European Journal of Operational Research 326 (2025) 439–450

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/eor

Discrete Optimization

Scatter search with path relinking for linear bilevel problems[☆]

Herminia I. Calvete[a,*], Carmen Galé[b], José A. Iranzo[c], Manuel Laguna[d]

[a] Departamento de Métodos Estadísticos, Instituto Universitario de Matemáticas y Aplicaciones (IUMA), Universidad de Zaragoza, Pedro Cerbuna 12, 50009 Zaragoza, Spain
[b] Departamento de Métodos Estadísticos, Instituto Universitario de Matemáticas y Aplicaciones (IUMA), Universidad de Zaragoza, María de Luna 3, 50018 Zaragoza, Spain
[c] Departamento de Métodos Estadísticos, Instituto Universitario de Matemáticas y Aplicaciones (IUMA), Universidad de Zaragoza, Violante de Hungría 23, 50009 Zaragoza, Spain
[d] Leeds School of Business, University of Colorado Boulder, Boulder, CO 80308-0419, USA

ARTICLE INFO

MSC:
90C26
90C30

Keywords:
Bilevel optimization
Linear bilevel problem
Scatter search
Path relinking
Extreme point

ABSTRACT

The literature includes very few instances of scatter search applications to bilevel optimization. These implementations have been proposed for problems in the field of logistics involving integer variables and are based on a structure where scatter search sets the values of the decisions at the upper level followed by the solution of the lower level problem. In this work, we develop a scatter search for solving linear bilevel problems. Our proposal employs a tailored path relinking procedure that generates solutions that are boundary feasible extreme points located in the trajectory between infeasible and feasible bilevel solutions. We perform scientific experimentation to determine the most effective configuration of our scatter search with path relinking. We also perform competitive experiments to determine where the proposed solution method stands when compared to the state of the art for tackling linear bilevel problems.

1. Introduction

Bilevel programming (Bard, 1998; Dempe, 2002) addresses problems where there are two levels of interdependent decision-making within a hierarchical structure in which each level of the hierarchy controls a subset of decision variables. The decision maker at the upper level (UL) of the hierarchy, known as the leader, strives to optimize his/her objective function while considering a set of constraints that account for the response of the decision maker at the lower level (LL) of the hierarchy, known as the follower, to the leader's plan. Consequently, bilevel programs are formulated as optimization problems that encompass another optimization problem within the constraint set. Bilevel programs are non-convex by nature, making them particularly difficult to manage and solve. Even when all the functions involved are linear, bilevel problems remain (strongly) NP-hard (Hansen et al., 1992). To deal with these problems, a variety of exact algorithmic approaches have been developed. These methods involve enumerative schemes or are based on the replacement of the LL problem by its Karush-Kuhn–Tucker conditions followed by the application of penalty function or gradient methods, among others. For a more detailed discussion of these approaches, the reader is referred to the work by Dempe and Zemkoho (2020). Metaheuristic methods have also been employed to tackle bilevel programming problems. Camacho-Vallejo et al. (2024) provide a comprehensive overview of the state-of-the-art of applying metaheuristic approaches to bilevel optimization.

Our work focuses on the Linear Bilevel Problem (LBP), a special case of Bilevel Problems where both the objective functions and the constraints are linear and the variables are continuous, formulated as:

$$\max_{x_1,x_2} \quad c_1x_1 + c_2x_2 \tag{1a}$$

subject to:

$$x_1 \geqslant 0_{n_1} \tag{1b}$$

where x_2 solves

$$\max_{x_2} \quad d_1x_1 + d_2x_2 \tag{1c}$$

subject to:

$$A_1x_1 + A_2x_2 \leqslant b \tag{1d}$$

$$x_2 \geqslant 0_{n_2} \tag{1e}$$

[☆] This research has been funded by the Spanish Ministry of Economy, Industry and Competitiveness under grant PID2022-139543OB-C43 (AEI/10.13039/501100011033) and by the Gobierno de Aragón, Spain under grants E41-20R and E41-23R.

[*] Corresponding author.
E-mail addresses: herminia@unizar.es (H.I. Calvete), cgale@unizar.es (C. Galé), joseani@unizar.es (J.A. Iranzo), laguna@colorado.edu (M. Laguna).

https://doi.org/10.1016/j.ejor.2025.04.043
Received 19 June 2024; Accepted 23 April 2025
Available online 8 May 2025

where $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$ denote the sets of variables controlled by the UL decision maker and the LL decision maker, respectively c_1 and d_1 are n_1-dimensional row vectors, c_2 and d_2 are n_2-dimensional row vectors, A_1 is an $m \times n_1$-matrix, A_2 is an $m \times n_2$-matrix, b is an m-dimensional column vector and 0_{n_1}, 0_{n_2} are n_1 and n_2-dimensional column vectors consisting of zeros. As the UL decision maker sets the values of the variables x_1, the first term in the LL objective function remains constant, hence it can be removed from the problem formulation. We adopt the optimistic approach for tackling LBPs, which assumes that the UL objective function maximizes over x_1 and x_2. Furthermore, we adhere to the prevalent formulation found in literature, in which the UL constraints (1b) exclusively involve UL variables. Alternative methods for addressing LBPs or the consequences of having coupling constraints, i.e., UL constraints involving both UL and LL variables, are discussed in Audet et al. (2006), Mersha and Dempe (2006), Dempe and Zemkoho (2020) and Henke et al. (2025).

Calvete and Galé (2020) provide an in-depth analysis of both the properties and algorithms proposed in the literature to address LBPs, where it is evident that no Scatter Search (SS) applications to these problems exist. In fact, the review by Camacho-Vallejo et al. (2024) identifies SS as one among the least explored population-based metaheuristics for bilevel optimization, thus suggesting an interesting research direction. SS is a metaheuristic designed to find high-quality solutions, usually, to combinatorial optimization problems. SS maintains a diverse pool of candidate solutions called the reference set. These solutions represent different regions of the search space and are used to generate new high-quality solutions through combinations and improvements. SS employs techniques such as path relinking to generate new solutions by combining elements from existing solutions in the reference set. This enables the exploration of new areas of the search space while maintaining and improving solution quality.

We propose the first SS procedure to tackle LBPs. The approach leverages two key properties of the LBP. First, it exploits the existence of an extreme point of the region defined by the constraints (1b), (1d) and (1e) that solves the problem. Second, for non-trivial instances (see Section 3), this optimal extreme point has at least one adjacent extreme point that is not a feasible solution to the LBP. Therefore, the proposed SS method explores the space of extreme points of the constraint region, rather than focusing solely on feasible solutions of the LBP. That is, it searches for better solutions by moving along the boundary rather than through the interior of the constraint region. Moreover, the reference set contains some extreme points that are feasible solutions to the LBP and some that are not. Then, building on the idea of path relinking, the algorithm searches for trajectories from a bilevel feasible extreme point to an infeasible one (or vice versa), with the goal of finding a feasible extreme point with at least one adjacent infeasible extreme point that has the potential to be an optimal solution. When that point is found, a local search step is applied by exploring adjacent extreme points with the goal of improving the UL objective function.

The main contributions of this paper are the following:

- We propose the first SS approach for solving LBPs. This algorithm moves along the boundary of the constraint region of the LBP, as it only considers extreme point solutions.
- We propose a novel path relinking procedure based on the properties of LBPs. The reference set consists of feasible and infeasible extreme point solutions. The path relinking process moves along adjacent extreme points, starting from a feasible solution to the LBP and ending at an infeasible one (or vice versa), until an extreme point is reached that could potentially lead to an optimal solution. This strategy, sometimes referred to as tunneling in the path relinking literature (Martí et al., 2006), benefits from exploring infeasible regions and, in our case, enables the identification of solutions on the boundary of feasible regions, which are promising candidates for the optimal solution of the problem. During this process, the successive adjacent extreme points are obtained through simplex iterations when solving an ad hoc linear programming problem.
- We propose an improvement method which consists of examining adjacent extreme points of the potential optimal solution provided by the path relinking process, in an order which depends on the values of the marginal costs based on the UL objective function.
- We perform extensive computational experiments on benchmark instances that show that the algorithm provides high-quality solutions in short computational times.

The remainder of this paper is organized as follows. After the Introduction section, Section 2 presents a brief literature review on SS in bilevel problems. Then, Section 3 outlines the fundamental properties of the LBP used in the development of the algorithm that we propose. In Section 4 a detailed description of the algorithm is presented with emphasis on those elements, such as the choice of the solutions involved in the path relinking process, that make the procedure an innovative implementation of a SS. Sections 5 and 6 presents the results of the extensive computational experiments carried out. The scientific experimentation focuses on identifying the most effective configuration by testing various settings of the main components of the algorithm. The competitive experimentation focuses on assessing the efficiency of the chosen configuration by comparing its performance provided with the best results available in the literature. We provide empirical evidence that the proposed algorithm is competitive especially for large problems. Finally, in Section 7 we present some concluding remarks and directions for future work.

2. Literature review

SS (Glover et al., 2000; Laguna, 2014; Resende et al., 2010) is an evolutionary metaheuristic designed to find high-quality solutions, usually, to combinatorial optimization problems. It works by iteratively generating and improving a diverse set of candidate solutions, combining them in a structured manner to explore the solution space effectively. The core idea involves maintaining a reference set of solutions, which is refined using a combination of diversification and intensification strategies, balancing exploration and exploitation throughout the process. The reference set is updated by combining solutions and producing new candidates that are evaluated and added back to the set if they improve the solution quality. A key feature of SS is path relinking, a mechanism used to enhance the search process. Path relinking is employed to explore trajectories between elite solutions by creating paths that connect them. This process generates new solutions by moving from one solution to another while attempting to improve along the way. By exploring different paths between promising solutions, path relinking helps in intensifying the search around high-quality regions of the solution space, thus enhancing the algorithm's ability to escape local optima and improve overall performance.

