People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Borj Bou Arréridj
Faculty of Mathematics and Computer Science
Department of Mathematics

THESIS

In order to obtain the Doctorate degree in LMD (3rd cycle)
Branch : Applied Mathematics
Option : Modelization, Control and Optimization

THEME

Etude d'un problème d'optimisation à deux niveaux multicritères

By: Wafa Bouguern

Publicly defended on: 17/05/2026

In front of the jury composed of:

Dr. Zeghdane Rebiha	University of B.B.A	**President**
Dr. Adddoune Smail	University of B.B.A	**Supervisor**
Dr. Debbiche Hanene	University of B.B.A	**Co-Supervisor**
Dr. Brahmi Boualam	University of B.B.A	**Examiner**
Pr. Zouache Djaafar	University of B.B.A	**Examiner**
Dr. Khelladi Samia	University of Sétif 1	**Examiner**

2025/2026

Contents

List of Figures

List of Tables

Notations

Symbol	Meaning
$\mathbb{R}^q_+$	The nonnegative orthant in $\mathbb{R}^q$, i.e., $\{x \in \mathbb{R}^q : x_i \geq 0 \ \forall i\}$
$\mathbb{R}^q_{++}$	The strictly positive orthant in $\mathbb{R}^q$, i.e., $\{x \in \mathbb{R}^q : x_i > 0 \ \forall i\}$
$\overline{\mathbb{R}}$	$\mathbb{R} \cup \{-\infty, +\infty\}$
$\langle \cdot, \cdot \rangle$	Inner product in Euclidean space
$\|\cdot\|$	Euclidean norm
$u \cdot v$	Hadamard product of vectors u and v
e	The vector of all ones
$u < v$	$u_i < v_i$ for all $i = 1, \ldots, n$
$u \leq v$	$u_i \leq v_i$ for all $i = 1, \ldots, n$
$u \prec v$	$u \leq v$ and $u \neq v$
$d(u, v)$	Euclidean distance between vectors u and v
$\Pi(u, S)$	Euclidean projection of u onto the set S
$\mathrm{dom}(f)$	Domain of a function f
$\mathrm{epi}(f)$	Epigraph of a function f

Table 1: List of notations

Introduction

Bilevel optimization problems were first introduced by H. von Stackelberg in 1934. The mathematical study of this type of problem began in the 1970s. Since then, many researchers have been interested in this model, either for its applicability to practical problems in economics, engineering, chemistry, and biology or for its theoretical and numerical analysis.

In game-theoretic terms, the Stackelberg model describes a hierarchical game in which two decision-makers strive to achieve specific goals. The leader aims to minimize their objective function while accounting for the follower's response. The leader's decision affects the feasible region and the objective function of the follower's problem, and the follower's response then affects the leader's objective. Thus, the bilevel optimization problem represents the leader's dilemma, with the follower's dilemma serving as one of its constraints.

Bilevel optimization problems are NP-hard because constraints are violated at every point within the feasible domain, making the search for an optimal solution challenging. Furthermore, these problems are non-convex and non-differentiable, even when the data are linear at both levels, which introduces an additional degree of complexity. The existence of optimal solutions and the convergence of algorithms are closely related to the continuity of the set-valued mappings associated with the problem.

The literature offers several methods to address these challenges, including the Karush-Kuhn-Tucker technique (see, e.g., [3], [31]), penalty approaches ([7], [8]), and branch-and-bound methods [13], among others.

Other research efforts have focused on DC (Difference of Convex functions) programming and the DC algorithm (DCA), first introduced by Pham Dinh Tao in 1986, and have been extensively developed since 1994 (see [60], [61]). A notable contribution was made in 2012 by Le Thi Hoai An and Tran Duc Quynh [63], who applied DC programming and DCA to solve bilevel problems with a quadratic and convex upper-level

objective function, while the lower-level objective function and constraints remained linear. In addition, another study [12] reformulated the bilevel programming problem as a single-level optimization problem using the Karush-Kuhn-Tucker (KKT) conditions. Following this reformulation, DC programming techniques have been applied, particularly to problems with quadratic upper-level and linear lower-level objective functions. More recently, DC programming has been widely applied to solve various classes of bilevel programming problems (see [35], [76]). These recent studies highlight the versatility and efficiency of DC programming and DCA in addressing complex bilevel problems.

In further work, academics often use metaheuristic methods to address bilevel programming challenges due to their advantages. Mathieu et al. [70] presented one of the pioneering evolutionary algorithms (EAs) for solving bilevel programming problems [50]. Subsequent developments have led to the proposal of several EA-based approaches, as highlighted in [1] and [49]. The genetic algorithm (GA) is the most prominent variant of evolutionary algorithms. Several genetic algorithms GAs have been proposed in [18], [50], [58], and [71] for solving bilevel programming problems. In addition, some researchers have focused on other techniques such as tabu search (TS) [70], bat algorithm (BA) [69], differential evolution (DE) [10], and particle swarm optimization (PSO) [42], [45], [79].

The Gray Wolf Optimizer (GWO) is a metaheuristic method introduced in 2014 by Seyedali Mirjalili [54]. It is a population-based metaheuristic that mimics the natural hierarchical organisation and hunting strategies of grey wolves. The majority of the literature (see, e.g., [37], [50], [82]) advocates a nested methodology for solving bilevel programming problems, using a classical technique for the lower-level and an evolutionary algorithm for the upper-level. However, this technique is computationally intensive and inefficient for large-scale problems.

Bilevel multicriteria optimization is a contemporary area of interest. It involves tackling multicriteria optimization problems at the first and/or second levels. Solving these problems is a significant task for the mathematical community. Some research has focused on applying evolutionary algorithms to bilevel multicriteria optimization problems (BLMOPs) (see [5], [24], [66], [78]). We can use three categories to classify

BLMOPs. First, there are several objectives at the upper level and just one objective at the lower level ([5],[19]); second, there is a single objective at the upper level and multiple objectives at the lower level ([11], [17], [20],[40], [65]); and the last category is multiple objectives at both levels ([4], [36], [44], [48], [62], [77], [80], [81]).

The general form of a bilevel programming problem can be represented as follows:

$$(BP) \quad \begin{cases} \underset{x,y}{\text{Minimize}} \ F(x,y) \\ \text{subject to} \\ G(x,y) \leq 0, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ denotes the set of optimal solutions corresponding to the lower-level problem

$$\begin{cases} \underset{y}{\text{Minimize}} \ f(x,y) \\ \text{subject to} \\ g(x,y) \leq 0, \end{cases}$$

In this formulation, the functions F and $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ are the objective functions of the upper and lower levels, respectively. The constraint functions $G : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_1}$ and $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_2}$ define the feasible regions for the upper and lower levels, respectively. The vector x stands for the decision variables at the upper level, while y corresponds to those at the lower level.

Tackling bilevel problems directly is often challenging. To tackle this issue, many researchers have devised methods to convert the bilevel formulation into a single-level equivalent. There are two primary strategies for doing so. The first strategy consists of substituting the lower-level problem with its corresponding Karush–Kuhn–Tucker (KKT) conditions. A comprehensive overview of this method, along with its benefits and limitations, is provided in [28].

The second strategy, adopted in this work, involves reformulating the lower-level problem using its value function, known as the Lower-Level Value Function Reformulation (LLVFR) (see, e.g., [59, 75]). This method allows conventional optimization

techniques by reformulating the bilevel model as a single-level problem:

$$\begin{cases} \underset{x,y}{\text{Minimize}} \;\; F(x,y) \\ \text{subject to} \\ G(x,y) \leq 0, \\ g(x,y) \leq 0, \\ f(x,y) - \varphi(x) \leq 0, \end{cases}$$

where the lower-level value function is defined as

$$\varphi(x) = \inf_{y} \{ f(x,y) \mid g(x,y) \leq 0 \}.$$

Note that the feasible set resulting from this reformulation is generally non-convex. The presence of the non-differentiable function $\varphi(x)$ introduces additional complexity to the problem and poses significant challenges for standard optimization techniques.

This thesis offers a comprehensive overview of multicriteria and bilevel programming problems, treating each topic separately. We review commonly used solution methods for both problem types. Additionally, we present our original contributions and summarize the results of numerical experiments. Below, we outline the main contributions of this work.

Work 1: We focus on a class of bilevel programming problems in which the lower-level problem is nonconvex and the objective function is coupled with both the leader's and the follower's decision variables. We propose an algorithm for solving this class of problems using DC (Difference of Convex functions) programming and demonstrate its effectiveness through illustrative examples. The results of this work are presented in our research paper [16].

Work 2: We examine another class of bilevel programming problems in which the variables are subject to box constraints. To address this setting, we propose an approximation method based on α-dense curves and solve the resulting problem using an evolutionary algorithm. This work forms the basis of research paper [15].

Chapter 1

Fundamentals of multicriteria optimization

1.1 Introduction

Multicriteria programming is a specialized field for addressing complex decisions that require the simultaneous achievement of multiple objectives. Take, for instance, the decision to purchase a vehicle. In such a decision, the buyer aims to accomplish interconnected objectives. They may prefer a vehicle that is affordable to minimize initial expenses, while being fuel-efficient to ensure long-term economic viability. Furthermore, safety, design, and comfort factors may hold substantial importance. In these scenarios, multicriteria programming balances multiple objectives and identifies the optimal solution that best meets the customer's needs.

This chapter provides an overview of the basic concepts of multicriteria programming. Includes a general introduction to its principles and an explanation of how it helps solve problems that involve optimizing multiple objectives.

1.2 Basic concepts

1.2.1 Problem statement

Let X be a non-empty subset of $\mathbb{R}^n$, and $f : X \to \mathbb{R}^q$ (with $q \geq 2$) be a mapping representing q objective functions. We consider the following multicriteria optimization

problem:

$$(MP) \quad \begin{cases} \underset{x}{\text{Minimize}}\ f(x) := \big(f_1(x), \ldots, f_q(x)\big) \\ \text{subject to} \\ x \in X. \end{cases}$$

The objective here is to find a feasible solution $x \in X$ that simultaneously minimizes all the objective functions, $f_1, \ldots, f_q$.

The space $\mathbb{R}^n$, which contains the set X of feasible points, is called the **decision space**. The set $Y = f(X)$, referred to as the set of feasible vectors, lies in the **criteria space** $\mathbb{R}^q$.

When faced with a multicriteria problem, the solution is not just one optimal point; it is a set of efficient or Pareto-optimal solutions, which are discussed in greater detail in the following sections.

1.2.2 Dominance and efficiency

Definition 1.1 ***(Dominance)****[33] In the criteria space, a vector* $\mathbf{y}$ *is said to dominate another vector* $\mathbf{z}$*, denoted by* $\mathbf{y} \prec \mathbf{z}$*, if* $y_i \leq z_i$ *for all* $i = 1, \ldots, q$ *and* $y_j < z_j$ *for at least one* j*.*

A vector $\mathbf{y} \in Y$ is said to be non-dominated if there does not exist any other vector in Y that dominates it.

Definition 1.2 ***(Efficiency)****[33] A point* $x^* \in X$ *is Pareto optimal (or efficient) if there is no other point* $x \in X$ *that dominates* x^**. That is, it is not possible to have* $f_i(x) \leq f_i(x^*)$ *for all* $i = 1, \ldots, q$*, with* $f_k(x) < f_k(x^*)$ *for at least one* k*.*

Each non-dominated vector corresponds to an efficient point, and vice versa.

The set of non-dominated vectors is called the Pareto front and denoted by Y_N. Similarly, $X_e \subset X$ represents the set of efficient points.

Definition 1.3 ***(Weakly efficient)****[33] A point* $x^* \in X$ *is said to be weakly efficient if there is no point* $x \in X$ *such that* $f(x) < f(x^*)$*. The corresponding point* $y^* = f(x^*)$ *is referred to as weakly non-dominated.*

Definition 1.4 ***(Strictly efficient)****[33] A point* $x^* \in X$ *is said to be strictly efficient if there is no point* $x \in X$ *such that* $x \neq x^*$ *and* $f(x) \leq f(x^*)$*.*

The sets of weakly efficient points, strictly efficient points, and weakly non-dominated points will be denoted by X_{we}, X_{se}, and Y_{wN}, respectively.

Remark 1.1 *Let $x^* \in X$, we have (see Figure 1.1.):*

- $x^* \in X_e \iff \left(f(x^*) - \mathbb{R}^q_+\right) \cap Y = \{f(x^*)\}$.
- $\hat{x} \in X_{we} \iff \left(f(\hat{x}) - \mathbb{R}^q_{++}\right) \cap Y = \emptyset$.

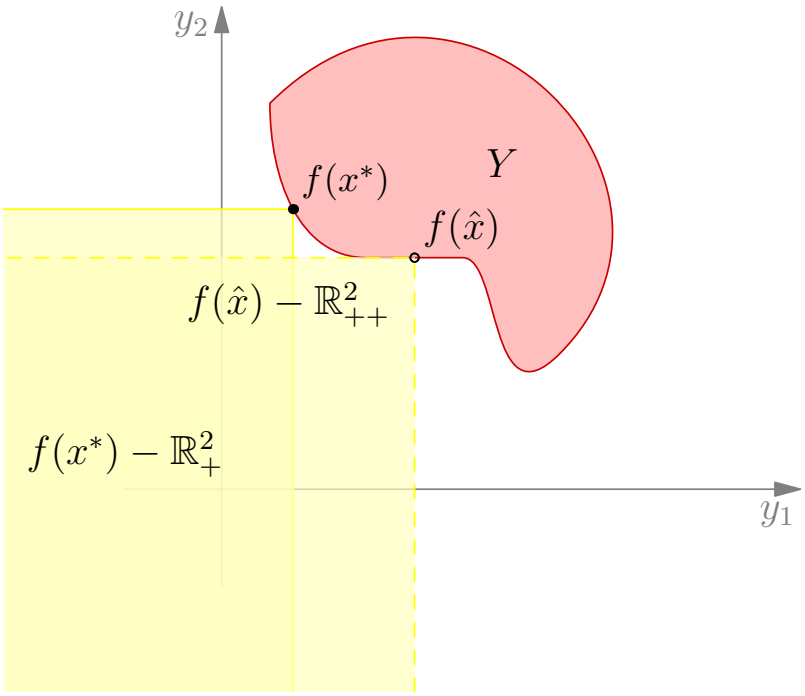

Figure 1.1: Illustration of the Remark

Remark 1.2 *We have:*

- $Y_N \subset Y_{wN}$, *and* $X_{se} \subset X_e \subset X_{we}$.
- $x^* \in X_{se} \iff x^* \in X_e$ *and* $|\{x \in X : f(x) = f(x^*)\}| = 1$.

Definition 1.5 ***(Ideal point)***[33] *The point*

$$z^I = \left(\min_{x \in X} f_1(x), \min_{x \in X} f_2(x), \ldots, \min_{x \in X} f_q(x)\right)$$

It is called the ***ideal point*** *of the multicriteria optimization problem. It represents the optimal attainable values for each objective function, assuming that they can be optimized separately.*

Definition 1.6 ***(Nadir point)***[33] *The point*

$$z^N = \left(\max_{x \in X_e} f_1(x), \max_{x \in X_e} f_2(x), \ldots, \max_{x \in X_e} f_q(x) \right)$$

It is called the ***nadir point*** *of the multicriteria optimization problem. It signifies the most unfavorable objective values among all solutions on the Pareto front.*

The ideal and nadir points for a nonconvex problem are illustrated in Figure 1.2.

