786**	155	**32141**	203	54890
12	**2998**	60	**3687**	108	**14692**	156	**32854**	204	54593
13	**1568**	61	**7717**	109	**10200**	157	**21797**	205	33694
14	**1619**	62	**6470**	110	**10110**	158	**20231**	206	31970
15	**1833**	63	**6272**	111	**9983**	159	**19939**	207	32078
16	**3068**	64	**12550**	112	14956	160	32581	208	55329
17	**3119**	65	**11905**	113	**14516**	161	33427	209	55064
18	**3132**	66	**12014**	114	**14901**	162	32388	210	54401
19	**2229**	67	**7562**	115	**9969**	163	21048	211	**36004**
20	**1996**	68	**7773**	116	**9909**	164	21825	212	**40849**
21	**1635**	69	**7681**	117	9248	165	21052	213	**38463**
22	**3177**	70	**12855**	118	14661	166	33192	214	58676
23	**3253**	71	**12707**	119	15082	167	33215	215	58050
24	**3198**	72	**12968**	120	14428	168	33479	216	58757
25	**1739**	73	**7591**	121	**15688**	169	20871	217	36821
26	**2012**	74	**8422**	122	**17015**	170	20975	218	38245

Continued

Table A.1
(Continued)

Instance	Z	Instance	Z	Instance	Z	Instance	Z	Instance	Z
27	**1929**	75	**7897**	123	**16014**	171	20718	219	39422
28	**3237**	76	**12860**	124	**28383**	172	32695	220	59134
29	**3177**	77	**13070**	125	**28037**	173	33057	221	59702
30	**3129**	78	**12740**	126	**28704**	174	32943	222	59793
31	**2131**	79	**7565**	127	**18777**	175	20215	223	38028
32	**2467**	80	**7741**	128	**16107**	176	21686	224	40721
33	**1956**	81	**7621**	129	**16600**	177	19260	225	38785
34	**3663**	82	**13559**	130	**29367**	178	32829	226	59846
35	**3634**	83	**13530**	131	**29285**	179	33663	227	60008
36	**3526**	84	**13420**	132	**29556**	180	32704	228	58704
37	**2362**	85	**7666**	133	**18555**	181	**28276**	229	37183
38	**2219**	86	**8037**	134	**17540**	182	**29084**	230	42982
39	**2272**	87	**8067**	135	**16963**	183	**28616**	231	38274
40	**3466**	88	**13304**	136	**29707**	184	**51077**	232	59070
41	**3531**	89	**13485**	137	**30771**	185	**50555**	233	59026
42	**3595**	90	**13630**	138	**30543**	186	**49957**	234	59590
43	**2386**	91	**9616**	139	**18562**	187	**30305**	235	37897
44	**2327**	92	**9196**	140	17452	188	31887	236	40755
45	**2467**	93	**8503**	141	**18790**	189	**27845**	237	38844
46	**3685**	94	**14827**	142	**30618**	190	52529	238	58863
47	**3766**	95	**14492**	143	**30944**	191	52914	239	59793
48	**3683**	96	**14476**	144	**29841**	192	53750	240	59275