Kalra et al. (2021) provide a comprehensive review of SS published work. This review discusses the effectiveness of SS as a solution method in application areas ranging from manufacturing, data mining, healthcare, and various fields of engineering. SS applications include multiobjective optimization, where the methodology has been successfully interpreted to explore solution and objective function spaces to produce near-optimal Pareto fronts (Molina et al., 2007). Scatter search's ability to balance exploration and exploitation, maintain a diverse set of solutions, leverage the storage of high-quality solutions, combine solutions effectively, adapt to different problem domains, and utilize parallelization makes it an effective metaheuristic methodology for solving complex optimization problems.

As mentioned above, the work by Dempe and Zemkoho (2020) and Camacho-Vallejo et al. (2024) provides an updated and extensive review of methods developed to solve bilevel problems. Hence, we focus on the few articles in the literature related to the application of SS to bilevel optimization problems. SS has been used to tackle bilevel problems that arise in supply chains, transportation networks, and electric distribution systems. In all of these applications, SS explores the solution space of the UL decision maker, i.e. it determines the value of the UL decision variables, while the LL decision variables' values are determined by optimally solving the corresponding LL problem. Camacho-Vallejo et al. (2015) propose a heuristic algorithm based on SS to address a supply chain's production-distribution problem. Supported by successful vehicle routing applications of SS, they employ it to develop distribution plans. Following these decisions, the LL production problem is solved to optimality in order to assess the merit of the distribution plan. Similarly, González Velarde et al. (2015) use SS to solve the bilevel problem formulated for determining highway tolls in a transportation network. Once the SS determines tolls, the LL problem is solved to optimize the flows in the arcs of a transportation network. Casas-Ramírez and Camacho-Vallejo (2015) present a variant of the p-median problem in which customers can freely select from open facilities. The resulting bilevel problem is solved by a hybrid heuristic algorithm which includes SS to explore the leader's decision space, that is, all possible ways to open p facilities. Finally, Pérez Posada et al. (2017) develop a SS heuristic for the optimal location, sizing and pricing of distributed generation in electricity distribution systems. The pricing decisions are taken at the UL of the hierarchy. Once the distributed generation owner has set these values, the distribution company at the LL decides the best option for purchasing energy.

The literature includes some work on path relinking applied to bilevel optimization problems. Maldonado-Pinto et al. (2016) address the uncapacitated facility location problem with customer preferences, formulating it as a bilevel problem. The proposed algorithm combines an evolutionary algorithm and path relinking, integrating path relinking into the crossover phase of the genetic algorithm to explore connections between two parents. Ahuja and Orlin (1997) illustrate that the use of path relinking as a solution recombination method can be a key feature of genetic algorithms. Path relinking offers a significant advantage over other crossover operators by generating multiple solutions very efficiently. Andrade et al. (2021) develop a biased random key genetic algorithm with an implicit path relinking procedure. While traditional path relinking implementations exploit the problem structure by directly operating in the solution neighborhood, Andrade et al. (2021) rely solely on the implicit structure of the random-key vector within their procedure. Polino et al. (2024) propose two metaheuristics to solve large-scale instances of a location problem for extracurricular workshop planning, which is formulated as a bilevel model to account for student preferences. In one of the metaheuristics, a path relinking operator is proposed to intensify the search by exploring the path connecting two previously found feasible solutions.

3. Background

To make the paper self-contained, this section presents the fundamental concepts of linear bilevel optimization used in our study. Let S be the polyhedron defined by the constraints (1b), (1d) and (1e). We assume that S is non-empty and bounded. Let S_1 be the projection of S on $\mathbb{R}^{n_1}$. For each $x_1 \in S_1$, the LL decision maker solves the following linear programming problem, called LL problem:

$$\max_{x_2} \quad d_2 x_2 \tag{2a}$$

subject to:

$$A_2 x_2 \leqslant b - A_1 x_1 \tag{2b}$$

$$x_2 \geqslant 0_{n_2} \tag{2c}$$

Denote by $M(x_1)$ the set of optimal solutions to (2). Assuming that $M(x_1) \neq \emptyset$, the feasible region of the UL decision maker, referred to as inducible region (IR), is defined as:

$$\text{IR} = \{(x_1, x_2) \;:\; x_1 \in S_1, \quad x_2 \in M(x_1)\}.$$

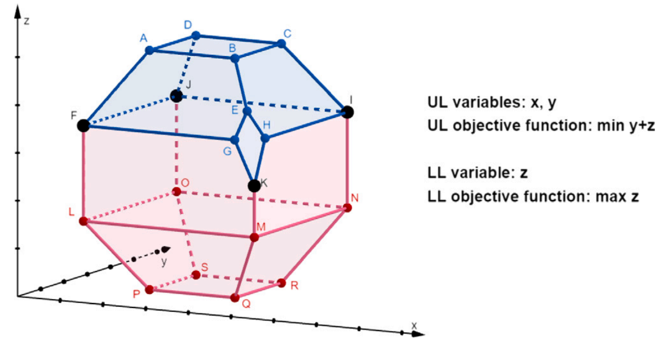

Fig. 1. Polyhedron S of a LBP. The UL objective function is to minimize $y + z$. The LL objective function is to maximize z. IR is colored in blue. The optimal solution is the boundary feasible extreme point K. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Any point $(x_1, x_2) \in$ IR represents a feasible solution of problem (1). IR is formed by the union of connected faces of the polyhedron S. Therefore, IR is typically a nonconvex set, which makes LBP difficult to solve. However, there exists an extreme point of IR, thus an extreme point of the polyhedron S, that is an optimal solution to the LBP (Bard, 1998; Bialas & Karwan, 1982; Dempe, 2002; Savard, 1989).

Let us consider the relaxed linear optimization problem (3), also called high point problem (Bard, 1998), which omits the LL objective function in problem (1):

$$\max_{x_1, x_2} \quad c_1 x_1 + c_2 x_2 \tag{3a}$$

subject to:

$$A_1 x_1 + A_2 x_2 \leqslant b \tag{3b}$$

$$x_1 \geq 0_{n_1}, \; x_2 \geq 0_{n_2} \tag{3c}$$

Clearly, if an optimal solution of the relaxed problem (3), belongs to IR, then it is an optimal solution of the LBP and thus the corresponding LBP can be considered 'trivial' in the sense that the relaxed problem is not difficult to solve. This paper focuses on non-trivial LBPs for which an algorithm can be developed to search for the best extreme point in S according to the UL objective function that also belongs to IR. The SS with path relinking approach developed in this paper relies on two primary properties of the LBP. The first aforementioned property is that there is a extreme point of the polyhedron S that solves problem (1). The second property, and most crucial for path relinking, is that if there exists an extreme point of S not in IR that is an optimal solution of the relaxed problem, then there exists a boundary feasible extreme point that solves the LBP (Liu & Hart, 1994). A point $(x_1, x_2) \in$ IR qualifies as a boundary feasible extreme point if there exists an edge E of S such that (x_1, x_2) is an extreme point of E, and the other extreme point of E is not an element of IR. Hence, the algorithm constructs extreme points in IR, henceforth referred to as bilevel basic feasible (BF) solutions, and extreme points in S that are not in IR, henceforth referred to as non-bilevel basic feasible (NBF) solutions. Then, for the purpose of identifying boundary feasible extreme points, BF solutions and NBF solutions are connected by paths that are certain to pass through boundary feasible extreme points.

Fig. 1 illustrates these properties. The figure depicts a polyhedron S associated with an LBP, where x and y represent the variables controlled by the UL decision maker, and z refers to the variable controlled by the LL decision maker. The UL objective function aims to minimize $y + z$, while the LL objective function aims to maximize z. IR is highlighted in blue. The boundary feasible extreme points, drawn with large black dots in the figure, are F, I, and J and K. Among them, the optimal solution corresponds to the extreme point K.

Next section describes in detail how the Bilevel Scatter Search (BSS) algorithm developed in this paper operates.

4. SS with path relinking for LBPs: The BSS algorithm

As mentioned above, the procedure presented in this paper, bilevel scatter search (BSS), exploits the optimality properties of LBPs. This is done by searching for boundary feasible extreme points while traversing paths from BF solutions to NBF solutions and vice versa. Note that, since the procedure is designed to search in the space of extreme points, we refer to extreme points and solutions interchangeably.

BSS starts by generating a set P of distinct solutions of the LBP, half BF solutions and half NBF solutions. Let P_{BF} and P_{NBF} the corresponding sets of solutions. Next, a reference set ($RefSet$) is built from P containing solutions of both types, half of each type. The algorithm proceeds by considering pairs of solutions within $RefSet$ consisting of a BF solution and an NBF solution. The combination of each pair of solutions by applying a path relinking approach produces a new BF solution that is a boundary feasible extreme point of the polyhedron S. These BF solutions are improved via local search. Finally, the $RefSet$ is updated by maintaining the best BF solution according to the UL objective function obtained so far and adding unused solutions from P. This process is repeated over $maxIter$ iterations. Upon completion, the algorithm provides the best BF solution, X^{best}. Quality comparisons among solutions are made with respect to the UL objective function.