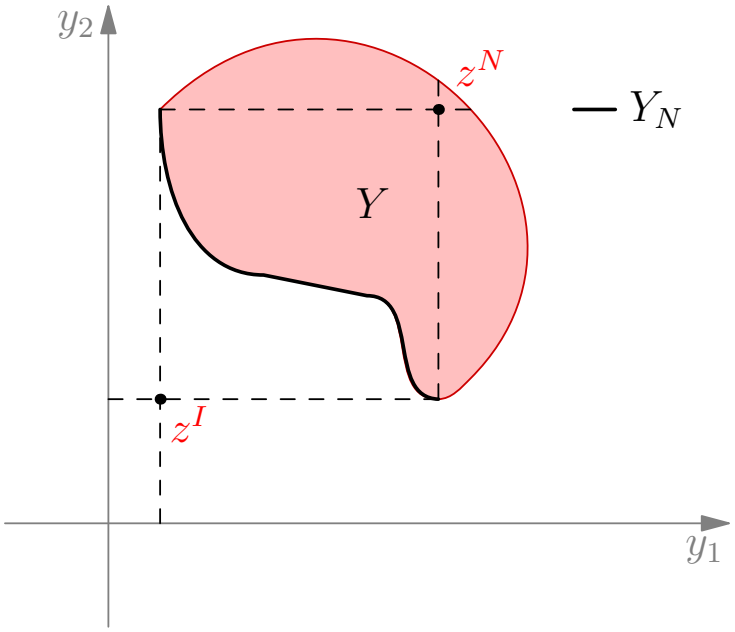

Figure 1.2: Graphical representation of Nadir and Ideal points

Proposition 1.1 *We have* $Y_N = \left(Y + R_+^q \right)_N$.

Proof: See [33] for details.

Proposition 1.2 *Let* $Y \subset \mathbb{R}^q$. *We have* $Y_N \subset \delta Y$, *where* δY *is the boundary of* Y.

Proof: See [33] for details.

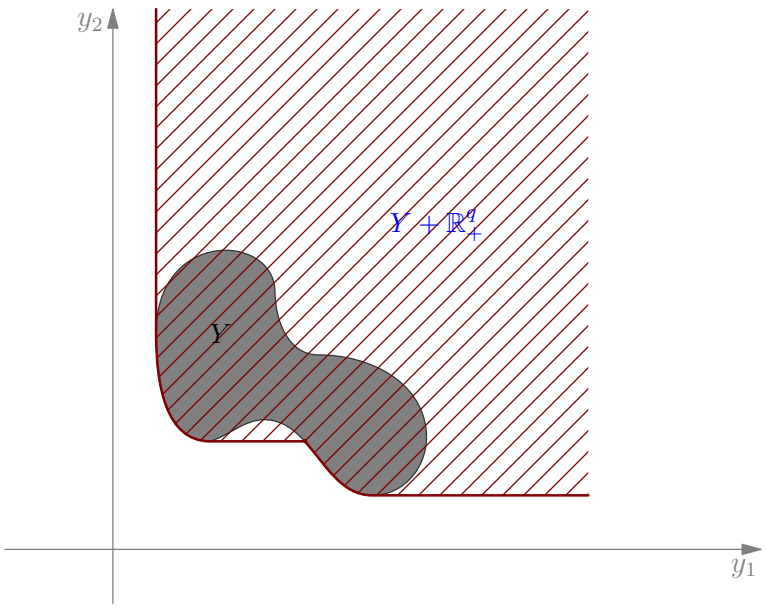

Figure 1.3: Graphical representation of $\left(Y + R^q_+\right)_N$

Proposition 1.3 *The following properties are true:*

1. $(Y_1 + Y_2)_N \subset Y_{1N} + Y_{2N}$,
2. $(\lambda Y)_N = \lambda Y_N$ *for all* $\lambda \in \mathbb{R}$.

Definition 1.7 ***(Properly efficient)*** *A feasible solution x^* is called properly efficient if it satisfies the following conditions:*

- *x^* is an efficient solution.*
- *There exists a real number $M > 0$ such that, for any $x \in X$ and for all indices i satisfying $f_i(x) < f_i(x^*)$, there exists an index j such that $f_j(x^*) < f_j(x)$, and the following inequality holds:*

$$\frac{f_i(x^*) - f_i(x)}{f_j(x) - f_j(x^*)} \leq M.$$

The corresponding point $y^* = f(x^*)$ in the objective space is called properly nondominated.

1.3 Scalarization techniques

In multicriteria optimization, scalarization methods transform vector-valued objectives into a single scalar objective, thereby enabling the application of standard op-

timization techniques. This section overviews several commonly used scalarization methods, highlighting their principles and applications in solving multicriteria optimization problems.

1.3.1 Weighted sum method

The weighted sum approach is among the most straightforward and widely used scalarization techniques. In this approach, the transformation combines multiple objectives into a single objective function using weighted coefficients, as detailed below:

Let

$$\Sigma_q = \left\{\lambda \in \mathbb{R}^q_+ : \sum_{i=1}^{q} \lambda_i = 1\right\}, \quad \Sigma_q^+ = \left\{\lambda \in \mathbb{R}^q_{++} : \sum_{i=1}^{q} \lambda_i = 1\right\}.$$

The weighted sum problem is defined as follows:

$$(WS_\lambda) \quad \begin{cases} \text{Minimize } \displaystyle\sum_{i=1}^{q} \lambda_i f_i(x) \\ s.t. \\ x \in X, \end{cases}$$

where $\lambda \in \Sigma_q$ denotes the vector of scalar weights.

The following propositions, as outlined in Ehrgott [33], demonstrate the connection between the multicriteria optimization problem (MP) and the weighted sum (WS_λ) problem.

Proposition 1.4 *[33, Proposition 3.9] Let x^* be an optimal solution of (WS_λ). We have:*

- *If $\lambda \in \Sigma_q$, then x^* is a weakly efficient solution for (MP).*
- *If $\lambda \in \Sigma_q^+$, then x^* is an efficient solution for (MP).*

A significant limitation of the weighted sum method arises when applied to non-convex multicriteria optimization problems: it fails to generate the complete Pareto front. Specifically, the supporting hyperplanes created by weighted sums only intersect the convex hull of the admissible set, neglecting non-convex regions. This limitation is illustrated in Figure 1.4, where certain nondominated points, such as $\bar{y}$, remain inaccessible regardless of the weight choice.

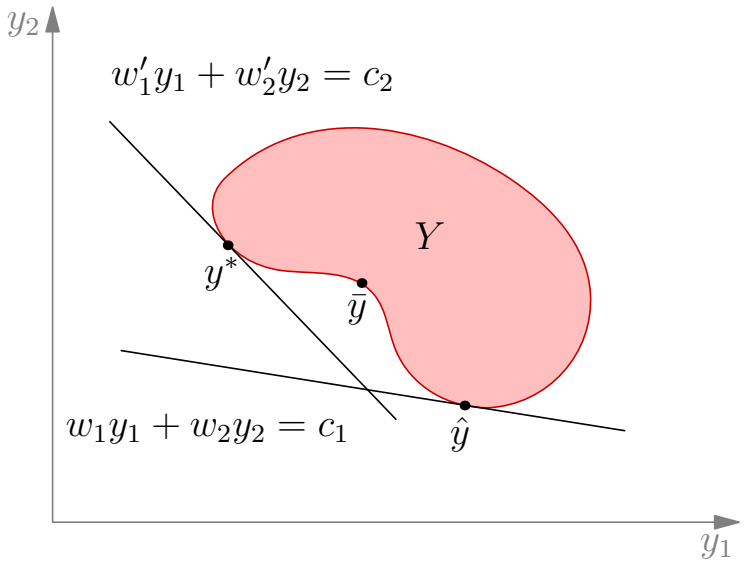

Figure 1.4: Weighted sum method for non-convex problem

Proposition 1.5 *[33, Proposition 3.10] Let f_k be convex functions for $k = 1,\dots,q$, and let X be a convex set. A point x^* is considered weakly efficient if and only if there exists a vector $\lambda \in \Sigma_q$ such that x^* is an optimal solution of the problem (WS_λ). In this case, all weakly efficient points can be obtained using the weighted sum method.*

The weighted sum method is often inadequate for non-convex multicriteria problems, as it cannot capture the nonlinear and potentially discontinuous nature of the Pareto front. In contrast, the ε-constraint approach offers a more robust solution by handling each objective separately and treating others as constraints with bounds, enabling exploration of non-convex regions in the Pareto front.

1.3.2 ε-constraint method

The ε-constraint method is one of the most well-known and widely used techniques for solving multicriteria optimization problems. The core idea of this method is to convert a multicriteria problem (MP) into a series of single-objective optimization problems by selecting one objective to optimize and treating the remaining objectives as constraints with thresholds ε_k.

The ε-constraint problem, for a given vector $\varepsilon \in \mathbb{R}^{q-1}$, is defined as follows:

$$(EC_j(\varepsilon)) \quad \begin{cases} \underset{x \in X}{\text{Minimize}}\ f_j(x) \\ s.t. \\ f_k(x) \leq \varepsilon_k, \quad k = 1, \cdots, q,\ k \neq j. \end{cases}$$

Proposition 1.6 *[33, Proposition 4.3] For some j, let x^* be an optimal solution of $(EC_j(\varepsilon))$. Then, x^* is weakly efficient for the (MP).*

Figure 1.5 illustrates the minimization of the objective function f_1 subject to different upper-bound values ε imposed on the objective function f_2. The restrictive upper bound ε_1 leads to an empty feasible region. In contrast, ε_4 imposes negligible restrictions, yielding the Pareto-optimal solution $y^{(4)}$. Similarly, ε_3 produces the solution $y^{(3)}$, while ε_2 corresponds to the optimal solution $y^{(2)}$.

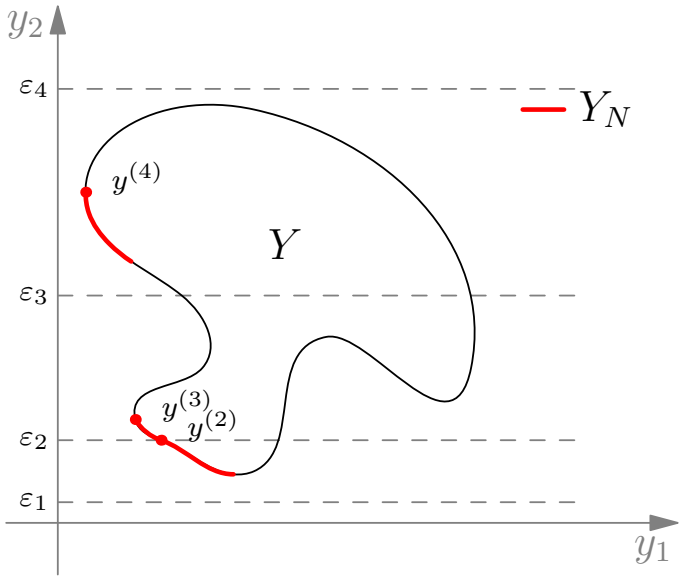

Figure 1.5: Upper bound variations in the ε-constraint method

Proposition 1.7 *[33, Proposition 4.4] For some j, let x^* be a unique optimal solution of $(EC_j(\varepsilon))$. Then, x^* is strictly efficient for the (MP).*

Theorem 1.1 *[33, Theorem4.5] The feasible solution x^* is efficient for (MP) if and only if there exists an $\bar{\varepsilon} \in \mathbb{R}^{q-1}$ such that x^* is an optimal solution of $(EC_j(\bar{\varepsilon}))$, for all $j = 1, \ldots, q$.*

Example 1.1 *Consider the following example of a convex bi-objective optimization problem.*

$$\begin{cases} \underset{x}{Minimize} \left(x+1, x^2-4x+5\right) \\ s.t. \\ x \geq 0. \end{cases}$$

Figure 1.6 shows the Pareto front of this example.

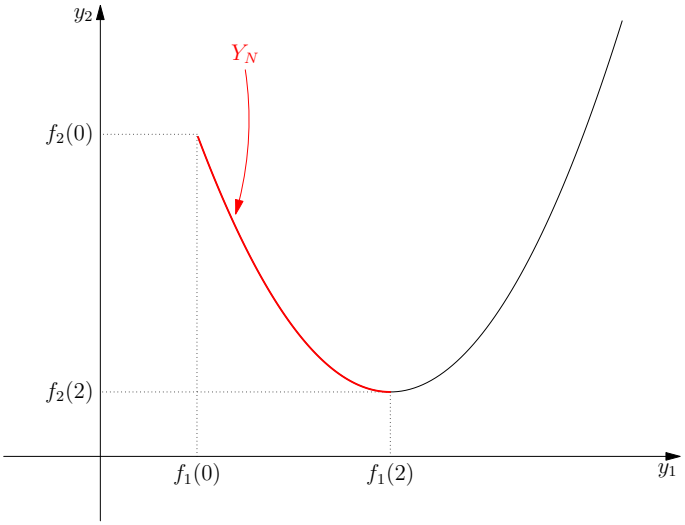

Figure 1.6: Pareto front of the previous example

When the ε-constraint method is applied, we solve the following scalarized optimization problem:

$$\begin{cases} \underset{x \geq 0}{Minimize}\ x+1 \\ s.t. \\ x \geq 0, \\ x^2-4x+5 \leq \varepsilon,\ \varepsilon \in \mathbb{R}. \end{cases}$$

According to Proposition 1.6, the ε-constraint method generates weakly Pareto optimal solutions without requiring additional assumptions. Moreover, since the considered problem is a convex bi-objective optimization problem, the ε-constraint method yields Pareto optimal solutions, as guaranteed by Proposition 1.7.