Algorithm 1 shows the BSS pseudocode, whose main elements are described in the following subsections: the diversification generation method in Section 4.1, the reference set update method in Section 4.2, the subset generation method in Section 4.3, the solution combination method in Section 4.4 and the improvement method in Section 4.5. In addition, taking into account the computational cost involved in checking the bilevel feasibility of a solution, Section 4.6 explains how these calculations are addressed.

Algorithm 1: Pseudocode of the BSS algorithm

```
1  Let $X^{rel}$ be an optimal solution of the relaxed problem;
2  if $X^{rel}$ is a BF solution then
3      return $X^{rel}$;            /* optimal solution found */
4  end
5  $P_{BF} = \emptyset$;
6  $P_{NBF} = \{X^{rel}\}$;
7  Use the diversification generation method to build $P_{BF}$ and
     $P_{NBF}$;
8  Let $X^{best}$ be a solution with the greatest UL objective function
     value in $P_{BF}$;
9  for $i = 1, \dots, maxIter$ do
10     Use the reference set update method to build or update
         $RefSet$;
11     Generate $NewSubsets$ by applying the subset generation
         method;
12     for $s \in NewSubsets$ do
13         Apply the solution combination method to obtain a new
             trial solution $X$ from subset $s$;
14         Apply the improvement method to $X$;
15         Update the solution $X^{best}$ if $X$ improves it.
16     end
17 end
18 return $X^{best}$
```

To simplify notation, given a extreme point X of the polyhedron S, we will denote by $B(X)$ the set of indices of its basic variables. Moreover, from now on, we rewrite the polyhedron S as:

$$S = \{(x_1, x_2, u) \in \mathbb{R}^n : A_1x_1 + A_2x_2 + u = b,\ x_1 \geq 0_{n_1},\ x_2 \geq 0_{n_2},\ u \geq 0_m\}$$

where $u \in \mathbb{R}^m$ denote the m-dimensional column vector of slack variables, which are controlled by the LL decision maker, and 0_m is an m-dimensional column vector of zeros.

4.1. Diversification generation method (DGM)

DGM generates a set P of $PSize$ distinct solutions. This set is divided into two subsets: P_{BF} contains $PSize/2$ BF solutions and P_{NBF} has $PSize/2$ NBF solutions. To construct these sets, DGM computes solutions in S, whose bilevel feasibility is checked (see Section 4.6 for details). Then, depending on whether the points are in IR or not, they are added to P_{BF} or P_{NBF}, respectively, discarding duplicate solutions.

Since we are considering non-trivial problems, the first solution in P_{NBF} is an optimal solution of the relaxed problem (3), X^{rel}. The first solution in P_{BF} is X^*, an optimal solution of the problem:

$$\begin{aligned} \max_{x_1,x_2,u} \quad & c_1x_1 + d_2x_2 \\ \text{subject to:} \quad & \\ & (x_1, x_2, u) \in S \end{aligned}$$

Note that the objective function of this problem partially takes into account the leader's preferences while ensuring a BF solution due to the coefficient d_2 multiplying x_2.

In order to diversify the initial population of solutions and to encourage the exploration of the entire polyhedron S, the UL objective function is maximized over the maximal dimension hyperplanes of the polyhedron S to which X^{rel} does not belong. For this purpose, the following m linear problems are solved:

$$\begin{aligned} P_1^j: \quad \max_{x_1,x_2,u} \quad & c_1x_1 + c_2x_2 \\ \text{subject to:} \quad & \\ & A_1x_1 + A_2x_2 + u = b \\ & X_j = 0 \\ & x_1 \geq 0_{n_1},\ x_2 \geq 0_{n_2},\ u \geq 0_m \end{aligned}$$

where X_j denotes the jth component of $X = (x_1, x_2, u)$ and $j \in B(X^{rel})$. After solving each P_1^j, the bilevel feasibility of the corresponding solution is evaluated and the solution is added to P_{BF} or P_{NBF}, as appropriate. In general, most solutions are NBF solutions. Therefore, to promote feasibility, the following m linear problems related to X^* are solved:

$$\begin{aligned} P_2^j: \quad \max_{x_1,x_2,u} \quad & c_1x_1 + d_2x_2 \\ \text{subject to:} \quad & \\ & A_1x_1 + A_2x_2 + u = b \\ & X_j = 0 \\ & x_1 \geq 0_{n_1},\ x_2 \geq 0_{n_2},\ u \geq 0_m \end{aligned}$$

where X_j denotes the jth component of $X = (x_1, x_2, u)$ and $j \in B(X^*)$. Note that the solutions of these problems are BF solutions, thus they are added to P_{BF}.

After updating P with the solutions provided by the problems above and only in the case that there are no $PSize/2$ solutions in each subset P_{NBF} and P_{BF}, to encourage diversity, we propose to randomize the generation of initial solutions. To this end, as many times as necessary to reach $PSize/2$ NBF distinct solutions, the following problem is solved:

$$\begin{aligned} P_3: \quad \max_{x_1,x_2,u} \quad & Rnd_1x_1 + Rnd_2x_2 \\ \text{subject to:} \quad & \\ & (x_1, x_2, u) \in S \end{aligned}$$

where $Rnd_1 \in \mathbb{R}^{n_1}$ and $Rnd_2 \in \mathbb{R}^{n_2}$ are randomly generated coefficients within the uniform interval $(-C, C)$, where $C \geqslant 0$ is a constant. The bilevel feasibility of the solutions is checked and solutions are accordingly added to P_{NBF} or P_{BF} (while avoiding duplicates). If at the time the set P_{NBF} is completed, the set P_{BF} has not yet been completed, the following problem is solved as many times as necessary to complete it:

$$P_4: \quad \max_{x_1,x_2,u} \quad Rnd_1 x_1 + d_2 x_2$$
$$\text{subject to:}$$
$$(x_1, x_2, u) \in S$$

where $Rnd_1 \in \mathbb{R}^{n_1}$ is generated as indicated above. From pilot testing, we concluded that $C = 5\|d_2\|_\infty$ resulted in an effective balance between the elements of the objective function to produce diverse solutions.

4.2. Reference set update method (RSUM)

The $RefSet$ consists of $RSize$ solutions, where half are BF solutions and half are NBF solutions, seeking a balance between high quality solutions and diverse solutions. RSUM selects solutions from P to be added to $RefSet$. Selected solutions are removed from P to avoid considering them again in successive iterations. The value of $RSize$ is one-tenth of the value of $PSize$. Typical values for these parameters in the SS literature are $PSize = 100$ and $RSize = 10$.

To build $RefSet$ for the first time the following steps are taken. Steps 1 and 2 look for solution quality, step 3 looks for diversity. To measure how diverse two solutions are, we introduce a distance that measures the number of different basic variables in terms of the index of the basic variable, not its value. Thus, given the solutions X and Y, $dist(X,Y) = m - |B(X) \cap B(Y)|$, where $|B(X) \cap B(Y)|$ provides the cardinality of set $B(X) \cap B(Y)$, i.e. the number of common basic variables.

Steps:

1. Select the best $\lfloor RSize/4 \rfloor$ solutions from set P_{BF} and add them to $RefSet$.
2. Select the best $\lfloor RSize/4 \rfloor$ solutions from set P_{NBF} and add them to $RefSet$.
3. Successively, repeat steps (a) and (b) until $RefSet$ is completed.
   (a) Select a solution in P_{BF} that maximizes the minimum distance to the solutions in $RefSet$. Add it to $RefSet$.
   (b) Select a solution in P_{NBF} that maximizes the minimum distance to the solutions in $RefSet$. Add it to $RefSet$.

In typical SS implementations, the reference set is built at the beginning of the search and then it is updated with solutions that result from applying the combination method followed by the improvement method if any. However, as will be explained below, the combination and the improvement methods in the BSS algorithm only provide BF solutions, making it impossible to replace the NFB solutions and, in the short term, making it impossible to obtain increasingly better BF solutions. Hence, the $RefSet$ is almost completely renewed in each successive iteration of the BSS algorithm. Only the best BF solution so far is maintained, and the steps described above are applied to select solutions from those remaining in P.

4.3. Subset generation method (SGM)

From $RefSet$, SGM creates subsets of two solutions: a BF solution and an NBF solution. Three variants of SGM have been considered whose performance will be analyzed in the Computational Study section:

- SGM = All: All possible pairs of solutions, one being a BF solution and the other an NBF solution, are treated as subsets. This variant of SGM produces a total of $(RSize/2)^2$ subsets.
- SGM = Ord: A total of $RSize/2$ subsets are created by taking the best BF solution of $RefSet$ and pairing it with the best NBF solution. Then, the second best solutions in $RefSet$ of each type are paired. The process continues until each BF solution is paired with an NBF solution.

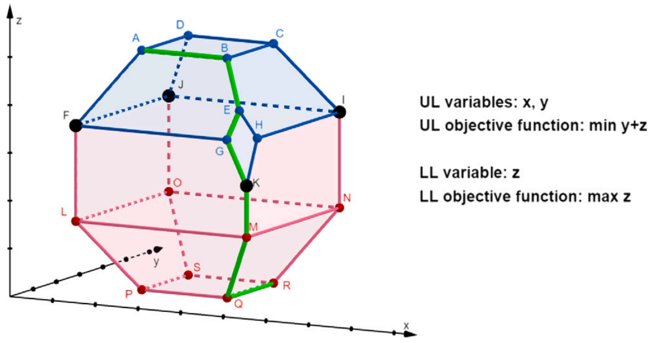

Fig. 2. The green line highlights a path between the BF extreme point A and the NBF extreme point R. The path traverses K, a boundary feasible extreme point. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- SGM = Dis: a total of $RSize/2$ subsets are created. For this purpose, the distance (as defined in Section 4.2) between each BF solution and each NBF solution in $RefSet$ is computed. Then, successively, the pair of solutions which maximizes this distance, is selected to create a subset and the corresponding solutions are discarded, until $RSize/2$ subsets are created.