*For the numerical solution of the scalarized problem, we used the **Julia** programming language. Different values of ε were considered in the interval $[0, 20]$ with a step size of 0.1. The resulting solutions obtained for these values of ε are illustrated in Figure 1.7.*

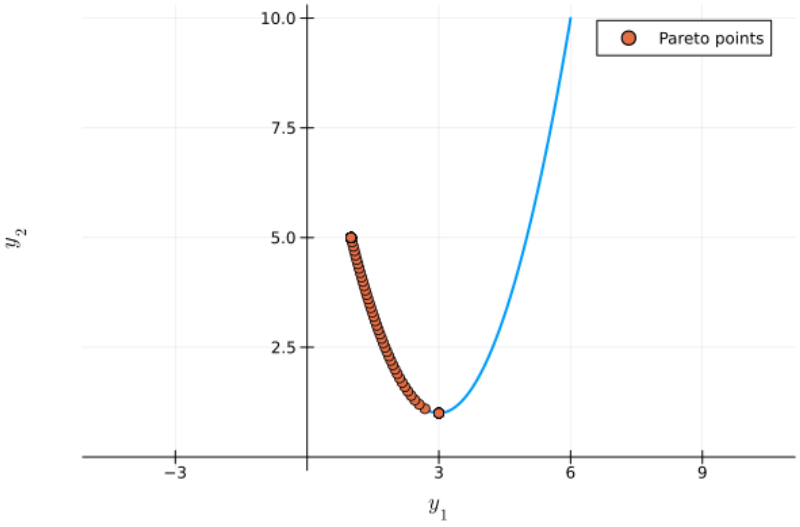

Figure 1.7: Solutions obtained from the ε-constraint method

1.3.3 Elastic constraint method

The scalarized problem $(EC_j(\varepsilon))$ can be challenging to solve in practice because of the added constraints $f_k(x) \leq \varepsilon_k$. To resolve this issue, we can **relax** these constraints by permitting violations and penalizing them in the objective function. The scalarized problem is defined as follows:

$$(EL_j(\mu)) \quad \begin{cases} \underset{x,s}{\text{Minimize}}\ f_j(x) + \sum_{k \neq j} \mu_k s_k \\ s.t. \\ f_k(x) - s_k \leq \varepsilon_k,\ k \neq j, \\ s_k \geq 0, x \in X. \end{cases}$$

where $\mu_k \geq 0$.

Proposition 1.8 *[33, Proposition 4.8] Let $\mu \geq 0$ and (x^*, s^*) be an optimal solution of $(EL_j(\mu))$. Then, x^* is weakly efficient for the (MP).*

Proposition 1.9 *[33, Proposition 4.9] If x^* is the unique optimal solution of $(EL_j(\mu))$. Then x^* is a strictly efficient solution of the (MP).*

Using Theorem 1.1, we derive the following corollary:

Corollary 1.1 *[33, Corollary 4.11] Let x^* be an efficient solution of (MP). Then there exist $\varepsilon, \mu \geq 0$ and s^* such that (x^*, s^*) is an optimal solution of $(EL_j(\mu))$, for all $j = 1, \ldots, q$.*

1.3.4 Benson's method

The Benson scalarization method, proposed by Benson in 1978 [14], is a commonly employed scalarization technique. The process begins with an initial feasible solution, x_0. If this solution is not efficient, the method seeks a dominating solution. This is achieved by using nonnegative deviation variables $L_k = f_k(x_0) - f_k(x)$ and maximizing their sum. Figure 1.8 illustrates this principle in the objective space. The problem is formulated as follows:

$$(BS(x_0)) \quad \begin{cases} \underset{x,L}{\text{Maximize}} \sum_{k=1}^{q} L_k \\ s.t. \\ f_k(x_0) - L_k - f_k(x) = 0, \ k = 1, \ldots, q, \\ L \geq 0, \ x \in X. \end{cases}$$

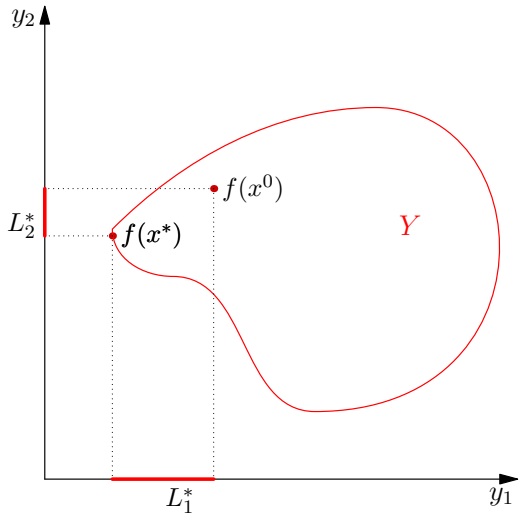

Figure 1.8: Illustration of Benson's method

Theorem 1.2 *[33, Theorem 4.14] The point $x_0 \in X$ is an efficient solution if and only if the optimal objective value of $(BS(x_0))$ is 0.*

Example 1.2 *[33] Consider the following multicriteria optimization problem*

$$\begin{cases} \underset{x}{\text{Minimize}} \left(x^2 - 4, (x-1)^4\right) \\ s.t. \\ -x - 100 \leq 0. \end{cases}$$

Benson's problem $(BS(x_0))$ *takes the form*

$$\begin{cases} \underset{x,l_1,l_2}{Maximize}\ l_1 + l_2 \\ s.t. \\ -x - 100 \leq 0, \\ (x_0)^2 - 4 - l_1 - x^2 + 4 = 0, \\ (x_0 - 1)^4 - l_2 - (x-1)^4 = 0, \\ l_1, l_2 \geq 0. \end{cases}$$

When we take $x_0 = 0$, *we obtain*

$$\begin{cases} \underset{x,l_1,l_2}{Maximize}\ l_1 + l_2 \\ s.t. \\ -x - 100 \leq 0, \\ -l_1 - x^2 = 0, \\ 1 - l_2 - (x-1)^4 = 0, \\ l1, l_2 \geq 0. \end{cases}$$

From the second constraint and the nonnegativity of l_1, *we deduce that* $x = l_1 = 0$. *Then, from the remaining constraints, it follows that* $l_2 = 0$. *Therefore, the objective value of the problem is equal to* 0. *By Theorem 1.2, we have* $x_0 = 0 \in X_e$.

Proposition 1.10 *[33, Proposition 4.15] If* (x^*, L^*) *is an optimal solution of the problem* $(BS(x_0))$, *then* x^* *is efficient for (MP) .*

Chapter 2

Bilevel optimization

2.1 Introduction

Bilevel programming problem (BP) is a complex and challenging optimization problem in which a constraint in the upper-level problem defines a lower-level optimization problem, known as the follower problem. The decision-making process in this structure is hierarchical, with two optimization levels: the upper-level problem (the leader) and the lower-level problem (the follower). Solving the leader problem requires understanding the follower's ideal response to each proposed solution. Bilevel programming problems often arise in domains such as game theory, engineering design, and economics, where decisions at different levels are interrelated. Linear bilevel programming is a unique category in which both levels have linear objective functions and constraints. Despite its linear characteristics, linear bilevel programming involves complex interactions between the two levels, necessitating sophisticated techniques to address the interdependencies between the leader's decisions and the follower's responses. This chapter will explain the mathematical formulation of bilevel programming problems, outlining the structure and relationships between the upper and lower levels. We will also present an example to illustrate the complexity of these problems and the challenges in finding the optimal solution. In addition, we will discuss commonly used solution methods and modern approaches proposed by researchers to address the challenges associated with bilevel programming.

2.2 Hierarchical leader-follower structure

Bilevel programming can be interpreted as a Stackelberg game involving two main players: the leader at the upper level and the follower at the lower level.

The variable $x \in X$ represents the leader's decision, where X denotes the leader's set of admissible strategies, and the follower's decision variable is y. The leader's objective function is denoted by $F(x, y)$, and the follower's objective function is denoted by $f(x, y)$. The follower's feasible set, which depends on the leader's decision, is defined as

$$FS(x) = \{y \in \mathbb{R}^m \mid g(x, y) \leq 0\}.$$

For a given leader decision x, the follower chooses a decision $y \in FS(x)$ that optimizes $f(x, y)$. Consequently, the leader's decision influences the follower's response. Regarding solutions to the lower-level problem, two main scenarios may arise: either a unique optimal solution exists, or multiple optimal solutions exist.

2.2.1 Unique lower-level solution

The lower-level problem admits a unique optimal solution for each decision made by the leader. This can be expressed mathematically as

$$\underset{y \in FS(x)}{\operatorname{argmin}} f(x, y) = \{y(x)\},$$

where $y(x)$ represents the unique solution of the lower-level problem for a given x. Consequently, the leader's problem reduces to

$$\begin{cases} \underset{x}{\text{Minimize}}\ F(x, y(x)) \\ s.t. \\ x \in X, \end{cases}$$

which is a single-level optimization problem over x.

2.2.2 Multiple lower-level solutions

When the optimal solution of the lower-level problem is not unique but instead forms a set, denoted by $\Psi(x)$, this indicates that, for a given x, there exist multiple optimal so-

lutions $y_1, y_2, \ldots, y_k \in \Psi(x)$. In this case, the values of the lower-level objective function are identical for all such solutions, that is,

$$f(x, y_1) = f(x, y_2) = \cdots = f(x, y_k).$$

However, the corresponding values of the upper-level objective function differ, i.e.,

$$F(x, y_1) \neq F(x, y_2) \neq \cdots \neq F(x, y_k).$$

In the presence of multiple lower-level solutions, two principal situations concerning cooperation between the leader and the follower are typically considered.

1. **Optimistic case (Cooperation):**

In this scenario, it is assumed that the follower will select the solution most favorable to the leader. That is, the follower cooperates with the leader by choosing a decision $y \in \Psi(x)$ that minimizes $F(x, y)$ among all lower-level optimal solutions. The leader's problem becomes as follows:

$$(P_o) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x, y) \\ \text{s.t.} \\ x \in X, \\ y \in \Psi(x). \end{cases}$$

Consider also the following problem:

$$(P_o)' \quad \begin{cases} \underset{x}{\text{Minimize}}\ \varphi_o(x) \\ \text{s.t.} \\ x \in X, \end{cases}$$

where

$$\varphi_o(x) = \inf_{y \in \Psi(x)} F(x, y).$$

The two problems (P_o) and $(P_o)'$ are closely related with respect to their global optimal solutions. Regarding local optimality, if $\bar{x}$ is a local optimal solution of $(P_o)'$ and $\bar{y} \in \Psi(\bar{x})$, then the pair $(\bar{x}, \bar{y})$ is a local optimal solution of (P_o), however, the converse implication does not hold (see [32] for more details).

2. **Pessimistic case (Non-cooperation):**

This situation illustrates an absence of cooperation, in which the follower chooses the value of y that maximizes $F(x,y)$ among all feasible $y \in \Psi(x)$. The leader's problem then becomes as follows:

$$(P_p) \quad \begin{cases} \underset{x}{\text{Minimize}}\ \varphi_p(x) \\ \text{s.t.} \\ x \in X, \end{cases}$$

where

$$\varphi_p(x) = \sup_{y \in \Psi(x)} F(x,y).$$

Throughout this work, the optimistic bilevel formulation is adopted.

2.3 Linear bilevel programming problems (LBP)

Linear Bilevel Programming (LBP) refers to a specific class of bilevel programming problems characterized by the linearity of all functions involved, including the objective functions of both the leader and the follower, as well as their corresponding constraint functions. LBP has garnered significant interest from researchers. A multitude of studies have focused on devising solution methodologies and examining the intricacy of these issues (see [6], [7],[39], [67], [73]).

2.3.1 Mathematical formulation of LBP

The general formulation of the linear bilevel programming problem is given by

$$(LBP) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) = a^\top x + b^\top y \\ \text{s.t.} \\ A_1 x + B_1 y \leq b_1, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ denotes the set of optimal solutions to the lower-level problem

$$(P_x) \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) = d^\top y \\ \text{s.t.} \\ A_2x + B_2y \leq b_2, \end{cases}$$

where $a \in \mathbb{R}^n$, $b, d \in \mathbb{R}^m$, $b_1 \in \mathbb{R}^{p_1}$, $b_2 \in \mathbb{R}^{p_2}$, $A_1 \in \mathbb{R}^{p_1 \times n}$, $A_2 \in \mathbb{R}^{p_2 \times m}$, $B_1 \in \mathbb{R}^{p_1 \times m}$, $B_2 \in \mathbb{R}^{p_2 \times m}$, $F, f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ are the objective functions of the upper and lower levels, respectively, and the decision variables of the upper and lower levels are represented by x and y, respectively.

1. The feasible set of the lower-level problem (P_x) for a given x is represented by:

$$FS(x) = \{y \in \mathbb{R}^m \mid A_2x + B_2y \leq b_2\}.$$

2. The feasible region of the bilevel problem (LBP) is described as follows:

$$FS = \{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m \mid A_1x + B_1y \leq b_1,\ A_2x + B_2y \leq b_2\}.$$

3. The rational reaction set of the lower level, for each fixed x, is defined as:

$$RS(x) = \operatorname{argmin}\left\{f(x,y) = d^\top y,\ y \in FS(x)\right\}.$$

4. The inducible region of problem (LBP) is defined as:

$$IR = \{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m \mid (x,y) \in FS,\ y \in RS(x)\}.$$

We provide an example from [26] illustrating that, although all functions in the bilevel programming problem are linear, the problem remains challenging to solve. This complexity is due to the non-convex nature of the problem, and the feasible region may not be connected. The following example illustrates this difficulty:

Example 2.1 *Let us take this linear bilevel programming problem:*

$$\begin{cases} \underset{(x,y)}{\textit{Minimize}}\ F(x,y) = x + 3y \\ \textit{s.t.} \\ 1 \leq x \leq 6, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ *represents the set of optimal solutions to the lower-level problem*

$$\begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) = -y \\ s.t. \\ x + y \leq 8, \\ x + 4y \geq 8, \\ x + 2y \leq 13. \end{cases}$$

The feasible set $FS(x)$ *of the lower-level problem for any fixed* x *is given by:*

$$FS(x) = \{y \in \mathbb{R} \mid x + y \leq 8,\ x + 4y \geq 8,\ x + 2y \leq 13\}.$$

The constraint region FS *is defined as:*

$$FS = \{(x,y) \in \mathbb{R} \times \mathbb{R} \mid x + y \leq 8,\ x + 4y \geq 8,\ x + 2y \leq 13,\ 1 \leq x \leq 6\}.$$

Given that the objective of the lower level is to minimize $-y$*, for each value of* $1 \leq x \leq 6$*:*

$$y(x) = \begin{cases} (13 - x)/2 & for \quad 1 \leq x \leq 3 \\ 8 - x & for \quad 3 \leq x \leq 6 \end{cases}$$

The objective function of the upper level must be minimized within the following inducible region IR:

$$IR = \{(x, 6.5 - 0.5x) : 1 \leq x \leq 3\} \cup \{(x, 8 - x) : 3 \leq x \leq 6\}.$$

The constraint region FS for this problem can be seen in figure 2.1. This figure illustrates the region where optimal solutions can be found at both the upper and lower levels. Furthermore, the figure shows that the inducible region is non-convex despite all linear functions.