4.4. Solution combination method (SCM)

SCM is applied to every subset created by SGM aiming to obtain new trial solutions by combining each pair of solutions. Let X and Y be the solutions to be combined. Our combination strategy consists of building a path from X to Y through adjacent extreme points of the polyhedron S. Since one of the solutions (X or Y) is BF and the other (Y or X) is NBF, any path constructed in this way contains at least two consecutive extreme points where one is BF and the other is NBF, that is, it contains at least one boundary feasible extreme point. Therefore, boundary feasible extreme points, where optimal solutions of the bilevel problem lie, are the result of the path relinking process. Going back to the example presented in Fig. 1, Fig. 2 shows a path connecting extreme point A (a BF solution) and extreme point R (an NBF solution), that contains the boundary feasible extreme point K.

BSS performs the path relinking process through simplex iterations to move from one extreme point of the polyhedron S to an adjacent one in order to transform the initiating solution X into the guiding solution Y. For this purpose, taking into account that all non-basic variables are zero due to the nonnegativity constraints defining S, we solve the linear problem that minimizes the sum of the non basic variables in the guiding solution Y over the polyhedron S. Starting from the initiating solution X the application of the simplex algorithm to this problem would produce a path of extreme points to the guiding solution Y. After each iteration of the simplex algorithm, the extreme point obtained is checked for bilevel feasibility and the procedure is stopped as soon as a boundary feasible extreme point is detected, that is, either when transitioning from a BF solution to an NBF solution or vice versa. The solution returned as the output of the SCM is the last BF solution visited when the simplex algorithm stops.

Two variants of the SCM have been considered, which depend on the type of starting solution and whose performance will be studied in the computational study section:

- SCM = BF: The combination is performed by starting the path at the BF solution towards the NBF solution.
- SCM = NBF: The combination is performed by starting the path at the NBF solution towards the BF solution.

To reduce the number of calculations performed when applying SCM, the procedure checks whether the current path or subpath has already been examined. In such a case, SCM stops and does not return a solution. A record of previously visited paths and subpaths is kept for this purpose. To achieve this efficiently, each solution is assigned a hash code using the C++ standard library tools. This hash code is calculated from the binary vector indicating whether each variable is basic or not. The hash codes associated with each pair of solutions to which the combination method has been applied is then stored, along with the hash codes of the intermediate solutions of the path found. This information is used to avoid duplicating analyses already performed.

4.5. Improvement method (IM)

IM aims to find a BF solution of higher quality, i.e. with a better value of the UL objective function, starting from a BF solution provided by the SCM, i.e. a boundary feasible extreme point. Let X be this solution. The application of IM to X involves examining adjacent extreme points in S that improve the leader's objective function. The procedure moves from the incumbent solution to an adjacent one only if the adjacent solution is a BF solution.

The order in which adjacent solutions are examined depends on the values of the marginal costs (calculated based on the leader's objective function) of the non-basic variables of the incumbent solution. Adjacent solutions with basic variables that are non-basic in the incumbent solution and have the highest positive marginal cost are examined first. If none of the adjacent solutions that improve the current solution is a BF solution, IM stops. Otherwise, the search moves to the adjacent solution and the above process is repeated from the new incumbent solution. Note that, at termination, IM provides a boundary feasible extreme point.

Taking into account that IM is a deterministic process, for computational efficiency, IM stops when it finds a solution to which it has already been applied. As in SCM, the hash codes of the solutions to which the improvement method has been applied are stored, which prevents IM from being applied to solutions that have already been explored.

4.6. Evaluation of bilevel feasibility

To check whether the extreme point $X = (x_1, x_2, u) \in S$ is a bilevel feasible solution, we have to check whether (x_2, u) is an optimal solution to problem (2). According to the developments in Calvete et al. (2008), this reduces to checking whether the following linear system has a solution:

$$\begin{aligned} & wA_2 - v = d_2 \\ & w^t \geqslant 0_m,\ v^t \geqslant 0_{n_2} \\ & w^j = 0,\ j \in J,\ v^k = 0,\ k \in K \end{aligned} \tag{4}$$

where w is an m-dimensional row vector of dual variables, v is an n_2-dimensional row vector of slack variables, w^j and v^k refer to the jth and kth components of vectors w and v, respectively, J is the index set of variables complementary (according to duality theory) to variables of u which are basic variables, K is the index set of variables complementary to basic variables of x_2, and ^{t} refers to transpose.

Calvete et al. (2008) suggest to solve problem (4) by applying the first phase of the two-phase method of linear programming. However, working with artificial variables involves an extra cost, both in the amount of computations to be performed and in the accumulation of pivots by application of the revised simplex algorithm. Therefore, we propose to initially solve problem (5) until all the artificial variables are non-basic variables:

$$\begin{aligned} \min_{w,v,a} \quad & \sum_{i=1}^{n_2} a_i \\ \text{subject to:} \quad & \\ & wA_2 - v + a = d_2 \\ & w^t \geqslant 0_m,\ v^t \geqslant 0_{n_2},\ a^t \geqslant 0_{n_2} \end{aligned} \tag{5}$$

where a is the n_2-dimensional row vector of artificial variables and a_i refers to its ith component. Let (w^*, v^*) be an optimal solution of problem (5). Then, (w^*, v^*), already without artificial variables and without accumulated pivots, is used as the starting solution in the application of the revised simplex algorithm thereafter to problem (6):

$$\begin{aligned} \min_{w,v} \quad & \sum_{j\in J} w_j + \sum_{k\in K} v_k \\ \text{subject to:} \quad & \\ & wA_2 - v = d_2 \\ & w^t \geqslant 0_m,\ v^t \geqslant 0_{n_2} \end{aligned} \tag{6}$$

Note that if, at termination, the optimal objective function value of problem (6) is zero, X is a BF solution; otherwise, it is an NBF solution.

In addition, it is worth pointing out that BSS needs to check the bilevel feasibility of a large number of solutions in the application of DGM, SCM and IM. Therefore, given the computational cost that this requires, in order not to repeat unnecessary calculations, a history of the solutions whose bilevel feasibility has been previously checked is kept during the running of the algorithm. The following steps only apply if the solution under consideration has not been analyzed previously.

4.6.1. Bilevel feasibility checking in DGM

To check the bilevel feasibility of X, a solution provided by DGM, starting from (w^*, v^*), problem (6) is solved.

4.6.2. Bilevel feasibility checking in SCM

SCM builds a path of adjacent solutions. Therefore, we apply Lemma 1 and Theorems 2, 4 and 5 in Calvete et al. (2008), when possible, to identify if the incumbent solution is BF or NBF without having to solve an additional problem. To make the paper self-contained, the properties applied are described below:

- If the current solution is a BF solution and the adjacent solution in the path has the same LL basic variables, then the adjacent solution is also a BF solution.
- If the current solution is an NBF solution and the adjacent solution in the path has the same LL basic variables, then the adjacent solution is also an NBF solution.
- If the current solution is a BF solution, an UL variable enters the basis and an LL variable leaves the basis, then the adjacent solution is also a BF solution.
- If the current solution is an NBF solution, an LL variable enters the basis and an UL variable leaves the basis, then the adjacent solution is also an NBF solution.

If previous properties cannot be applied, problem (6) needs to be solved. Bearing in mind that the procedure construct adjacent solutions, we can expect that consecutive solutions will be similar to each other in terms of the variables w_j and v_k which should be equal to zero. Hence, the starting feasible solution to solve the current problem (6) is either the optimal solution of the preceding problem (6) solved, if available, or (w^*, v^*) if no previous solution exists. From pilot tests, we have noticed that this reduce significantly the computational time. However, the drawback is that, since the revised simplex algorithm is used to solve the problem, many pivots accumulate in the long run and then their management is time consuming. Experimentally, it has been observed that it is convenient to retake (w^*, v^*) as the starting solution every $3n_2$ iterations.

4.6.3. Bilevel feasibility checking in IM

This case is similar to that of Section 4.6.2. However, the starting solution is the one obtained after stopping SCM. Moreover, in general, several adjacent NBF solutions are examined until a BF solution that is better than the incumbent is found, if any. Since the path is not shifted to NBF solutions, pivots are not update in this case. This avoids the accumulation of unnecessary pivots.

5. Computational study: Selecting the configuration of BSS

This section presents the results of the computational experiments conducted to configure BSS. The numerical experiments were performed on a PC 13th Gen Intel Core i9-13900F at 2.0 GHz × 32 having 64.0 GB of RAM, and Windows 11 64-bit as the operating system. The metaheuristics was coded in C++, MSVC 19.38. We begin this section by outlining the computational setup, which includes a description of the set of instances used for testing and the configurations of BSS to be examined. Next, we analyze the effect of updating the $RefSet$ on the quality of the resulting solutions. Finally, we focus on parameter fine-tuning to identify the most promising algorithm configuration. As usual, performance is assessed by monitoring both solution quality and computational effort.

5.1. Computational setup

This computational experience has been carried out on a collection of 630 LBPs introduced by Calvete et al. (2008). This set contains instances that differ in the number of variables in the upper and lower levels, as well as in the number of constraints. The collection includes three main groups of instances categorized by the total number of variables n and the percentage of UL and LL variables, excluding slack variables: G_1: $n = 40$, G_2: $n = 60$ and, G_3: $n = 100$, with the number of LL variables n_2 equal to $0.3n$, $0.5n$ and $0.8n$. Within each group, there are three types of instances defined by the number of constraints m, which takes the values $0.3n$, $0.5n$ and $0.8n$. The test set includes 30 instances of each type in G_1 and G_2, and 10 instances of each type in G_3. Thus, G_1 and G_2 each have 270 instances, while G_3 has 90 instances, for a total of 630 instances. To ensure the boundedness of the polyhedron, the coefficients of one of the constraints in each instance are generated from a uniform distribution within the interval $(0, 10)$. The coefficients of the objective functions at both levels and the remaining elements of the coefficient matrix are generated from a uniform distribution within the interval $(-10, 10)$. The right-hand side of each constraint was made equal to the sum of the absolute values of the constraint coefficients.

The relaxed problem (3) was solved for each instance. After this step, all trivial instances, i.e., whose solution to problem (3) was a BF solution, were removed from the test set. This step left 195 instances in G_1, 197 in G_2 and 72 in G_3, i.e. a total of 464 instances to which BSS was applied and whose results will be analyzed in the following subsections. All experiments were performed with the number of iterations ($maxIter$) set to five.

Several algorithmic configurations are possible when considering the main BSS elements. We constructed a full factorial experiment to identify the most promising configuration. The four factors that we analyzed are:

- Size of the reference set ($Rsize = 6, 12, 24$)
- Subset generation method (SGM = All, Ord, Dis)
- Solution combination method (SCM = BF, NBF)
- Improvement method (IM = Yes, No)

The full factorial design consists of $3 \times 3 \times 2 \times 2 = 36$ treatments, i.e., configurations of BSS. Note that, since the size of $RefSet$ is one-tenth the size of P, the population sizes to be tested are 60, 120, and 240. Table 1 summarizes the BSS configurations. To simplify the table, only the levels of $Rsize$, SGM and SCM, in this order, are indicated. The configurations are numbered from 1 to 36. Configurations 1 to 18 use the improvement method i.e. IM = Yes. Configurations 19 to 36 (number within a parenthesis) have IM = No.

Table 1
Description of the BSS configurations: $Rsize$-SGM-SCM. The first number corresponds to the case IM = Yes, the number in parentheses corresponds to the case IM = No.

Conf.	Factor levels	Conf.	Factor levels	Conf.	Factor levels
1 (19)	6-All-BF	7 (25)	12-All-BF	13 (31)	24-All-BF
2 (20)	6-All-NBF	8 (26)	12-All-NBF	14 (32)	24-All-NBF
3 (21)	6-Dis-BF	9 (27)	12-Dis-BF	15 (33)	24-Dis-BF
4 (22)	6-Dis-NBF	10 (28)	12-Dis-NBF	16 (34)	24-Dis-NBF
5 (23)	6-Ord-BF	11 (29)	12-Ord-BF	17 (35)	24-Ord-BF
6 (24)	6-Ord-NBF	12 (30)	12-Ord-NBF	18 (36)	24-Ord-NBF

5.2. Analyzing the impact of the number of iterations on solution quality and computational effort.

A run of BSS under a configuration consists of five iterations and X^{best} is the best solution at the end of the last iteration. Since the best solution can be identified at the end of any of the iterations, the first part of the analysis aims to determine the impact of running between 1 and 5 iterations of BSS. For this purpose, for each configuration, let us define $\sharp Ni$ as the number of best solutions found at iteration i. Fig. 3 shows the number of best solutions found at each of the five iterations by each configuration of the procedure, from the red line (one iteration) to the green line (five iterations). The red line labeled $\sharp N1$ shows that all configurations have found the best solution for at least 300 instances in the first iteration. In addition, there are several configurations for which most of the best solutions were obtained in the first iteration. For instance, configuration 13 found 426 out of 464 best solutions in the first iteration. The figure reveals that the number best solutions obtained as the number of iterations increases depends on the configuration. There is a slight change of behavior between the configurations for which IM = Yes (i.e., the first 18) and those with IM = No (i.e., the last 18). Moreover, patterns associated with factor levels can also be seen in the figure. For instance, the three lowest $\sharp N1$ values for configurations 19 to 36 occur when SGM = All and SCM = BF (see configurations 19, 25, and 31). In general, increasing the number of iterations contributes to improving solution quality.

As expected, the more iterations, the greater the computational effort. Fig. 4 shows the average time in seconds (across all instances) required to complete up to the corresponding iteration. The line labeled Iteration i is the average time spent performing iterations 1 through i. Configurations 14 and 32 require the largest amount of time, when IM = Yes and IM = No, respectively, due to $Rsize$ set at 24 and the subset generation method set at All.

Table 2 shows a summary of statistical measures associated with the computational times. Statistics are presented separately according to the number of iterations. For each configuration, the average computational time is computed over all instances. The two first rows in the table display the minimum and the maximum over all configurations of these values. Similarly, for each configuration the maximum computational time is computed, and the remaining rows display the minimum, the maximum and the 90th percentile of these values over all configurations. For example, the average computational time for the first four iterations ranges from 0.53 s to 55.95 s. If five iterations are performed, these values increase to 0.55 and 69.53 s, respectively. Regarding the maximum values, there is an increase from 894.4 s (14.91 min) to 1103.3 s (18.39 min) when increasing the number of iterations from four to five. Finally, from the last row we conclude that most configurations need no more than 262.51 s (4.38 min) to complete five iterations.

Taking into account both the contribution to the number of best solutions and the corresponding computational effort, we have chosen to continue the remaining analysis with the data set corresponding to the completion of the five iterations.

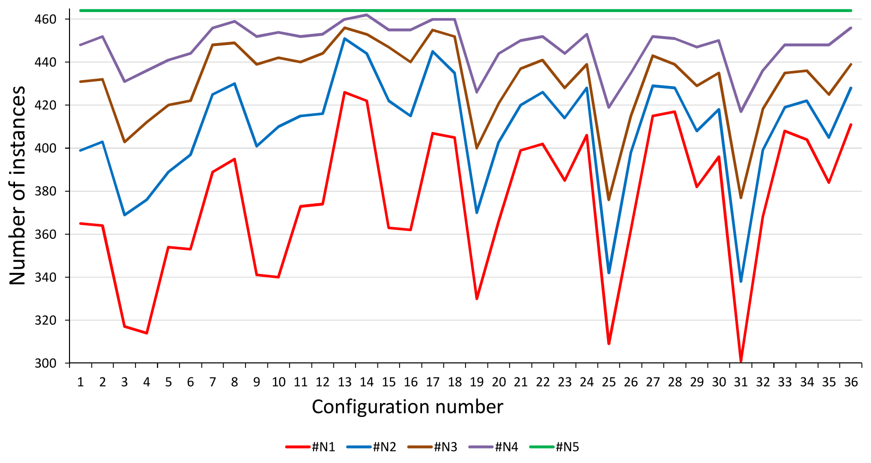

Fig. 3. Number of best solutions available in each of the five iterations.

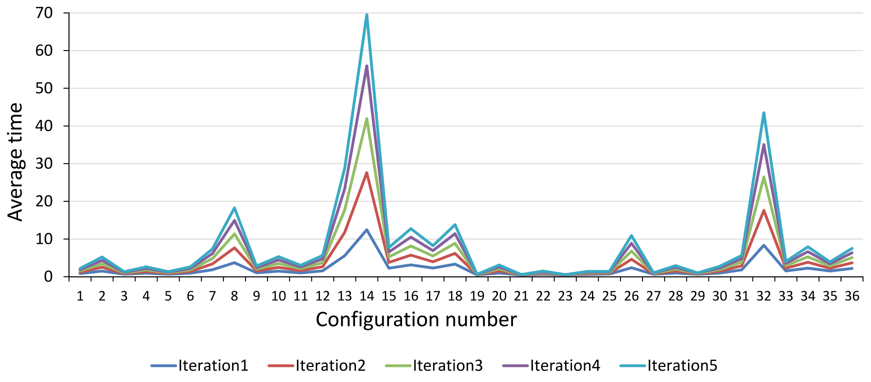

Fig. 4. Average time in seconds spent performing iterations 1 through i for each configuration.

Table 2
Statistical measures of the computational times (in seconds).

	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5
Min of average	0.46	0.49	0.51	0.53	0.55
Max of average	12.45	27.60	41.95	55.95	69.53
Min of maximum	7.63	7.76	7.87	8.00	8.09
Max of maximum	209.96	441.06	665.52	894.40	1103.30
90th Percentile of maximum	56.85	112.22	162.69	211.19	262.51

Table 3
Values of ♯ Success and Rate for the 36 configurations.