From Figure 2.1, we conclude that the optimal solution is at point $D(6,2)$*, with an optimal objective value of* 12*.*

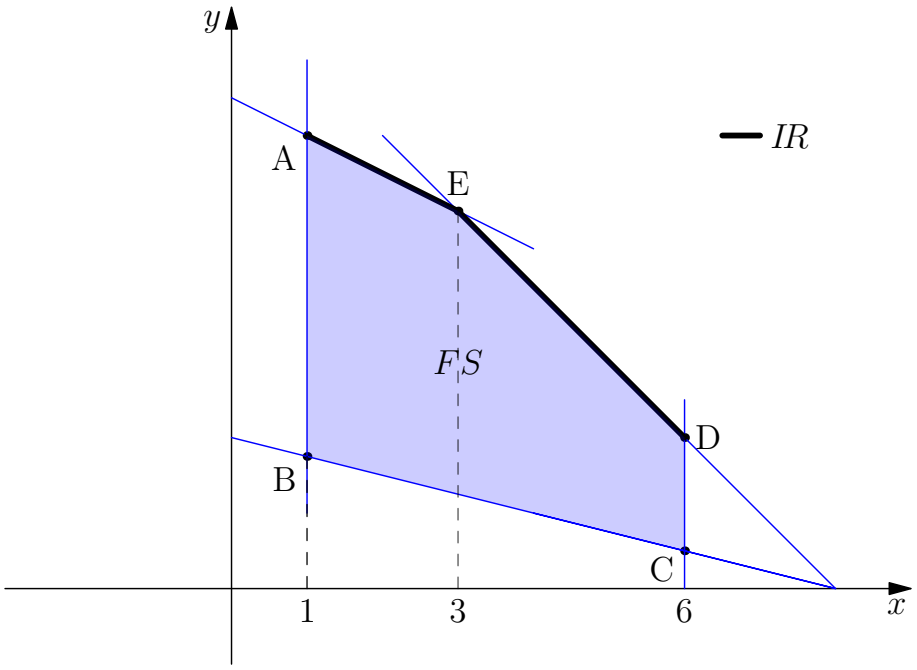

Figure 2.1: Illustration of the feasible region for the linear bilevel programming problem

Adding a constraint to the upper-level problem that depends on the lower-level optimal solution increases the complexity, as the problem may become disconnected. Below is the previous example after the modification.

$$\begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) = x + 3y \\ \text{s.t.} \\ 0 \le x \le 8,\ y \le 5, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ *represents the set of optimal solutions of the lower-level*

$$\begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) = -y \\ \text{s.t.} \\ x + y \le 8,\ x + 4y \ge 8, \\ x + 2y \le 13,\ -7x + 2y \le 0. \end{cases}$$

Figure 2.2 depicts the feasible set for this example.

As a result, linear bilevel programming problems often exhibit nonconvexity and nondifferentiability, and may possess a disconnected feasible region, making them particularly challenging to solve. Nonconvexity leads to multiple local optima, while the

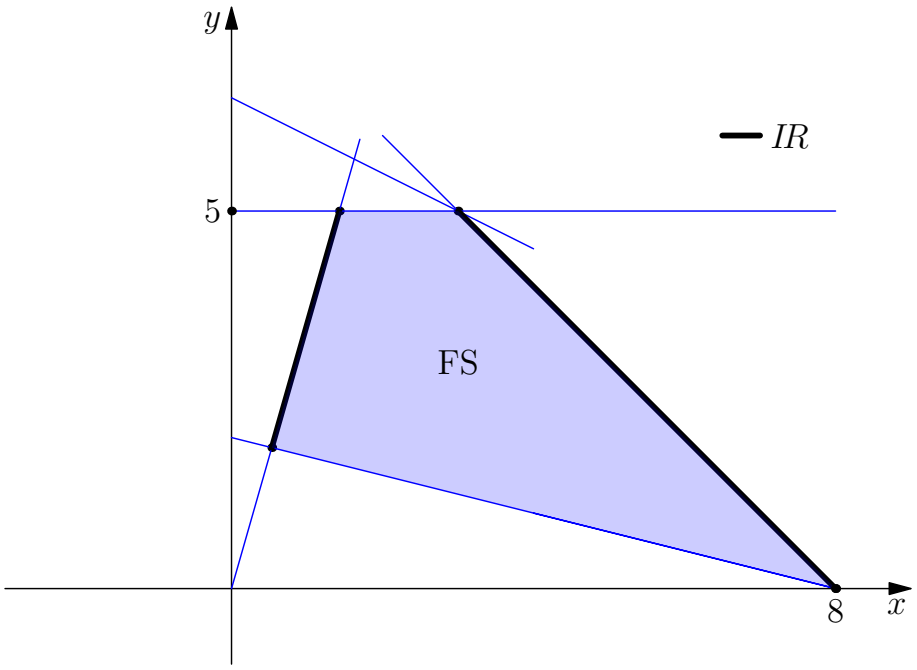

Figure 2.2: Illustration of the linear bilevel programming problem with disconnected set

lack of differentiability prevents the application of standard gradient-based optimization methods. Furthermore, the presence of disconnected regions in the feasible set complicates the solution process, requiring specialized algorithms that can identify optimal solutions across multiple disjoint subregions.

2.4 Nonlinear bilevel programming problems (NBP)

Unlike linear bilevel programming problems, nonlinear bilevel programming problems (NBP) have nonlinear objective functions and/or nonlinear constraints at both the upper and lower levels. This nonlinearity can further complicate the situation, making it more challenging to solve. Below, we provide the mathematical formulation of this problem.

2.4.1 Mathematical formulation of NBP

The general formulation of the nonlinear bilevel programming problem (NBP) is given as follows:

$$(NBP) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ G(x,y) \leq 0, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ represents the set of optimal solutions to the lower-level problem

$$(P_x) \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) \\ \text{s.t.} \\ g(x,y) \leq 0, \end{cases}$$

Where $F, f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ are the objective functions of the upper and lower levels, respectively, and $G : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_1}$, $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_2}$ are the constraint functions of the upper and lower levels.

1. The feasible set of the lower-level problem (P_x) for a given x is represented by:

$$FS(x) = \{y \in \mathbb{R}^m \mid g(x,y) \leq 0\}.$$

2. The feasible region of the bilevel problem (NBP) is described as follows:

$$FS = \{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m \mid G(x,y) \leq 0,\ g(x,y) \leq 0\}.$$

3. The rational reaction set of the lower level, for each fixed x, is defined as:

$$RS(x) = \text{argmin}\ \{f(x,y),\ y \in FS(x)\}.$$

4. The inducible region of problem (NBP) is defined as:

$$IR = \{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m \mid (x,y) \in FS,\ y \in RS(x)\}.$$

Example 2.2 *Consider the following nonlinear bilevel programming problem:*

$$\begin{cases} \underset{(x,y)}{\textit{Minimize}}\ F(x,y) = x^2 + y^2 \\ \textit{s.t.} \\ -1 \leq x \leq 1, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ *represents the set of optimal solutions of the lower-level problem*

$$\begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) = -xy \\ s.t. \\ 0 \leq y \leq 1. \end{cases}$$

The set of optimal solutions of the lower-level problem is given by:

$$\Psi(x) = \begin{cases} [0,1], & \text{if} \quad x = 0, \\ \{1\}, & \text{if} \quad x > 0, \\ \{0\}, & \text{if} \quad x < 0. \end{cases}$$

The situation in both the optimistic and pessimistic scenarios of this example is as follows:

$$\varphi_o(x) = \begin{cases} x^2 & \text{if } x \leq 0, \\ x^2 + 1 & \text{if } x > 0. \end{cases} \qquad \varphi_p(x) = \begin{cases} x^2 & \text{if } x < 0, \\ x^2 + 1 & \text{if } x \geq 0. \end{cases}$$

In the optimistic case, an optimal solution exists at $x = 0$*, with* y *taking any value in the interval* $[0,1]$*, whereas under the pessimistic formulation, no solution exists (see Figure 2.3).*

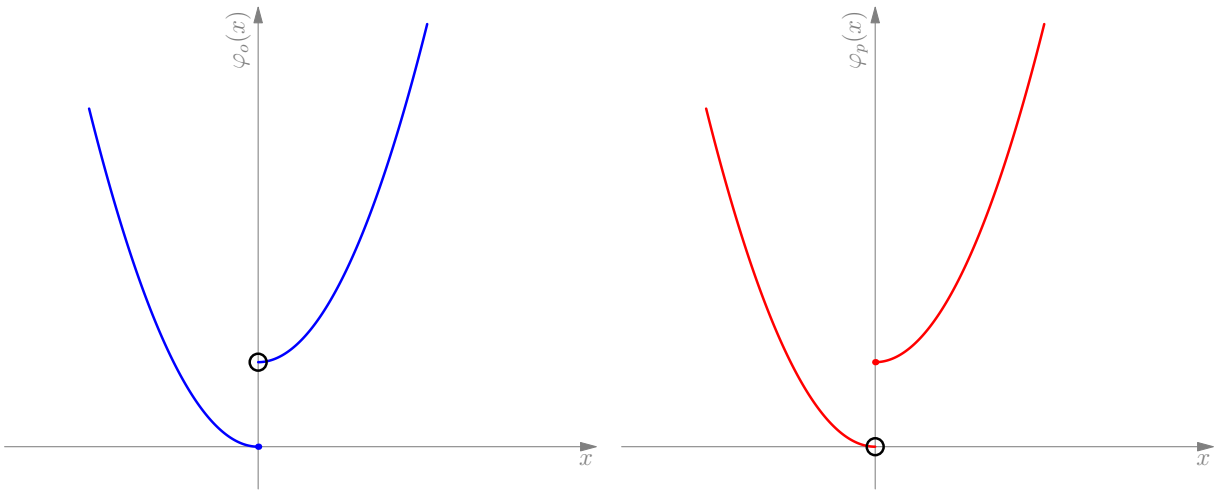

Figure 2.3: Graphical representation of the optimistic and pessimistic set-valued mappings

2.4.2 Concepts of the optimal solution in bilevel programming

Consider the following bilevel programming problem:

$$(BPP) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ x \in X, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ represents the set of optimal solutions of the lower-level problem.

$$(P_x') \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) \\ \text{s.t.} \\ g(x,y) \leq 0, \\ h(x,y) = 0. \end{cases}$$

where $F, f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, and $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_2}$, $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_3}$, $X \subseteq \mathbb{R}^n$ the problem excludes coupling upper-level constraints. We consider the feasible set of the upper-level problem:

$$X = \{x \in \mathbb{R}^n \mid G(x) \leq 0\}.$$

where $G : \mathbb{R}^n \to \mathbb{R}^{p_1}$. The reason for this exclusion is explained in Example 2.1.

The following regularity conditions are used in the definitions below.

(A_1): We say that the Mangasarian-Fromovitz Constraint Qualification (MFCQ) holds at a point (x^*, y^*) if:

- There exists a vector $d \in \mathbb{R}^m$ such that:

$$\nabla_y g_i(x^*, y^*)^\top d < 0 \quad \text{for all} \quad i \quad \text{where} \quad g_i(x^*, y^*) = 0,$$

$$\nabla_y h_j(x^*, y^*)^\top d = 0 \quad \text{for all} \quad j.$$

- The gradients of the equality constraints $\nabla_y h_j(x^*, y^*)$ are linearly independent.

(A_2): The set $\{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m : g(x,y) \leq 0,\ h(x,y) = 0\}$ is both non-empty and compact.

We start by defining the concept of an optimal solution. We begin with the simplest scenario, in which the optimal solution at the lower level is uniquely determined by all the parameter values.

Definition 2.1 *Let the solution of the lower-level problem be unique. A point (x^*, y^*) is a local optimal solution of problem (BPP) if $x^* \in X$, $y^* = \Psi(x^*)$, and there exists an open neighborhood $B_\varepsilon(x^*)$ with $\varepsilon > 0$ such that*

$$F(x^*, y^*) \leq F(x, \Psi(x)) \quad \text{for all } x \in X \cap B_\varepsilon(x^*).$$

Theorem 2.1 *[26, Theorem 5.1] Assume that conditions (A_1) and (A_2) hold for every point $(x, y) \in X \times \mathbb{R}^m$, where $y \in FS(x)$. Additionally, suppose that for each $x \in X$, the lower-level problem $(P_x^{'})$ has a unique optimal solution. Under these assumptions, if the bilevel programming problem (BPP) has at least one feasible solution, then it has a global optimal solution.*

Now, consider the concept of optimal solutions, where the optimal solution at the lower level is not unique.

Definition 2.2 *A pair (x^*, y^*) is said to be a local optimistic solution of problem (BPP) if $x^* \in X$ and $y^* \in \Psi(x^*)$ satisfying*

$$F(x^*, y^*) \leq F(x^*, y) \quad \text{for all } y \in \Psi(x^*),$$

Moreover, there exists $\varepsilon > 0$ such that

$$\varphi_o(x^*) \leq \varphi_o(x) \quad \text{for all } x \in X \cap B_\varepsilon(x^*).$$

Theorem 2.2 *[26, Theorem 5.2] Assume that the conditions (A_1) and (A_2) hold for every point $(x, y) \in X \times \mathbb{R}^m$, where $y \in FS(x)$. If there is a feasible solution, then the bilevel programming problem (P_o) has a global optimistic solution.*

Definition 2.3 *A pair (x^*, y^*) is said to be a local pessimistic solution of problem (BPP) if $x^* \in X$ and $y^* \in \Psi(x^*)$ satisfying*

$$F(x^*, y^*) \geq F(x^*, y) \quad \text{for all } y \in \Psi(x^*),$$

Moreover, there exists $\varepsilon > 0$ such that

$$\varphi_p(x^*) \leq \varphi_p(x) \quad \text{for all } x \in X \cap B_\varepsilon(x^*).$$

Theorem 2.3 *[26, Theorem 5.3] Assume that the point-to-set mapping $\Psi(\cdot)$ is lower semi-continuous at all points $x \in X$, and that condition (A_2) is satisfied. Then, a global pessimistic solution exists for problem (P_p), provided that problem (P_p) admits a feasible solution.*

The following section discusses various methods and techniques for solving bilevel programming problems, providing a comprehensive overview of both traditional and modern approaches.

2.5 Methods for solving bilevel programming problems: A comprehensive overview

2.5.1 Reformulation-Based methods

Most approaches to solving bilevel programming problems involve transforming them into single-level problems. There are two primary methods for achieving this. The first strategy replaces the lower-level problem with its KKT conditions, while the second approach is described below.

2.5.1.1 Karush-Kuhn-Tucker (KKT) Approach

The main idea of this approach is to replace the lower-level optimization problem with the Karush–Kuhn–Tucker (KKT) conditions, thereby converting the bilevel problem into a single-level problem. This transformation assumes that the functions f and g are differentiable with respect to y. The resulting formulation is given by

$$(P_{KKT}) \quad \begin{cases} \underset{(x,y,z)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ \nabla_y L(x,y,z) = 0, \\ G(x,y) \leq 0, \\ g(x,y) \leq 0, \\ z \geq 0, z^\top g(x,y) = 0, \end{cases}$$

where

$$L(x,y,z) = f(x,y) + z^\top g(x,y),$$

and z is a vector of Lagrange multipliers.

This approach requires strong assumptions to ensure that the optimal solution of the bilevel optimization problem coincides with that of the reformulated single-level problem. For a detailed discussion, see [28].

2.5.1.2 Lower-level value function reformulation (LLVFR)

The second primary method for converting the problem (NBP) into a single-level optimization problem is the lower-level value function reformulation (LLVFR). This reformulation yields to the following optimization problem:

$$(P_{LLVFR}) \quad \begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ G(x,y) \leq 0, \\ g(x,y) \leq 0, \\ f(x,y) - \varphi(x) \leq 0, \end{cases}$$

where $\varphi(x)$ denotes the optimal value function of the lower-level, defined as

$$\varphi(x) = \min_{y} \{ f(x,y) \mid g(x,y) \leq 0 \}.$$

It is important to note that the problem P_{LLVFR} is complex and challenging to solve because it involves the function φ, which is non-differentiable and non-convex. Consequently, the resulting optimization problem is also non-differentiable and non-convex, making it difficult to solve using conventional optimization methods.