Conf.	♯ Success (Rate)	Conf.	♯ Success (Rate)	Conf.	♯ Success (Rate)
1	405 (0.873)	7	443 (0.955)	13	454 (0.978)
2	406 (0.875)	8	447 (0.963)	14	456 (0.983)
3	368 (0.793)	9	400 (0.862)	15	431 (0.929)
4	368 (0.793)	10	401 (0.864)	16	427 (0.920)
5	384 (0.828)	11	425 (0.916)	17	437 (0.942)
6	387 (0.834)	12	423 (0.912)	18	446 (0.961)
19	100 (0.216)	25	104 (0.224)	31	112 (0.241)
20	98 (0.211)	26	104 (0.224)	32	113 (0.244)
21	96 (0.207)	27	99 (0.213)	33	100 (0.216)
22	96 (0.207)	28	99 (0.213)	34	101 (0.218)
23	99 (0.213)	29	102 (0.220)	35	104 (0.224)
24	98 (0.211)	30	104 (0.224)	36	106 (0.228)

5.3. Effect analysis of the algorithm design factors on solution quality and computational time

In order to select the best configuration of BSS, for each instance, we computed BAC, the best solution obtained across the 36 configurations. Then, for each configuration we defined ♯Success as the number of instances for which the best value provided by the configuration equals BAC and Rate as the success rate, i.e. Rate = ♯Success/464. Table 3 displays ♯Success and Rate for each configuration. The value of Rate for configurations 1 to 18 for which IM = Yes is clearly higher with a difference of up to 0.74. Configuration 14 yields the highest performance based on the success metric, matching 456 instances with a Rate of 0.98. These results suggest that, regardless of other settings, BSS should include the improvement method. Therefore, the remainder of the analysis assumes that IM = Yes and focuses on the other three factors.

5.3.1. Influence of algorithm design factors on solution quality

Next, we assess Rate with respect to $Rsize$, SGM, and SCM along with the parameters that characterize the instance size, i.e. n, n_2%, i.e. the percentage that defines n_2 from n, and m%, i.e. the percentage that defines m from n. Fig. 5 shows the main effect plots with the average values of Rate for each level of the corresponding factor. The top three plots are related to the instance size. These plots show that as the size of the problem increases Rate decreases. The bottom three plots are related to the BSS configuration. In this case, increasing $Rsize$ leads to improved results. SGM = All provides the best results, with the other two levels being quite similar. Main effects for SCM show no difference between both levels.

Fig. 6 shows the interaction effects plots. These plots help us to understand the effect of one factor depending on the value of another factor. The top three plots show the interaction effects among size instance factors. As n increases, the decrease in Rate is greater with the increase in n_2% and m%. The bottom three plots show the interaction effects among factors related to the BSS configuration. No interaction between SCM and the other two factors is observed, as shown in the

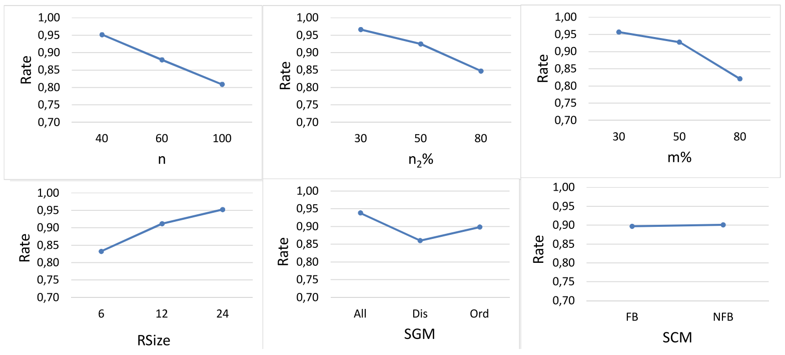

Fig. 5. Main effects plots for rate.

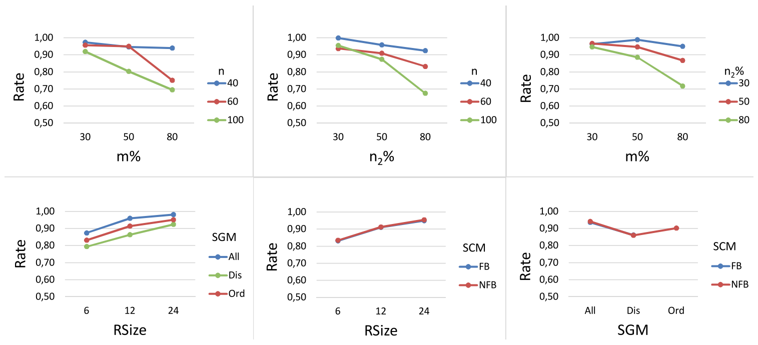

Fig. 6. Interactions effects plots for rate.

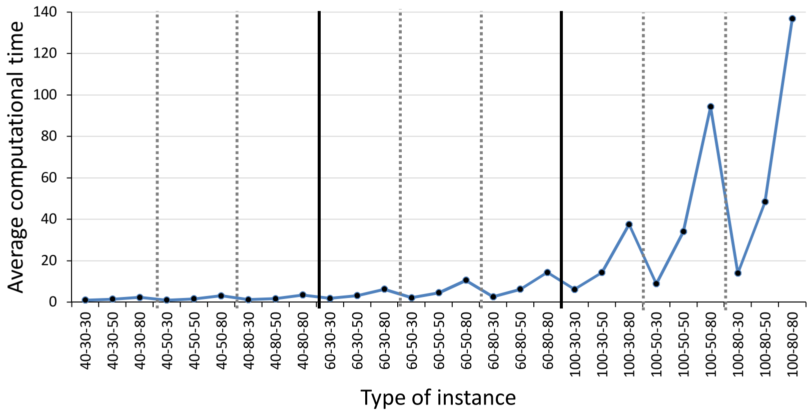

Fig. 7. Average computational time in seconds of each type of instance identified by $n - n_2\% - m\%$.

bottom middle and right-hand-side plots in which both lines connecting Rate coincide for SCM = BF and SCM = NBF. The bottom left-hand-side plot shows that Rate increases with the value of $RSize$ regardless of the level of SGM. The above analyses reveal that the variability of Rate is primarily due to the size of the problem, although the $RSize$ and SGM factors are also significant.

5.3.2. Influence of algorithm design factors on computational time

As might be expected, computational time is also impacted by the size of the instance. Fig. 7 depicts the average computational time for five iterations of BSS configurations 1 to 18. Each type of instance is labeled on the x-axis as $n - n_2\% - m\%$. The effect of $n_2\%$ and $m\%$ on the computational time is more noticeable as n increases. The largest time values (more than one minute of computational time) occur when $n = 100$ and $m\% = 80$. Table 4 shows statistics of the computational time in seconds. For each configuration, the minimum, average and maximum over the 464 instances are presented. Clearly, computational times increase with $RSize$. The highest computational effort is associated with $RSize$ = 24 and SGM = All, where configuration 14 reached 1103.3 s in one instance.

Table 4
Descriptive statistics of the computational time. For BSS configurations 1 to 18: minimum, average and maximum over the total of instances.

Conf.	Min.	Aver.	Max.	Conf.	Min.	Aver.	Max.	Conf.	Min.	Aver.	Max.
1	0.1	2.2	47.7	7	0.4	7.3	119.7	13	2.4	28.9	422.2
2	0.1	5.2	106.4	8	0.5	18.3	316.6	14	2.5	69.5	1103.3
3	0.1	1.3	32.1	9	0.3	2.8	46.3	15	1.9	7.8	84.1
4	0.1	2.6	50.1	10	0.3	5.3	91.0	16	1.9	12.7	160.0
5	0.1	1.3	29.8	11	0.3	3.0	52.4	17	1.8	8.2	103.7
6	0.1	2.6	53.2	12	0.3	5.6	96.0	18	1.8	13.8	199.4

In order to assess the effect of the $RSize$, SGM and SCM factors on the computational time, the main effects plots are shown at the top of Fig. 8 while the pairwise interactions plot are displayed at the bottom.

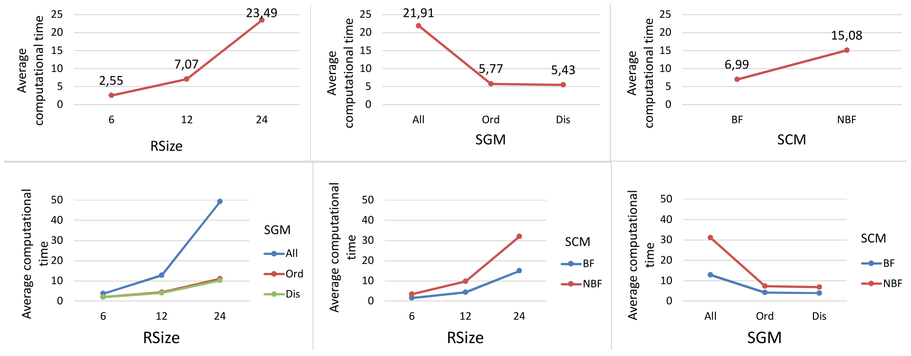

Fig. 8. Main and interaction effects plots for computational time.

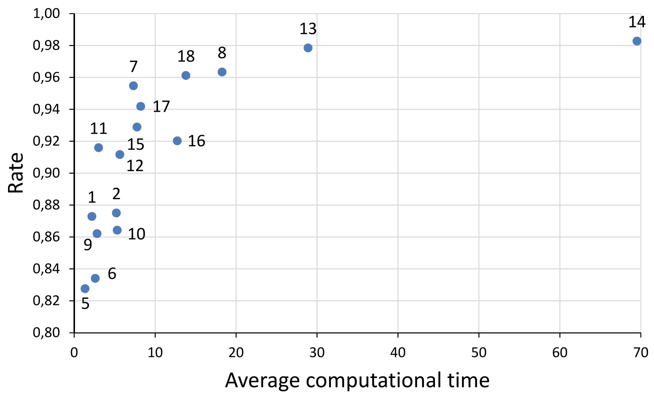

Fig. 9. Rate vs average computational time for the BSS configurations 1 to 18.