Several researchers have used the LLVFR to determine optimality conditions (see, for example, [27]). However, only a limited number of recent studies have focused on numerical analysis (see, for example, [46], [74]).

2.5.2 Penalty methods

Penalty methods, widely used in nonlinear programming, were applied to bilevel programming problems by E. Aiyoshi and K. Shimizu [2], who adopted a penalty approach that converts the bilevel problem into a single-level unconstrained problem. This approach was later used by D. White and G. Anandalingam ([7], [73]), who employed a

penalty function based on the difference between the primal and dual objective values at the lower level. Additionally, Lv, Yibing, et al. [48] applied this method by converting a bilevel linear programming problem into a single-level problem using the KKT conditions. In this formulation, the complementary conditions of the lower-level problem are incorporated into the upper-level objective function as a penalty.

2.5.3 Descent method

The descent method assumes that the lower-level problem has a unique solution. Under this assumption, we can analyze the bilevel problem solely in terms of the upper-level variable x. Given a feasible point x, the purpose is to identify a feasible descent direction $d \in \mathbb{R}^n$ along which the upper-level objective function decreases. This involves calculating a new point $x+\alpha d$ for some $\alpha > 0$. One key challenge in this method is determining the gradient of the upper-level objective function at a feasible point. For more information on the descent method, see the works of Dempe (p. 194) [26].

2.5.4 Metaheuristic methods

Metaheuristic techniques are commonly used to address difficult and complex bilevel programming problems, including the following approaches:

1. **Genetic Algorithms (GA)**

Many researchers have used genetic algorithms to solve bilevel programming problems (e.g., [50], [57]). This popularity is mainly due to the efficiency of genetic algorithms in handling high-dimensional, nonlinear problems. A GA can be applied to both the upper and lower levels of a bilevel structure through selection, crossover, and mutation. In this context, the GA produces candidate solutions at both levels by evolving populations that satisfy the problem's constraints and objectives.

2. **Particle Swarm Optimization (PSO)**

The collective behavior of schools of fish and flocks of birds inspires particle swarm optimization. This method uses a swarm of particles that move through the space of possible solutions to find the best possible solution. PSO can be used to optimize the

upper-level decision variables x. At each iteration, the lower-level problem is resolved, and the upper-level particles are subsequently updated (see [37], [82]).

3. **Tabu Search (TS)**

Tabu Search is a metaheuristic that addresses complex optimization challenges by employing a local search approach that uses memory structures to avoid returning to previously investigated solutions. In bilevel programming, the upper-level solution is updated iteratively. At each iteration, the lower-level problem is addressed as a subproblem, and a memory structure known as a tabu list is utilized to prevent the search from becoming stuck in local minima (see [72]).

4. **Differential Evolution (DE)**

Differential Evolution is a population-based method specifically designed for continuous optimization problems. Like other population-based algorithms, it maintains a population of candidate solutions that evolve over time. By exploiting differences between randomly selected solutions, the algorithm generates new candidate solutions. In bilevel optimization problems, each candidate solution is evaluated by considering the corresponding lower-level problem (see [9]).

5. **Gray Wolf Optimizer (GWO)**

Gray Wolf Optimization is a population-based metaheuristic algorithm inspired by the natural world, specifically modeled after the behavior of gray wolves. It was presented by Mirjalili et al. (2014) [54] and is a relatively new approach that has been used sparingly in bilevel programming.

2.5.5 Other methods proposed by researchers

In this subsection, we will briefly overview some methods proposed by researchers for solving bilevel programming problems.

One such method was proposed in 2010 by M.S. Redjef and A. Anzi [64], who developed an algorithm for solving linear bilevel programming problems. The algorithm is based on the KKT conditions, which replace the lower-level problem with its KKT

optimality conditions, and is then solved using exact penalty methods employing DC (Difference of Convex) programming to find the optimal solution.

Building on this work, Le Thi Hoai An and Tran Duc Quynh proposed an algorithm in 2012 for a class of bilevel programming problems. In their approach, the lower-level objective function and constraints are linear, while the upper-level objective function is convex quadratic. The method uses the LLVFR to convert the bilevel problem into a single-level formulation, yielding a single concave constraint handled via a penalty method and reformulated as a DC program. Within the DC algorithm, the dual form of the lower-level value function is used to compute the subdifferential of the second component [63].

More recently, in 2022, Andreas Fischer and colleagues proposed a semi-smooth Newton-type method for bilevel programming problems, where upper- and lower-level constraints can be coupled. The approach uses the LLVFR to transform the bilevel problem into a single-level optimization problem, penalizing the value-function constraint and solving the resulting system via a semismooth method [34].

Chapter 3

Proposed method for solving bilevel programming problems using DC programming

3.1 Introduction

As mentioned above, in 2012, Le Thi Hoai An and Tran Duc Quynh proposed an algorithm to solve a class of bilevel programming problems in which the lower-level problem is linear, and the upper-level objective function is convex quadratic. They noted in their article that their algorithm is not generalizable and is only effective when the lower-level problem is linear. In response, we have proposed a DC programming method that is effective for a broader class of bilevel programming problems. The core idea of our work is to express the value function of a non-convex lower-level problem as the difference of convex functions. To achieve this, we introduced a regularization approach. By incorporating a penalty approach, the problem was reformulated as the difference of convex functions. This chapter introduces DC programming, focusing on DC functions and DC algorithms. In addition, we review regularization and penalty methods. We then formulate the bilevel programming problem within the DC programming setting and present the BL-DCA algorithm, a specialized application of DCA to bilevel programming. Finally, we present numerical results that demonstrate

the effectiveness of our proposed technique, which has been published in the journal Nonlinear Dynamics and Systems Theory [16].

In this chapter, we require the following definitions:

Definition 3.1 (Domain of a function) *Let $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ be a convex function. The effective domain of f, denoted by* $\mathrm{dom}(f)$, *is defined as*

$$\mathrm{dom}(f) = \{x \in \mathbb{R}^n \mid f(x) < +\infty\}.$$

Definition 3.2 (Epigraph of a function) *The epigraph of a function $f : X \to \overline{\mathbb{R}}$, where $X \subset \mathbb{R}^n$, denoted by* $\mathrm{epi}(f)$, *is defined as*

$$\mathrm{epi}(f) = \{(x, s) \in X \times \mathbb{R} \mid f(x) \leq s\}.$$

Definition 3.3 (Indicator function) *The indicator function, denoted as $\chi_A(x)$, for the set $A \subset \mathbb{R}^n$ is defined by*

$$\chi_A(x) = \begin{cases} 0, & \text{if } x \in A, \\ +\infty, & \text{if } x \notin A. \end{cases}$$

Definition 3.4 (Proper function) *A function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$ is said to be proper if it is not identically equal to $+\infty$ and never attains the value $-\infty$.*

Definition 3.5 (Subdifferential of a function) *The subdifferential of a convex function $f : X \to \mathbb{R}$ at a point $x \in X$ (where $X \subseteq \mathbb{R}^n$) is the set $\partial f(x)$ defined as*

$$\partial f(x) = \{q \in \mathbb{R}^n : f(z) - f(x) \geq \langle q, z - x\rangle \ \text{ for all } z \in X\}.$$

For a given $\epsilon \geq 0$, the ϵ-subdifferential of f at the point x, which is denoted by $\partial_\epsilon f(x)$, is defined as

$$\partial_\epsilon f(x) = \{q \in \mathbb{R}^n : f(z) - f(x) \geq \langle q, z - x\rangle - \epsilon \ \text{ for all } z \in X\}.$$

Definition 3.6 (Conjugate of a function) *Let the function $f : \mathbb{R}^n \to \overline{\mathbb{R}}$. Its conjugate function, denoted $f^* : \mathbb{R}^n \to \overline{\mathbb{R}}$, is defined by*

$$f^*(z) = \sup_{x \in \mathbb{R}^n} \{\langle z, x\rangle - f(x)\}.$$

Definition 3.7 (Lower semi-continuous function) *Let $f : X \to \mathbb{R} \cup \{+\infty\}$ and let $x_0 \in X$. The function f is said to be lower semi-continuous (l.s.c.) at x_0 if for every sequence (x_n) in X converging to x_0, we have*

$$f(x_0) \leq \liminf_{n \to \infty} f(x_n).$$

Definition 3.8 *A function f is convex if, for all $x, z \in \mathbb{R}^n$ and for all $\alpha \in [0,1]$, the following inequality holds:*

$$f(\alpha x + (1-\alpha)z) \leq \alpha f(x) + (1-\alpha) f(z).$$

Definition 3.9 (strongly convex) *A function f is said to be strongly convex with modulus $\mu > 0$ if, for all $x, z \in \mathbb{R}^n$ and for all $\alpha \in [0,1]$, the following inequality holds:*

$$f(\alpha x + (1-\alpha)z) \leq \alpha f(x) + (1-\alpha) f(z) - \frac{\mu}{2}\alpha(1-\alpha)\|x-z\|^2.$$

3.2 DC programming

DC programming is a specialized field of mathematical optimization that focuses on solving optimization problems in which the objective function f is expressed as the difference of two convex functions, g and h. This representation is called a DC decomposition of f, where g and h are the DC components. DC programming has proven extremely useful when tackling problems with non-convex objective functions, which pose considerable obstacles for standard optimization approaches.

3.2.1 DC functions

Let $C \subset \mathbb{R}^n$ be a convex set. A function f is called a **DC function** on C if it can be written as the difference of two convex functions on C; that is,

$$f(x) = g(x) - h(x), \quad \forall x \in C,$$

where g and h are convex functions defined on C.

Proposition 3.1 *Let f_i, for $i = 1, \ldots, m$, be DC functions on a convex set C. Then the following functions are also DC on C:*

- $\sum_{i=1}^{m} a_i f_i$, *where $a_i \in \mathbb{R}$ for $i = 1, \ldots, m$.*

- $\max\{f_1,\ldots,f_m\}$ *and* $\min\{f_1,\ldots,f_m\}$.
- $\prod_{i=1}^{m} f_i(x)$.
- *If f is a DC function, then the functions $f^+(x)=\max\{0,f(x)\}$ and $f^-(x)=\min\{0,f(x)\}$ are also DC functions.*

Example 3.1 *Here are some examples of DC functions:*

1. ***Inner product:*** *For $u,v\in\mathbb{R}^n$, the inner product can be written as*

$$u^\top v=\frac{1}{4}\left(\|u+v\|^2-\|u-v\|^2\right),$$

2. ***Square of the distance function:*** *Let $d:\mathbb{R}^n\to\mathbb{R}$ be the distance function defined by*

$$d(u,M)=\inf\{\|u-v\|:v\in M\}.$$

where M is any non-empty closed subset of $\mathbb{R}^n$. Then the squared distance function can be expressed as

$$\begin{aligned} d^2(u)&=\inf\left\{\|u-v\|^2,v\in M\right\}\\ &=\|u\|^2+\inf\left\{-\|u\|^2+\|u-v\|^2,v\in M\right\}\\ &=\|u\|^2+\inf\left\{\|v\|^2-2u^\top v,v\in M\right\}\\ &=\|u\|^2-\sup\left\{2u^\top v-\|v\|^2,v\in M\right\}. \end{aligned}\tag{3.1}$$

The second term in (3.1) is convex, since it is the pointwise supremum of affine functions of u.

3.2.2 DC programming and problem equivalence

In DC programming, the objective function is expressed as the difference of two convex functions. A DC programming problem can be written in the following general form:

$$(P_1)\quad\left\{\begin{array}{l}\underset{x\in X}{\text{Minimize}}\ f(x)=g(x)-h(x),\end{array}\right.$$

where g and h are convex functions, and X is a convex set.

Now consider the following two optimization problems:

$$(P_2) \quad \left\{ \underset{x \in X}{\text{Maximize}} \ f(x), \right.$$

where f is a convex function defined on the convex set X, and

$$(P_3) \quad \begin{cases} \underset{x \in X}{\text{Minimize}} \ g(x) - h(x) \\ \text{s.t.} \\ f_1(x) - f_2(x) \leq 0, \end{cases}$$

where g, h, f_1, and f_2 are convex functions defined on the convex set X.

Despite their different formulations, these three problems are equivalent. In particular, by setting $g = \chi_X$, the problem P_2 can be reformulated in the form of problem (P_1).

Moreover, problem (P_3) can be transformed into problem (P_2) by applying an exact penalization technique to the DC constraint $f_1(x) - f_2(x) \leq 0$.

3.2.3 Duality in DC programming

Let $g, h : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be convex, proper, and lower semi-continuous (l.s.c.) functions. The biconjugate of the function h is given by:

$$h^{**}(u) = \sup_{v \in \mathbb{R}^n} \{\langle u, v \rangle - h^*(v)\}.$$

Since h is convex, proper, and l.s.c., it follows that $h^{**} = h$.

The dual problem of the DC program

$$(P) \quad \left\{ \inf_{u \in \text{dom}(g)} \{g(u) - h(u)\}, \right.$$

is defined by

$$(D) \quad \left\{ \inf_{v \in \text{dom}(h^*)} \{h^*(v) - g^*(v)\}. \right.$$

Proof. By replacing h with its biconjugate expression, we obtain

$$\begin{aligned}
\inf_{u\in\mathrm{dom}(g)}\{g(u)-h(u)\} &= \inf_{u\in\mathrm{dom}(g)}\left\{g(u)-\sup_{v\in\mathrm{dom}(h^*)}\{\langle u,v\rangle-h^*(v)\}\right\}\\
&= \inf_{u\in\mathrm{dom}(g)}\left\{g(u)+\inf_{v\in\mathrm{dom}(h^*)}\{-\langle u,v\rangle+h^*(v)\}\right\}\\
&= \inf_{u\in\mathrm{dom}(g)}\inf_{v\in\mathrm{dom}(h^*)}\{g(u)+h^*(v)-\langle u,v\rangle\}\\
&= \inf_{v\in\mathrm{dom}(h^*)}\left\{h^*(v)+\inf_{u\in\mathrm{dom}(g)}\{g(u)-\langle u,v\rangle\}\right\}\\
&= \inf_{v\in\mathrm{dom}(h^*)}\left\{h^*(v)-\sup_{u\in\mathrm{dom}(g)}\{-g(u)+\langle u,v\rangle\}\right\}\\
&= \inf_{v\in\mathrm{dom}(h^*)}\{h^*(v)-g^*(v)\}.
\end{aligned}$$

■

From the above, a relationship between the solutions of the primal problem (P) and its dual (D) is established. We conclude that solving one problem leads to solving the other. Consequently, to solve (P), we can either solve (D) first or vice versa, depending on which is more straightforward to solve.

3.2.4 Optimality conditions in DC programming

Let the following DC problem be given by:

$$(P_{DC}) \quad \left\{ \underset{x\in\mathbb{R}^n}{\text{Minimize}}\ f(x)=g(x)-h(x), \right.$$

where $g,h:\mathbb{R}^n\to\overline{\mathbb{R}}$ are lower semi-continuous, proper, convex functions.

Proposition 3.2 *[43] (Necessary condition for local optimality) Let $\bar{x}$ be a local solution to (DC). Then*

$$\partial h(\bar{x})\subseteq\partial g(\bar{x}).$$

Theorem 3.1 *[43](Sufficient condition for local optimality) Suppose that x^* has a neighborhood U such that*

$$\partial h(x)\cap\partial g(x)\neq\emptyset,$$

for all $x\in U\cap dom(g)$. Then, x^* is a local minimum of $g-h$.

Theorem 3.2 *[38] A point x^* is a global minimum of (P_{DC}) if and only if*

$$\partial_\epsilon h(x^*)\subset\partial_\epsilon g(x^*),\quad\forall\epsilon>0.$$

3.2.5 DC Algorithm

The Difference of Convex Functions Algorithm (DCA) is a highly efficient and adaptive optimization algorithm for minimizing the differences of convex functions. DCA approximates a non-convex optimization problem by an iterative sequence of convex problems. Each iteration solves a convex subproblem derived from the original DC problem. This iterative process allows for gradual solution refinement and effectively handles non-convex optimization challenges. To ensure the well-definedness of the DCA algorithm, we make the following assumptions:

Assumption 1 $\emptyset \neq dom\,\partial g \subset dom\,\partial h$, *where* $\text{dom}(\partial g) = \{x \in \mathbb{R}^n \mid \partial g(x) \neq \emptyset\}$.

Assumption 2 *Each subproblem* (P_k) *has at least one optimal solution, although these solutions may not be unique.*

The implementation of the algorithm is outlined below:

Algorithm 1: DCA

Step 0: Select an initial point $x_0 \in \text{dom}\,\partial h$, choose the maximum number of iterations $itermax$, and set $k = 0$.

Step 1: for $k = 0, 1, \ldots, itermax$ **do**

1. Select $p^k \in \partial h(x^k)$, and solve the convex optimization problem

$$(P_k) \quad \left\{ \underset{x \in \mathbb{R}^n}{\text{Minimize}} \left\{ g(x) - \langle p^k, x \rangle \right\} \right.$$

To obtain a solution x^{k+1}.

2. If the convergence criterion is satisfied, terminate the process; otherwise, set $k := k + 1$.

end

We now present the following result regarding the convergence of the DCA algorithm:

Theorem 3.3 *[56] The DCA algorithm will either converge after a finite number of iterations or generate an infinite sequence of points* (x_k). *Any accumulation point of the sequence*

$\{x_k\}$ will be a critical point of the (P_{DC}) problem, provided that the sequences $\{x_k\}$ and $\{p_k\}$ are bounded and certain conditions are satisfied, such as the strong convexity of one of the functions g or h, and $\inf f > -\infty$.

3.3 Regularization method

Regularization is a fundamental approach often used in machine learning and optimization. In optimization, regularization involves adding a penalty term to the objective function to control the complexity of the solution.

Consider the following optimization problem:

$$(P) \quad \left\{ \underset{x}{\text{Minimize}}\ f(x). \right.$$

The regularized optimization problem of (P) can typically be expressed as:

$$(P_r) \quad \left\{ \underset{x}{\text{Minimize}}\ f(x) + rR(x), \right.$$

where $f(x)$ is the original objective function that you want to minimize, $R(x)$ denotes the regularisation term, which is generally a function of the solution x that imposes a penalty on complexity, and $r > 0$ is the regularization parameter that controls the intensity of the regularization.

There are three main types of regularization in optimization:

1. L_1 **Regularization**

In this case, the regularization term is expressed mathematically as:

$$R(x) = \sum_{i=1}^{n} |x_i|.$$

The regularized problem then becomes:

$$(P_{r_1}) \quad \left\{ \underset{x}{\text{Minimize}} \left(f(x) + r_1 \sum_{i=1}^{n} |x_i| \right), \right.$$

where $r_1 > 0$ is the regularization parameter.

2. L_2 **Regularization**

This regularization technique is often used in linear regression and many machine learning algorithms, and it discourages large variable values.

In this case, the regularization term is given by:

$$R(x) = \|x\|_2^2 = \sum_{i=1}^{n} x_i^2$$

The regularized problem then becomes:

$$(P_{r_2}) \quad \left\{ \underset{x}{\text{Minimize}} \left(f(x) + r_2 \|x\|_2^2 \right), \right.$$

where $r_2 > 0$ is the regularization parameter.

3. **Elastic Net Regularization**

Elastic Net integrates both L_1 and L_2 regularization techniques.

The regularized problem then becomes:

$$(P_{r_1,r_2}) \quad \left\{ \underset{x}{\text{Minimize}} \left(f(x) + r_1 \sum_{i=1}^{n} |x_i| + r_2 \|x\|_2^2 \right), \right.$$

In this context, $r_1 > 0$ and $r_2 > 0$ are the regularization parameters controlling the L_1 and L_2 regularization terms, respectively.

3.4 Penalty methods

Penalty methods are techniques used in constrained optimization to approximate the original problem by a sequence of unconstrained optimization problems. The fundamental concept is to incorporate a penalty term into the original objective function. This term penalizes any violation of the established constraints. The penalty term is often a function of the violated constraint, scaled by a penalty parameter. There are two basic types of penalty methods: **exact** and **inexact**.

Consider the following optimization problem with constraints that we are trying to solve:

$$(P') \quad \begin{cases} \underset{x}{\text{Minimize}}\ f(x) \\ \text{s.t.} \\ g_i(x) \leq 0,\ i = 1, \ldots, q. \end{cases}$$

- **Exact penalty methods** can identify the exact solution using a finite penalty parameter. They typically involve non-smooth functions, such as absolute values. The penalty term can be expressed as:

$$f(x) + \mu \sum_{i=1}^{q} \max\{0, g_i(x)\}$$

where μ is the penalty parameter.

- **Inexact penalty methods** often use smooth penalty functions; however, they require a very large penalty parameter to approximate the solution to the original constrained problem accurately. The penalty term is typically expressed as:

$$f(x) + \mu \sum_{i=1}^{q} \max\{0, g_i(x)\}^2$$

where μ is the penalty parameter.

3.5 Problem formulation

In this chapter, we will study the following class of bilevel programming problems:

$$(P_1) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ H_i(x,y) \leq 0,\ i = 1,\dots,q_1, \\ y \in \Psi(x), \end{cases}$$

where $\Psi(x)$ represents the set of optimal solutions to the lower-level problem:

$$(P_x) \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) = \langle Ax, y\rangle + \langle c, y\rangle \\ \text{s.t.} \\ h_i(y) \leq 0,\ i = 1,\dots,q_2. \end{cases}$$

Here $F, f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ denote the leader's and follower's objective functions, respectively, with $c \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$.

Let us introduce the following definitions, which will be essential for the study of problem (P_1).

1. The feasible solution set of the parameterized lower-level problem (P_x):

$$FS_0 = \{y \in \mathbb{R}^m \mid h_i(y) \leq 0,\ i = 1,\ldots,q_2\}.$$

2. The constraint region of (P_1):

$$FS = \left\{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m \mid H_i(x,y) \leq 0,\ h_j(y) \leq 0,\ i = 1,\ldots,q_1,\ j = 1,\ldots,q_2\right\}.$$

3. The rational reaction set of (P_x), corresponding to each fixed value of x:

$$R(x) = \arg\min\{\langle Ax,y\rangle + \langle c,y\rangle,\ y \in FS_0\}.$$

4. The inducible region of (P_1):

$$IR = \{(x,y) \in \mathbb{R}^n \times \mathbb{R}^m \mid (x,y) \in FS,\ y \in R(x)\}.$$

Assumption 3 *The functions F, H, and h are continuous convex.*

Assumption 4 *The feasible solution set FS_0 is assumed nonempty and compact.*

3.5.1 Complications involved in the studied problem

In our work, we employ DC algorithms to solve problem (P_1). Using the lower-level value function reformulation (LLVFR), the bilevel programming problem (P_1) is transformed into a single-level optimization problem of the form

$$\begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ H_i(x,y) \leq 0,\ i = 1,\ldots,q_1, \\ h_i(y) \leq 0,\ i = 1,\ldots,q_2, \\ \langle Ax,y\rangle + \langle c,y\rangle - V(x) \leq 0, \end{cases}$$

where

$$V(x) = \min_y\{\langle Ax,y\rangle + \langle c,y\rangle \mid h_i(y) \leq 0,\ i = 1,\ldots,q_2\}.$$

Next, an exact penalty function is applied to the last constraint, leading to the following optimization problem:

$$\begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) + \mu\left(\langle Ax, y\rangle + \langle c, y\rangle - V(x)\right) \\ \text{s.t.} \\ H_i(x,y) \leq 0,\ i = 1,\ldots,q_1, \\ h_i(y) \leq 0,\ i = 1,\ldots,q_2. \end{cases}$$

Note that the function $V(x)$ is concave since the mapping $x \mapsto \langle Ax, y\rangle + \langle c, y\rangle$ is the pointwise minimum of affine functions.

The inner product $\langle Ax, y\rangle$ admits a DC decomposition given by

$$\langle Ax, y\rangle = \frac{1}{4}\|Ax + y\|^2 - \frac{1}{4}\|Ax - y\|^2.$$

This reformulation leads to the following problem:

$$\begin{cases} \underset{(x,y)}{\text{Minimize}}\ \left(F(x,y) + \frac{\mu}{4}\|Ax + y\|^2 + \mu\langle c, y\rangle - \mu V(x)\right) - \left(\frac{\mu}{4}\|Ax - y\|^2\right) \\ \text{s.t.} \\ H_i(x,y) \leq 0, \quad i = 1,\ldots,q_1, \\ h_i(y) \leq 0, \quad i = 1,\ldots,q_2. \end{cases}$$

The above problem is a DC program of the form

$$\begin{cases} \underset{(x,y)}{\text{Minimize}}\ g(x,y) - h(x,y) \\ \text{s.t.} \\ (x,y) \in FS. \end{cases}$$

It is preferable that g be differentiable, as this ensures that the convex subproblem (P_k) is a smooth convex optimization problem. In contrast, the function h is not required to be differentiable, since in the DC Algorithm (DCA) only a subgradient of h is needed. To solve this problem, we adopt a regularization approach to incorporate the function $V(x)$ into h.

On the other hand, Le Thi et al. in [63] propose a DC algorithm for solving a class of bilevel programming problems where the lower-level problem is linear, and they note that their method is effective only in this linear setting. In contrast, we introduce new techniques to address a broader class of bilevel programming problems, particularly

those in which the lower-level problem is nonconvex with respect to both variables (x, y). As discussed above, the DC algorithm and the approach proposed in [63] cannot be directly applied to the problem. To address this challenge, we will use a regularization approach. Specifically, we will express the lower-level value function as a DC function with penalization, enabling DC programming. In the next subsection, we provide a thorough explanation of our approach.

3.5.2 Regularized bilevel optimization model

The first step is to replace the original lower-level problem (P_x) with the regularized problem (P_x^r), defined as

$$(P_x^r) \quad \begin{cases} \underset{y}{\text{Minimize}} \ f(x,y) = \langle Ax, y\rangle + \langle c, y\rangle + \dfrac{r}{2}\|y\|^2 \\ \text{s.t.} \\ h_i(y) \leq 0, \quad i = 1, \ldots, q_2, \end{cases}$$

where $r > 0$ is the regularization parameter.

In bilevel programming, regularization is often used under certain conditions to guarantee the uniqueness of solutions to the lower-level problem. This is particularly important in lower-level solution mapping settings that may be non-unique or discontinuous. Further theoretical foundations can be found in [25] and references therein. By applying the LLVFR to problems (P_1) and (P_x^r), we obtain the following equivalent problem (P_r):

$$(P_r) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}} \ F(x,y) \\ \text{s.t.} \\ H_i(x,y) \leq 0, \ i = 1, \ldots, q_1, \\ h_i(y) \leq 0, \ i = 1, \ldots, q_2, \\ \langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 - V^r(x) \leq 0, \end{cases}$$

where the value function associated with the regularized lower-level problem (P_x^r) is defined by

$$V^r(x) = \min_y \left\{ \langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 \mid h_i(y) \leq 0, \ i = 1, \ldots, q_2 \right\}.$$

As the regularization parameter r decreases and approaches zero, the influence of the penalty term $\frac{r}{2}\|y\|^2$ diminishes. Consequently, the solution $\bar{y}_r$ of the regularized prob-

lem converges to the solution $\bar{y}$ of the original problem, and the corresponding objective value $V^r(x)$ converges to $V(x)$. For sufficiently small values of r, the regularization term $\frac{r}{2}\|\bar{y}_r\|^2$ becomes negligible, ensuring that the regularized formulation accurately approximates the original optimization problem. This convergence can be characterized as follows: for every $\varepsilon > 0$, there exists $\delta > 0$ such that, whenever $0 < r < \delta$,

- The difference between the regularized and original objective functions satisfies:

$$\langle Ax, \bar{y}_r\rangle + \langle c, \bar{y}_r\rangle + \frac{r}{2}\|\bar{y}_r\|^2 - \langle Ax, \bar{y}\rangle - \langle c, \bar{y}\rangle \leq \frac{\varepsilon}{2},$$

- The regularization term is bounded by:

$$\frac{r}{2}\|\bar{y}_r\|^2 \leq \frac{\varepsilon}{2}.$$

By combining these results, we obtain the following inequality:

$$\left|\langle Ax, \bar{y}\rangle + \langle c, \bar{y}\rangle - (\langle Ax, \bar{y}_r\rangle + \langle c, \bar{y}_r\rangle)\right| \leq \varepsilon.$$

We express $V^r(x)$ as a difference of two convex functions through regularization, enabling the application of DC programming techniques, as shown in the following proposition.