Top plots show that the computational effort increases with the value of $RSize$, SGM = All and SCM = NBF. The interaction effects plots show that the computational time increases for $RSize$ = 24 and SGM = All. Higher average computational time values are also observed for SCM = NBF whatever the combination of the other two factors. As $Rsize$ increases or SGM = All, the effect on the average computational time is smaller when SCM = BF compared to SCM = NBF.

Finally, Fig. 9 compares average computational time versus Rate for each configuration. Clearly, a configuration is better than another when it has a higher Rate and needs less computational time. As the figure shows, computational time and Rate are conflicting criteria, since, in general, improving one makes the other worse. From a bi-objective perspective, some configurations are dominated by others, that is, some configurations are outperformed by others in both criteria. For instance, configuration 11 outperforms configurations 2, 10 and 12. The set of dominated configurations is {2, 6, 9, 10, 12, 15, 16, 17}. From the set of non-dominated configurations {1, 3, 4, 5, 7, 8, 11, 13, 14, 18}, we choose configuration 13 instead of configuration 14 because, in our opinion, the increased Rate does not compensate for the significant increase in computational time. However, should protection against excessively long computational times be important, configuration 7 seems the best compromise. This configuration achieves a Rate of 95% with an average computational time of 7.3 s and a maximum of 119.7 s compared to the 98% Rate of configuration 13 with an average computational time of 28.9 s and a maximum of 422.2 s.

5.3.3. Characteristics of the chosen BSS configurations.

From the previous analysis, we conclude that, in terms of Rate, IM = Yes should be selected. Moreover, as expected, the problem size has the greatest influence on Rate, whose value decreases as the size of the problem increases. Regarding the other factors, increasing $Rsize$ leads to improved results and SGM = All provides the best outcomes. When

Table 5
Selected configurations.

Configuration	IM	$Rsize$	SGM	SCM
BSS_7	Yes	12	All	BF
BSS_{13}	Yes	24	All	BF

computational time is considered, the configurations with $Rsize$ = 24 and SGM = All have the highest values. Additionally, the computational effort increases with $Rsize$, SGM = All and SCM = NBF.

Therefore, we select configurations 7 and 13, which will be denoted as BSS_7 and BSS_{13} from now on, and whose characteristics are displayed in Table 5. The performance evaluation of BSS, which involves comparing it with the results provided by other algorithms, will be limited to these two configurations.

6. Computational study: Evaluation of the performance of BSS

In this part of the computational study, the objective is to gather evidence regarding the quality of BSS. For this purpose, BSS is compared with BilevelJuMP.jl (BJu), a Julia package designed to support bilevel optimization within the JuMP framework (Garcia et al., 2024), which is the most recent and effective algorithm for solving LBPs. We used Julia v1.11.1, BilevelJuMP v0.6.2, and CPLEX v22.1.1 to solve the LBPs. CPLEX was configured with default settings, except for a few modifications: the time limit was set to 3600 s, the number of threads was set to one, and the integrality tolerance was set to 1e-6. The LBPs were solved using the SOS1 reformulation mode in BilevelJuMP.

In this study, we also include the results provided by Calvete et al. (2008), which involve applying the Kth-best exact algorithm (Bialas & Karwan, 1984) to solve instances of groups G_1 and G_2, as well as the genetic algorithm GAH proposed by Hejazi et al. (2002), and the genetic algorithm GABB developed by Calvete et al. (2008) to solve instances of groups G_1, G_2 and G_3.

In addition to the groups of instances G_1, G_2 and G_3, two larger sets were generated in the same manner as described in Section 5.1. Group G_4 consists of 30 instances with $n = 200$, 10 of which have $n2 = 0.5n$ and $m = 0.8n$, 10 have $n2 = 0.8n$ and $m = 0.5n$, and the remaining 10 have $n2 = 0.8n$ and $m = 0.8n$. Group G_5 has a similar composition, but with $n = 300$.

At the end of Section 5.2, for the groups of instances G_1, G_2 and G_3 we decided to continue the computational study assuming that BSS runs five iterations. However, as the size of the instance increases, sometimes the time required for the five iterations exceeds one hour, making it difficult to compare the results with BJu. Hence, for groups G_4 and G_5, we established a stop criterion based on either running five iterations or reaching a time limit of 3600 s, whichever occurs first. Once BSS stops, the best feasible solution found is provided.

Table 6 displays global results of BJu. The first column shows the name of the instance group, while the second column indicates the

Table 6
Results provided by BilevelJuMP.jl after 3600 s.

Group	# Nontrivial	BJu		
		Optimal	Best feasible solution	No feasible solution
G_1	195	195	–	–
G_2	197	190	6	1
G_3	72	57	8	7
G_4	30	2	8	20
G_5	30	1	8	21

Table 7
Number of instances with a solution equal to (E) or worse than (W) the best known solution.

Group	# Nontrivial	Kth-Best		GAH		GABB		BSS$_7$		BSS$_{13}$		BJu	
		E	W	E	W	E	W	E	W	E	W	E	W
G_1	195	108	87	153	42	166	29	188	7	194	1	195	0
G_2	197	50	147	123	74	135	62	188	9	190	7	191	6
G_3	72	–	–	22	50	39	33	65	7	68	4	57	15
G_4	30	–	–	–	–	–	–	17	13	24	6	2	28
G_5	30	–	–	–	–	–	–	22	8	19	11	3	27

Table 8
Number of instances with a solution better than (B), equal to (E) or worse than (W) the solution provided by BJu.

Group	# Nontrivial	BSS$_7$			BSS$_{13}$		
		B	E	W	B	E	W
G_1	195	0	188	7	0	194	1
G_2	197	5	184	8	5	185	7
G_3	72	14	55	3	14	56	2
G_4	30	28	1	1	28	1	1
G_5	30	27	1	2	27	1	2

number of nontrivial instances in each group. The fourth, fifth and sixth columns show the number of instances in which BJu finds an optimal solution, finishes with a feasible solution after 3600 s, and fails to provide a feasible solution after 3600 s, respectively. Note that BJu finds an optimal solution for all the smallest instances in group G_1, 190 out of 197 instances in group G_2, and 57 out of 72 instances in group G_3. However, it fails to find any feasible solution after one hour of computation in 49 out of 524 instances. More concerning, it finds no feasible solutions in 41 out of 60 larger instances.

Next, the best available objective function value has been calculated for each instance in each group. Table 7 displays the number of instances in which each procedure matches the best known solution. The first column displays the name of the instance group, and the second column indicates the number of nontrivial instances in each group. The remaining columns are grouped in blocks of two: for each algorithm, the first column (E) shows the number of instances in which the algorithm matches the best known solution, and the second column (W) shows the number of instances with a worst value. The "–" symbol means that the algorithm was not applied to the corresponding group of instances. From this table, it is clear that the two configurations of BSS and Bju outperform the results of the Kth-Best, GAH and GABB in all groups. Moreover, BSS$_{13}$ and BJu are basically equal in groups G_1 and G_2, with BSS$_7$ being slightly worst. However, as the instance size increases, BSS$_7$ and BSS$_{13}$ are notably better than BJu. This is confirmed in Table 8, where each BSS configuration and Bju are directly compared. In this table, each block of three columns shows the number of instances in which the corresponding BSS configuration provides a solution that is better, equal to, or worse than the solution provided by BJu. As the instance size increases, the number of instances in which BSS outperforms BJu grows, regardless of the configuration.

Let $Best$ be the best objective function value available for each instance. Next, for each algorithm and group, we focus on those instances

Table 9
Statistical summary for the percentage gap.

Group	BSS$_7$				BSS$_{13}$				BJu			
	# ins.	Mean	Min	Max	# ins.	Mean	Min	Max	# ins.	Mean	Min	Max
G_1	7	2.50	0.13	6.81	1	2.31	2.31	2.31	–	–	–	
G_2	9	4.55	0.17	16.96	7	2.47	0.17	11.22	5	45.54	5.05	131.08
G_3	7	1.03	0.04	3.06	4	0.95	0.26	2.50	8	11.57	0.26	39.64
G_4	13	1.65	0.02	7.48	6	1.48	0.33	4.02	8	11.98	1.00	43.73
G_5	8	15.64	0.42	91.09	11	2.51	0.38	5.68	6	10.91	0.04	30.19

Table 10
Statistical summary for computational time in seconds.

Group	BSS$_7$			BSS$_{13}$			BJu		
	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
G_1	1.32	0.41	3.56	5.89	2.37	15.28	0.69	0.04	15.78
G_2	4.62	0.92	14.87	17.87	4.43	54.06	155.32	0.04	3600.11
G_3	31.03	3.68	119.68	121.41	15.69	422.15	804.87	0.08	3629.74
G_4	2470.32	458.25	3600.00	3345.66	1340.69	3600.00	3406.50	7.08	3703.69
G_5	3561.89	2760.14	3600.00	3600.00	3600.00	3600.00	3505.42	748.37	3613.17

where the objective function value at termination, denoted $Value$, does not reach $Best$. For these instances, we compute the percentage gap, defined as:

$$PGap = \frac{(Best - Value)}{Best} \times 100 \tag{7}$$

Table 9 shows several statistical measures related to $PGap$. The first column indicates the group. The remaining columns are grouped into blocks: for each algorithm, the first column in each block (# ins.) shows the number of instances involved in the computation, while the second, third, and fourth columns show the mean, the minimum, and the maximum values of $PGap$ for the instances, respectively. The "–" symbol means that no instances exist where the objective function value is worse than $Best$. Notice that the second and sixth columns in Table 9 correspond to the tenth and twelfth columns in Table 7, respectively. However, the tenth column in Table 9 does not coincide with the last column in Table 7, as the latter includes instances for which BJu has not been able to find a feasible solution, and thus $PGap$ cannot be calculated for these instances. From Table 9, it is clear that BSS_{13} demonstrates the best performance, except for group G_1, where BJu finds the optimal solution for all instances. When BSS_{13} does not yield the best objective function value, it provides solutions that have small variability in their objective function values.