Proposition 3.3 *The function $V^r(x)$ is a DC function and admits the following DC decomposition:*

$$V^r(x) = V_1^r(x) - V_2^r(x), \quad x \in \mathbb{R}^n,$$

where

$$V_1^r(x) = \frac{r}{2}\left[d\left(-\frac{Ax+c}{r}, FS_0\right)\right]^2, \quad \textit{and} \quad V_2^r(x) = \frac{1}{2r}\|Ax+c\|^2.$$

Proof.

$$
\begin{aligned}
V^r(x) &= \min_y \left\{ \langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 \,\middle|\, y \in FS_0 \right\} \\
&= \min_y \left\{ \langle Ax + c, y\rangle + \frac{r}{2}\left\| y + \frac{Ax+c}{r} - \frac{Ax+c}{r}\right\|^2 \,\middle|\, y \in FS_0 \right\} \\
&= \min_y \left\{ \langle Ax + c, y\rangle + \frac{r}{2}\left[\left\| y + \frac{Ax+c}{r}\right\|^2 + \left\|\frac{Ax+c}{r}\right\|^2 - 2\left\langle \frac{Ax+c}{r}, y + \frac{Ax+c}{r}\right\rangle \right] \,\middle|\, y \in FS_0 \right\} \\
&= \min_y \left\{ \langle Ax + c, y\rangle + \frac{r}{2}\left\| y + \frac{Ax+c}{r}\right\|^2 + \frac{1}{2r}\|Ax+c\|^2 - \langle Ax+c, y\rangle - \frac{1}{r}\|Ax+c\|^2 \,\middle|\, y \in FS_0 \right\} \\
&= \min_y \left\{ \frac{r}{2}\left\| y + \frac{Ax+c}{r}\right\|^2 - \frac{1}{2r}\|Ax+c\|^2 \,\middle|\, y \in FS_0 \right\} \\
&= \min_y \left\{ \frac{r}{2}\left\| y + \frac{Ax+c}{r}\right\|^2 \,\middle|\, y \in FS_0 \right\} - \frac{1}{2r}\|Ax+c\|^2 \\
&= \frac{r}{2}\left[d\left(-\frac{Ax+c}{r}, FS_0\right)\right]^2 - \frac{1}{2r}\|Ax+c\|^2 \\
&= V_1^r(x) - V_2^r(x).
\end{aligned}
$$

■

Building on the preceding theoretical results and the above proposition, we formulate the following problem:

$$
(P_r) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ H_i(x,y) \le 0,\ i = 1,\dots,q_1, \\ h_i(y) \le 0,\ i = 1,\dots,q_2, \\ \langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 - \left(V_1^r(x) - V_2^r(x)\right) \le 0. \end{cases}
$$

Since the inequality

$$
\langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 - (V_1^r(x) - V_2^r(x)) \le 0
$$

originally represents an equality constraint (as discussed in [75] and [21]), and considering that $\langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 - \left(V_1^r(x) - V_2^r(x)\right) \ge 0$, we apply the exact penalty method to the last constraint to derive the following problem:

$$
(P_r^\mu) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) + \mu\left(\langle Ax, y\rangle + \langle c, y\rangle + \frac{r}{2}\|y\|^2 - (V_1^r(x) - V_2^r(x))\right) \\ \text{s.t.} \\ H_i(x,y) \le 0,\ i = 1,\dots,q_1, \\ h_i(y) \le 0,\ i = 1,\dots,q_2, \end{cases}
$$

where μ denotes a penalty parameter.

In addition, the inner product $\langle Ax, y\rangle$ can also be expressed as the difference of two convex functions. This decomposition plays an important role in simplifying the application of DC programming techniques and is given in the following manner:

$$\langle Ax, y\rangle = \frac{1}{4}\left\|Ax + y\right\|^2 - \frac{1}{4}\left\|Ax - y\right\|^2$$

Consequently, the problem presented above is equivalent to the following formulation:

$$\begin{cases} \underset{(x,y)}{\text{Minimize}} \left(F(x,y) + \dfrac{\mu}{4}\|Ax+y\|^2 + \mu\langle c, y\rangle + \dfrac{\mu r}{2}\|y\|^2 + \mu V_2^r(x)\right) - \left(\mu V_1^r(x) + \dfrac{\mu}{4}\|Ax-y\|^2\right) \\ \text{s.t.} \\ H_i(x,y) \leq 0,\ i = 1,\ldots,q_1, \\ h_i(y) \leq 0,\ i = 1,\ldots,q_2. \end{cases}$$

The last problem is formulated as a DC program, which takes the general form

$$\text{Minimize}\left\{F_r^{\mu}(x,y) = g(x,y) - h(x,y) : (x,y) \in FS\right\},$$

where $F_r^{\mu}(x,y)$ is the objective function expressed as the difference of two convex functions g and h.

The function $g(x,y)$ represents the convex component of the decomposition and is differentiable whenever F is differentiable. It is defined as

$$g(x,y) = F(x,y) + \frac{\mu}{4}\|Ax+y\|^2 + \mu\langle c, y\rangle + \frac{\mu r}{2}\|y\|^2 + \mu V_2^r(x),$$

while the second convex function $h(x,y)$, is given by

$$h(x,y) = \mu V_1^r(x) + \frac{\mu}{4}\|Ax-y\|^2.$$

Proposition 3.4 *Suppose that Assumption 4 holds. The function*

$$V_1^r(x) = \frac{r}{2}\left[d\left(-\frac{Ax+c}{r}, FS_0\right)\right]^2,$$

is differentiable on $\mathbb{R}^m$, *and*

$$\nabla V_1^r(x) = A^{\top}\left[\Pi\left(-\frac{Ax+c}{r}, FS_0\right) + \frac{Ax+c}{r}\right].$$

Proof.

Let $z(x) = -\frac{Ax+c}{r}$ and $V_1(x) = \Big(d(z(x), FS_0)\Big)^2$, so $V_1^r(x) = \frac{r}{2}V_1(x)$.

Since FS_0 is nonempty, closed, and convex, the squared distance $\varphi(z) = \Big(d(z, FS_0)\Big)^2$ is differentiable everywhere (see [51], Exercise 3.2), with

$$\nabla\varphi(z) = 2\Big(z - \Pi(z, FS_0)\Big).$$

By the chain rule, $V_1 = \varphi \circ z$, so the Jacobian satisfies

$$\begin{aligned} JV_1(x) &= J\varphi\Big(z(x)\Big) \cdot Jz(x) \\ &= 2\Big(z(x) - \Pi(z(x), FS_0)\Big)^\top \left(-\frac{A}{r}\right) \\ &= \frac{2}{r}\Big(\Pi(z(x), FS_0) - z(x)\Big)^\top A. \end{aligned}$$

Thus,

$$\nabla V_1(x) = JV_1(x)^\top = \frac{2}{r}A^\top\Big(\Pi(z(x), FS_0) - z(x)\Big).$$

Finally,

$$\nabla V_1^r(x) = \frac{r}{2} \cdot \frac{2}{r}A^\top\Big(\Pi(z(x), FS_0) - z(x)\Big) = A^\top\Big(\Pi(z(x), FS_0) - z(x)\Big).$$

Substituting $z(x) = -\frac{Ax+c}{r}$, we conclude:

$$\nabla V_1^r(x) = A^\top\left[\Pi\left(-\frac{Ax+c}{r}, FS_0\right) + \frac{Ax+c}{r}\right].$$

■

The DC algorithm requires the computation of a subgradient of the function $h(x, y)$. Generally, subgradients are employed because DC programming is designed to handle nondifferentiable convex functions. However, in our case, the function $V_1^r(x)$ is differentiable. As a result, the function $h(x, y)$, which contains $V_1^r(x)$, is also differentiable. This implies that the gradient of $h(x, y)$ exists and can be computed directly, and the subdifferential of h at any point reduces to its gradient:

$$\partial h(x, y) = \{\nabla h(x, y)\}.$$

The projection $\Pi\left(-\frac{Ax+c}{r}, FS_0\right)$ consists of finding the point in the set FS_0 that is nearest to $-\frac{Ax+c}{r}$. This is an optimization problem in which we want to minimize the squared distance between any point y in FS_0 and the target point $-\frac{Ax+c}{r}$.

Mathematically, this is expressed as

$$\Pi\left(-\frac{Ax+c}{r}, FS_0\right) = \arg\min_{y\in FS_0} \left\| y + \frac{Ax+c}{r} \right\|^2. \tag{3.2}$$

The subdifferential of $h(x,y)$ is expressed as:

$$\partial h(x,y) = \nabla h(x,y) = \begin{pmatrix} \mu \nabla V_1^r(x) + \frac{\mu}{2} A^\top (Ax-y) \\ -\frac{\mu}{2}(Ax-y) \end{pmatrix} \tag{3.3}$$

The DC Algorithm (DCA), applied to the DC program (P_r^μ), is iterative. At each iteration k, the method computes the subgradient pair (p_1^k, p_2^k) of the convex function h at the current point (x^k, y^k), specifically:

$$(p_1^k, p_2^k) \in \partial h(x^k, y^k).$$

The subsequent iteration (x^{k+1}, y^{k+1}) is derived by solving the following convex optimization problem:

$$\text{Minimize} \left\{ g(x,y) - \langle p_1^k, x\rangle - \langle p_2^k, y\rangle, \quad (x,y) \in FS \right\}$$

Equivalently,

$$\text{Minimize} \left\{ g(x,y) - \left\langle \mu^k \nabla V_1^r(x^k) + \frac{\mu^k}{2} A^\top (Ax^k - y^k), x \right\rangle + \left\langle \frac{\mu^k}{2}(Ax^k - y^k), y \right\rangle, \ (x,y) \in FS \right\} \tag{3.4}$$

In the following subsection, we present the BL-DCA algorithm, which outlines the key steps of our proposed approach.

3.5.3 BL-DCA Algorithm

Instead of employing a fixed regularization parameter r, we adopt an adaptive strategy in which r is updated iteratively. This adaptive modification enables the algorithm to progressively enhance the impact of regularization. Initial iterations may require more regularization to maintain stability and direct the search. However, subsequent iterations can use a lower regularization value to achieve more accurate convergence to the optimal solution. This adaptive technique can enhance both convergence behavior and the quality of the final solution by balancing exploration and accuracy. The algorithm with adaptive regularization is outlined as follows

Algorithm 2: BL-DCA

Initial parameter configuration:

Select $\varepsilon > 0$, an initial point $(x^0, y^0) \in FS$, an initial penalty parameter $\mu^0 > 0$, and an initial regularization parameter $r^0 > 0$. Choose a step size $\alpha > 0$, $0 < \beta < 1$, and specify the maximum number of iterations, denoted by $itermax$.

Iterative Process:

for $k = 0, 1, \ldots, itermax$ **do**

1. Compute the projection $\Pi\left(-\frac{Ax^k+c}{r^k}, FS_0\right)$ according to equation (3.2).
2. Identify a subgradient pair $(p_1^k, p_2^k) \in \partial h(x^k, y^k)$ using equation (3.3).
3. Solve the convex optimization problem (3.4) to obtain the next iterate (x^{k+1}, y^{k+1}).

if $\left\|(x^{k+1}, y^{k+1}) - (x^k, y^k)\right\| \leq \varepsilon$, **then**
Stop. (x^{k+1}, y^{k+1}) is an optimal solution of problem (P_1).

end

4. Update parameters:
$$\mu^{k+1} = \mu^k + \alpha, \quad r^{k+1} = \beta r^k.$$
5. Increment iteration counter: set $k := k + 1$.

end

3.5.4 Applicability of the proposed method to linear bilevel programming

The proposed method is not limited to the original bilevel problem formulation; it can also be adapted to other types of bilevel optimization problems in which the lower-

level problem is a linear program of the form:

$$(P'_x) \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) = d^\top y \\ \text{s.t.} \\ A_1 x + B_1 y \geq b, \\ y \geq 0. \end{cases}$$

Despite its specific structure as a linear program, it can be reformulated to fit within the general formulation of (P_x) considered earlier. We consider the dual formulation of (P'_x), which is given by:

$$(PD_x) \quad \begin{cases} \underset{\lambda}{\text{Minimize}}\ \lambda^\top A_1 x - \lambda^\top b \\ \text{s.t.} \\ B_1^\top \lambda \leq d, \\ \lambda \geq 0, \end{cases}$$

where λ denotes the vector of dual variables associated with the constraints of the primal problem.

Under appropriate regularity assumptions, such as primal and dual feasibility and boundedness, the strong duality theorem guarantees that the optimal values of (P'_x) and (PD_x) are equal. This equivalence allows us to replace the value function of the original problem (P'_x), denoted by $V_p(x)$, with the dual value function $V_d(x)$. Consequently, we obtain a reformulated bilevel problem that is consistent with the structure studied in our proposed approach.

When the upper-level objective function is also linear, the overall bilevel program becomes a **linear bilevel programming problem**. By reformulating the lower-level problem through its dual and applying our algorithmic approach, we extend the scope of the method to address this important class of problems effectively. This demonstrates both the flexibility and generality of the proposed approach.

3.5.5 Numerical results

The BL-DCA algorithm was implemented using Julia version 1.10.1. To take advantage of the capabilities of the CPLEX solver, known for its efficiency in solving linear and quadratic programming problems, we focused exclusively on test cases with linear

constraints. For performance evaluation, we selected a subset of problems from the Bilevel Optimization Library (BOLIB) [83], a well-established benchmark repository that provides a wide range of bilevel test problems with detailed specifications and known optimal solutions. The availability of structured formulations and reference results in BOLIB makes it highly suitable for testing bilevel optimization algorithms.

In our numerical experiments, the stopping criterion was set to $\varepsilon = 10^{-2}$, and the computation time was measured in seconds. The initial values of the penalty and regularization parameters, namely (μ^0, α) and (r^0, β), as well as the starting point (x^0, y^0), are listed in Table 1. The solution computed by the BL-DCA method is denoted by (x^*, y^*), and the total number of iterations is denoted by Iter. The reference global optimum, taken from the relevant literature, is given as Optimal Solution.