Finally, we focus on the computational effort. The structure of Table 10 is similar to that of Table 9, except that the columns related to the number of instances have been removed, as all instances in each group have been considered in the computations. Both configurations of BSS provide lower computational times in groups G_1, G_2 and G_3, with BSS$_7$ being slightly better. Groups G_4 and G_5 are comparable for both configurations and BJu, as for most instances they reach the time limit.

As a result of the previous analysis, we conclude that in terms of the ability to provide the best results and the time invested to achieve them, configuration BSS$_{13}$ outperforms all the other alternatives that we analyzed.

7. Conclusions

This paper presents an efficient SS with path relinking algorithm for solving the LBP. Two main properties of the LBP underlie its development: (1) there is an extreme point of the constraint region which solves the LBP and (2) there is a boundary feasible extreme point that solves the LBP. Thus, the first property leads to one of the novelties of the SS approach proposed since it enables us to search only in the discrete space of extreme points. The second property makes it possible for us to propose a tailored path relinking that searches for trajectories from feasible extreme points to infeasible extreme points, which necessarily pass through at least one boundary feasible extreme point.

Each of the stages of the scatter search has been designed ad hoc for the LBP. The diversification method generates two subpopulations of extreme points, feasible or infeasible for the LBP, aiming to cover the faces of the polyhedron defined by the constraints. In order to build the reference set, a distance measure is introduced which takes into account that the algorithm operates on extreme points. Hence, it measures the number of basic variables that are not shared. Then, the reference set takes extreme points of each of above mentioned subpopulations. In the path relinking procedure a trajectory of adjacent extreme points is built going from a feasible extreme point to an infeasible one or vice versa, in order to find a boundary feasible extreme point.

Several variants of the algorithm are proposed regarding the subset generation method and the solution combination method whose. After a boundary feasible extreme point has been provided by the combination method, a local improvement search can be applied with the purpose of obtaining a solution with a higher UL objective function value. The computational experiment has assessed the quality of the solution provided by 36 configurations of the algorithm on 464 benchmark instances as well as the computational time invested. As a result, two configurations have been selected as the most promising.

Finally, the performance of the both BSS configurations has been evaluated in comparison with BilevelJuMP.jl, as well as the Kth-best, GAH, and GABB algorithms, when the results are available in the literature. BSS clearly shows better performance than the last three methods. When compared with BilevelJuMP.jl, it is clear that this procedure outperforms BSS in the smallest instances. However, its performance deteriorates as the problem size increases. It fails to find feasible solutions for a significant number of large instances, whereas proposed SS reaches feasibility. Therefore, we can confidently conclude that both BSS configurations are valuable tools for solving large LBP instances.

We hope that our work inspires others to continue the exploration of the application of scatter search and path relinking to bilevel problems. These problems are, in general, very complex and without a defined structure. However, as we showed here, taking advantage of some of their theoretical properties and problem characteristics can lead to effective solution procedures. Future research directions could involve, for instance, linear fractional bilevel problems, for which a similar property to the existence of a boundary feasible extreme point that solves the problem is verified. Moreover, other optimization problems with extreme point optimal solutions could also benefit from the use of boundary paths between extreme points in the solution process.

CRediT authorship contribution statement

Herminia I. Calvete: Writing – review & editing, Writing – original draft, Validation, Methodology, Conceptualization. **Carmen Galé:** Writing – review & editing, Writing – original draft, Validation, Methodology, Conceptualization. **José A. Iranzo:** Writing – review & editing, Writing – original draft, Software, Methodology, Conceptualization. **Manuel Laguna:** Writing – review & editing, Methodology, Conceptualization.

References

Ahuja, R. K., & Orlin, J. B. (1997). Commentary—Developing fitter genetic algorithms. *INFORMS Journal on Computing*, *9*(3), 251–253.

Andrade, C. E., Toso, R. F., Gonçalves, J. F., & Resende, M. G. (2021). The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications. *European Journal of Operational Research*, *289*(1), 17–30.

Audet, C., Haddad, J., & Savard, G. (2006). A note on the definition of a linear bilevel programming solution. *Applied Mathematics and Computation*, *181*, 351–355.

Bard, J. (1998). *Practical bilevel optimization. Algorithms and applications*. New York: Springer.

Bialas, W., & Karwan, M. (1982). On two-level optimization. *IEEE Transactions on Automatic Control*, *27*, 211–214.

Bialas, W., & Karwan, M. (1984). Two-level linear programming. *Management Science*, *30*, 1004–1024.

Calvete, H., & Galé, C. (2020). Algorithms for linear bilevel optimization. In S. Dempe, & A. Zemkoho (Eds.), *Bilevel optimization: advances and next challenges* (pp. 293–312). Cham: Springer International Publishing.

Calvete, H. I., Galé, C., & Mateo, P. M. (2008). A new approach for solving linear bilevel problems using genetic algorithms. *European Journal of Operational Research*, *188*, 14–28.

Camacho-Vallejo, J.-F., Corpus, C., & Villegas, J. G. (2024). Metaheuristics for bilevel optimization: A comprehensive review. *Computers & Operations Research*, *161*(106410), 1–26.

Camacho-Vallejo, J.-F., Muñoz-Sánchez, R., & González-Velarde, J. L. (2015). A heuristic algorithm for a supply chain's production-distribution planning. *Computers & Operations Research*, *61*, 110–121.

Casas-Ramírez, M.-S., & Camacho-Vallejo, J.-F. (2015). Solving the p-median bilevel problem with order through a hybrid heuristic. *Applied Soft Computing*, *60*, 73–86.

Dempe, S. (2002). *Foundations of bilevel programming*. Dordrecht, Boston, London: Kluwer Academic Publishers.

Dempe, S., & Zemkoho, A. (2020). *Bilevel optimization. Advances and next challenges*. Berlin: Springer.

Garcia, J. D., Bodin, G., & Street, A. (2024). BilevelJuMP.jl: Modeling and solving bilevel optimization problems in julia. *INFORMS Journal on Computing*, *36*(2), 327–335.

Glover, F., Laguna, M., & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, *29*(3), 653–684.

González Velarde, J. L., Camacho-Vallejo, J.-F., & Pinto Serrano, G. (2015). A scatter search algorithm for solving a bilevel optimization model for determining highway tolls. *Computación Y Sistemas*, *19*(1), 5–16.

Hansen, P., Jaumard, B., & Savard, G. (1992). New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, *13*, 1194–1217.

Hejazi, S., Memariani, A., Jahanshahloo, G., & Sepehri, M. (2002). Linear bilevel programming solution by genetic algorithm. *Computers and Operations Research*, *29*, 1913–1925.

Henke, D., Lefebvre, H., Schmidt, M., & Thürauf, J. (2025). On coupling constraints in linear bilevel optimization. *Optimization Letters*, *19*, 689–697.

Kalra, M., Tyagi, S., Kumar, V., Kaur, M., Khan Mashwani, W., Shah, H., & Shah, K. (2021). A comprehensive review on scatter search: Techniques, applications, and challenges. *Mathematical Problems in Engineering*, *2021*(5588486), 21.

Laguna, M. (2014). Scatter search. In E. K. Burke, & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (pp. 119–141). Boston, MA: Springer US.

Liu, Y., & Hart, S. (1994). Characterizing an optimal solution to the linear bilevel programming problem. *European Journal of Operational Research*, *73*(1), 164–166.

Maldonado-Pinto, S., Casas-Ramírez, M.-S., & Camacho-Vallejo, J.-F. (2016). Analyzing the performance of a hybrid heuristic for solving a bilevel location problem under different approaches to tackle the lower level. *Mathematical Problems in Engineering*, *2016*(1), Article 9109824.

Martí, R., Laguna, M., & Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, *169*(2), 359–372.

Mersha, A., & Dempe, S. (2006). Linear bilevel programming with upper level constraints depending on the lower level solution. *Applied Mathematics and Computation*, *180*, 247–254.

Molina, J., Laguna, M., Martí, R., & Caballero, R. (2007). SSPMO: A scatter tabu search procedure for non-linear multiobjective optimization. *INFORMS Journal on Computing*, *19*(1), 91–100.

Pérez Posada, A. F., Villegas, J. G., & López-Lezama, J. M. (2017). A scatter search heuristic for the optimal location, sizing and contract pricing of distributed generation in electric distribution systems. *Energies*, *10*(1449), 1–16.

Polino, S., Camacho-Vallejo, J.-F., & Villegas, J. G. (2024). A facility location problem for extracurricular workshop planning: bi-level model and metaheuristics. *International Transactions in Operational Research*, *31*(6), 4025–4067.

Resende, M. G., Ribeiro, C. C., Glover, F., & Martí, R. (2010). Scatter search and path-relinking: Fundamentals, advances, and applications. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (pp. 87–107). Boston, MA: Springer US.

Savard, G. (1989). *Contribution à la programmation mathématique à deux niveaux* (Ph.D. thesis), Université de Montréal, Montréal, QC, Canada: Ecole Polytechnique de Montréal.