Test problems

$$P_1[83]: \begin{cases} \underset{x,y}{\text{Minimize}} \ x_1 + y_1^2 + y_2^2 \\ \text{s.t.} \\ -1 \le x_1 \le 1 \\ -1 - x_2 \le 0 \\ 1 + x_2 \le 0 \\ y \in \arg\min\limits_{y} \begin{cases} x_1 y_1 + x_2 y_2 \\ \text{s.t.} \\ -2y_1 + y_2 \le 0 \\ y_1 \le 2 \\ 0 \le y_2 \le 2 \end{cases} \end{cases}$$

$$P_2[83]: \begin{cases} \underset{x,y}{\text{Minimize}} \ 2x_1 + x_2 - 2y_1 + y_2 \\ \text{s.t.} \\ -1 \le x_1 \le 1 \\ -1 \le x_2 \le -0.75 \\ y \in \arg\min\limits_{y} \begin{cases} x_1 y_1 + x_2 y_2 \\ \text{s.t.} \\ -2y_1 + y_2 \le 0 \\ y_1 \le 2 \\ 0 \le y_2 \le 2 \end{cases} \end{cases}$$

$$P_3[83]: \begin{cases} \underset{x,y}{\text{Minimize}} \ (x_1 - 0.5)^2 + (x_2 - 0.5)^2 - 3y_1 - 3y_2 \\ \text{s.t.} \\ y \in \arg\min\limits_{y} \begin{cases} x_1 y_1 + x_2 y_2 \\ \text{s.t.} \\ y_1 + y_2 \le 2 \\ -y_1 + y_2 \le 0 \\ y_1, y_2 \ge 0 \end{cases} \end{cases}$$

$$P_4[83]: \begin{cases} \underset{x,y}{\text{Minimize}} \ -x - y \\ \text{s.t.} \\ -0.5 \le x \le 0.5 \\ y \in \arg\min\limits_{y} \begin{cases} xy \\ \text{s.t.} \\ -1 \le y \le 1 \end{cases} \end{cases}$$

$$P_5[83]: \begin{cases} \underset{x,y}{\text{Minimize}} \ (x_1-0.5)^2+(x_2-0.5)^2+x_3^2-3y_1-3y_2-6y_3 \\ \text{s.t.} \\ y \in \arg\min\limits_{y} \begin{cases} x_1y_1+x_2y_2+x_3y_3 \\ \text{s.t.} \\ y_1+y_2+y_3 \le 2 \\ -y_1+y_2 \le 0 \\ y_1,y_2,y_3 \ge 0 \end{cases} \end{cases}$$

$$P_6: \begin{cases} \underset{x,y}{\text{Minimize}} \ (y_1-x_1+20)^2+(y_2-x_2+20)^2 \\ \text{s.t.} \\ y \in \arg\min\limits_{y} \begin{cases} 5x_1y_1-3x_1y_2+5x_2y_2+y_1+y_2 \\ \text{s.t.} \\ 0 \le y_1 \le 6 \\ 0 \le y_2 \le 6 \end{cases} \end{cases}$$

$$P_7[83]: \begin{cases} \underset{x,y}{\text{Minimize}} \ -x_1+10y_1-y_2 \\ \text{s.t.} \\ x \ge 0 \\ y \in \arg\min\limits_{y} \begin{cases} -y_1-y_2 \\ \text{s.t.} \\ x+y_1 \le 1 \\ x+y_2 \le 1 \\ y_1+y_2 \le 1 \\ y \ge 0 \end{cases} \end{cases}$$

$$
P_8[52]: \begin{cases}
\underset{(x,y)\geq 0}{\text{Minimize}} \ -4x_1 + 8x_2 + x_3 - x_4 + 9y_1 - 9y_2 \\
\text{s.t.} \\
-9x_1 + 3x_2 - 8x_3 + 3x_4 + 3y_1 \leq -1 \\
4x_1 - 10x_2 + 3x_3 + 5x_4 + 8y_1 + 8y_2 \leq 25 \\
4x_1 - 2x_2 - 2x_3 + 10x_4 - 5y_1 + 8y_2 \leq 21 \\
9x_1 - 9x_2 + 4x_3 - 3x_4 - y_1 - 9y_2 \leq -1 \\
-2x_1 - 2x_2 + 8x_3 - 5x_4 + 5y_1 + 8y_2 \leq 20 \\
7x_1 + 2x_2 - 5x_3 + 4x_4 - 5y_1 \leq 11 \\
y \in \underset{y\geq 0}{\arg\min} \begin{cases}
-9y_1 + 9y_2 \\
\text{s.t.} \\
-6x_1 + x_2 + x_3 - 3x_4 - 9y_1 - 7y_2 \leq -15 \\
4x_2 + 5x_3 + 10x_4 \leq 26 \\
-9x_1 + 9x_2 - 9x_3 + 5x_4 - 5y_1 - 4y_2 \leq -5 \\
5x_1 + 3x_2 + x_3 + 9x_4 + y_1 + 5y_2 \leq 32
\end{cases}
\end{cases}
$$

Results interpretation and discussion

Problems P_1–P_5 and P_7 are sourced from the BOLIB library to evaluate the proposed BL-DCA algorithm. The experimental results indicate that the algorithm consistently produces solutions that are optimal or close to the known optimal values. Moreover, the algorithm demonstrates reliable convergence, typically requiring a modest number of iterations and achieving solutions in a short computational time. In most cases, with appropriate parameter selections, μ and r, the algorithm converges rapidly. For instance, Problem P_1 reaches the optimal solution in four iterations when suitable values for r are chosen, resulting in minimal computation time. The algorithm's effectiveness is demonstrated in linear bilevel problems, as illustrated by the results obtained in Problem 8. Additionally, Problem P_6, which was proposed, was manually solved to validate the correctness of the algorithmic result. The manually derived solution matched the BL-DCA algorithm's output precisely, reinforcing its correctness.

In conclusion, the BL-DCA algorithm is an efficient approach for solving bilevel optimization problems, offering solution quality and low computational cost when properly configured.

Table 3.1: **Numerical Results for the BL-DCA Algorithm.**

Pr.	(μ^0, α)	(r^0, β)	(x^0, y^0)	(x^*, y^*)	CPU Time(s)	Iter	Optimal Solution
P_1[83]	$(1,2)$	$(1,0.6)$	$(0,-1,0,0)$	$(-0.01,-1,1,1.99)$	0.26	10	$(0,-1,1,2)$
	$(10,4)$	$(0.1,0.2)$	$(0,-1,0,0)$	$(0.06,-1,1,1.99)$	0.13	4	
	$(1,2)$	$(0.5,0.6)$	$(-1,-1,0,0)$	$(0.09,-1,1,1.99)$	0.96	7	
P_2[83]	$(0.1,2)$	$(0.1,0.1)$	$(-1,-1,0,0)$	$(-0.99,-0.98,1.99,1.99)$	0.12	4	$(-1,-1,2,2)$
	$(100,5)$	$(0.01,0.1)$	$(-1,-1,0,0)$	$(-0.99,-0.98,1.99,1.99)$	0.10	3	
	$(0.01,5)$	$(0.1,0.1)$	$(1,-0.75,0,0)$	$(-0.99,-0.98,1.99,1.99)$	0.14	4	
P_3[83]	$(1,2)$	$(0.1,0.6)$	$(0,0,0,0)$	$(-0.02,-0.02,1,0.99)$	0.15	4	$(0,0,1,1)$
	$(10,2)$	$(0.01,0.1)$	$(0,0,0,0)$	$(-0.001,-0.001,1,0.99)$	0.12	3	
	$(1,10)$	$(0.1,0.01)$	$(0,0,0,0)$	$(0.03,0.03,1,0.99)$	0.06	2	
P_4[83]	$(0.1,2)$	$(0.5,0.6)$	$(-0.5,0)$	$(-0.07,0.99)$	0.23	8	$(0,1)$
	$(1,2)$	$(0.01,0.06)$	$(0,0)$	$(-0.07,0.99)$	0.24	8	
	$(10,2)$	$(0.01,0.06)$	$(0,0)$	$(0.0.99)$	0.24	9	
P_5[83]	$(10^{-3},0.002)$	$(3,0.1)$	$(0,0,0,0,0,0)$	$(0.49,0.49,0,0,0,1.99)$	0.06	2	$(0.5,0.5,0,0,0,2)$
	$(0.1,0.01)$	$(3,0.1)$	$(0,0,0,0,0,0)$	$(0.49,0.49,0.05,0,0,1.99)$	0.11	4	
	$(1,2)$	$(0.1,0.1)$	$(0.5,0.5,0,0,0,0)$	$(0.49,0.49,0.07,0,0,1.99)$	0.04	2	
P_6	$(0.01,0.01)$	$(5,0.5)$	$(0,0,0,0)$	$(19.97,20.01,0,0)$	0.24	8	$(20,20,0,0)$
	$(0.001,0.01)$	$(5,0.1)$	$(0,0,0,0)$	$(19.97,20,0,0)$	0.14	4	
	$(10^{-4},0.01)$	$(5,0.01)$	$(2,2,6,6)$	$(19.99,19.99,0,0)$	0.10	3	
P_7[83]	$(0.1,0.001)$	$(4,0.01)$	$(0,0,0)$	$(0,0,1)$	0.13	4	$(0,0,1)$
	$(100,100)$	$(4,0.01)$	$(0.5,0.5,0)$	$(0,0,0.99)$	0.12	3	
	$(1000,10)$	$(0.01,0.01)$	$(0.5,0.5,0)$	$(0,0,0.99)$	0.04	2	
P_8[52]	$(10^{-5},10^{-5})$	$(0.1,0.6)$	$(0,0,0.125,$ $0,0,2.1607)$	$(1.49,0,0.799$ $0,0,2.075)$	0.05	2	$(1.50,0,0.8$ $0,0,2.075)$

Chapter 4

A novel approach to semi-vectorial bilevel programming

4.1 Introduction

Bilevel programming problems are a significant class of nonconvex optimization problems that present challenges for finding global optima due to their hierarchical structure. In this study, we introduce an efficient and robust method for addressing these issues. We analyze our approach in detail, as presented in [15], where we propose a novel method to solve these complex problems by converting the original bilevel formulation into an equivalent single-level optimization model. This transformation is achieved by integrating the optimal value function of the lower-level problem, which is approximated using α-dense curves. Finally, the reformulated problem is solved using two distinct evolutionary algorithms.

Moreover, we extend our approach to handle multicriteria bilevel programming problems, characterized by a single objective in the upper-level and multiple objectives in the lower-level, a class known as semi-vectorial bilevel programming problems. The effectiveness and performance of the proposed method are validated through a series of numerical experiments on nonlinear bilevel programming problem instances, which demonstrate the high efficiency and accuracy of our solution approach.

4.2 Evolutionary algorithms

Evolutionary algorithms are stochastic, population-based search techniques inspired by the principles of natural evolution. This section briefly overviews two notable metaheuristic algorithms, the Gray Wolf Optimizer (GWO) and Particle Swarm Optimization (PSO), and describes the key processes that govern their operation.

4.2.1 Gray wolf optimizer (GWO)

Gray Wolf Optimization (GWO) is a nature-inspired, population-based metaheuristic algorithm proposed by Mirjalili et al. [54]. It mimics the social structure of gray wolves by emphasizing three leading solutions: the best solution (α), the next best solution (β), and the third solution (σ). The algorithm changes the positions of the wolves according to mathematical equations that replicate their social behavior and encircling tactics during hunting. The following equations govern the position updates:

$$A = 2a^{(k)} \cdot r_1 - a^{(k)} \tag{4.1}$$

$$C = 2\,r_2 \tag{4.2}$$

$$D = |C \cdot X_p^{(k)} - X^{(k)}| \tag{4.3}$$

$$X^{(k+1)} = X_p^{(k)} - A \cdot D \tag{4.4}$$

where $a^{(k)} = \lambda_k e$, with λ_k defined as:

$$\lambda_k = \frac{2max - 2k}{max} \tag{4.5}$$

Where max is the maximum number of iterations, and k is the current number of iterations. The vectors **A** and **C** are acceleration coefficients that affect the movement of the wolves. $\mathbf{X}_p$ is the position of the prey, while **X** is the position of a gray wolf. The random vectors r_1 and r_2 are uniformly distributed in the interval $[0, 1]$.

Since the exact position of the prey is unknown, it is approximated using the three best solutions found so far, referred to as $\mathbf{X}_\alpha$, $\mathbf{X}_\beta$, and $\mathbf{X}_\sigma$. The positions of the wolves are then updated according to these reference points with the following equations:

$$D_\alpha = |C_1 \cdot X_\alpha - X|,\ D_\beta = |C_2 \cdot X_\beta - X|,\ D_\sigma = |C_3 \cdot X_\sigma - X|,$$

$$X_1 = X_\alpha - A_1 \cdot D_\alpha, \quad X_2 = X_\beta - A_2 \cdot D_\beta, \quad X_3 = X_\sigma - A_3 \cdot D_\sigma,$$

$$X^{k+1} = \frac{X_1 + X_2 + X_3}{3}. \tag{4.6}$$

In this context, $|X|$ represents the vector containing the absolute values of each component of X.

4.2.2 Particle swarm optimization (PSO)

Kennedy and Eberhart first introduced particle swarm optimization (PSO) in 1995. It was motivated by the collective behavior seen in schools of fish and flocks of birds. To efficiently explore the solution space, this technique iteratively updates the locations and velocities of a population of particles.

In this framework:

- $x_i^{(k)}$ denotes the position of the i-th particle at iteration k,
- $v_i^{(k)}$ represents the velocity of the i-th particle at iteration k,
- P_i^{best} is the personal best position discovered by particle i up to iteration k.
- g^{best} indicates the best position found by the entire swarm so far.

At each iteration, the velocity and position of each particle are updated using the following equations:

Velocity Update:

$$v_i^{(k+1)} = \omega v_i^{(k)} + c_1 r_1 \left(P_i^{\text{best}} - x_i^{(k)} \right) + c_2 r_2 \left(g^{\text{best}} - x_i^{(k)} \right) \tag{4.7}$$

Here, ω is the inertia weight, c_1 and c_2 are acceleration coefficients, and r_1, r_2 are random variables uniformly distributed over the interval $[0, 1]$.

Position Update:

$$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)} \tag{4.8}$$

This rule updates each particle's position by adding its new velocity to its current position.

4.3 A novel technique for solving bilevel programming problems

4.3.1 Unidimensional approximation

In 1983, Cherruault and Guillez introduced a transformation procedure [22] that significantly simplifies the original multidimensional optimization problem by reducing it to a single-variable problem. This reduction is achieved by employing a continuous parametric dense curve that projects the high-dimensional space onto a one-dimensional representation. This technique improves analytical efficiency and problem-solving capabilities by reducing dimensionality and consolidating complex interactions across multiple dimensions into a single governing parameter, while preserving the fundamental properties of the solution space.

We consider the following formulation of a bound-constrained global optimization problem:

$$(P'') \quad \underset{c \le x \le d}{\text{Minimize}}\ f(x),$$

where $x, c, d \in \mathbb{R}^m$, and the vectors c and d represent the lower and upper bounds, respectively. The objective function f is a continuous real-valued function, which may be non-differentiable.

We start by associating an α-dense search curve, denoted by $\mathcal{G}$, within the domain $[c, d]$, defined as:

$$\mathcal{G} : [0, T] \to [c, d],$$

$$t \mapsto (\mathcal{G}_1(t), \ldots, \mathcal{G}_m(t)),$$

where the parameter t acts as the search variable. The original problem (P'') is then approximated by a new problem $(P_{\mathcal{G}})$ which depends solely on the single variable t:

$$(P_{\mathcal{G}}) \quad \underset{t \in [0,T]}{\text{Minimize}}\ f(\mathcal{G}(t)),$$

where

$$f(\mathcal{G}(t)) = f(\mathcal{G}_1(t), \ldots, \mathcal{G}_m(t)).$$

Next, we will recall some definitions essential for subsequent discussions.

Definition 4.1 *A curve $\mathcal{G} : [0,T] \to [c,d]$, where $T > 0$, is said to be α-dense in $[c,d]$ if its image, $\mathcal{G}([0,T])$ is α-dense in $[c,d]$. This means that for every $y \in [c,d]$, there exists a $t \in [0,T]$ such that $\|y - \mathcal{G}(t)\| \leq \alpha$.*

Theorem 4.1 *[84] A parametrized curve defined by $\mathcal{G}(t) = (\mathcal{G}_1(t), \mathcal{G}_2(t), \ldots, \mathcal{G}_m(t))$ for $t \in [0,T]$ is α-dense in $[c,d]$ if the following conditions are met:*

1. *The functions $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ are continuous and surjective.*
2. *The functions $\mathcal{G}_2, \ldots, \mathcal{G}_m$ are periodic, with respective periods $\pi_2, \ldots, \pi_m$.*
3. *For any interval $I \subset [0,T]$ and for each $i = 2, \ldots, m$,*

$$\mu(I) < \pi_i \implies \mu(\mathcal{G}_{i-1}(I)) < \frac{\alpha}{\sqrt{m-1}},$$

here, μ denotes the Lebesgue measure.

Next, we introduce an example of an α-dense curve for our computational analysis.

Example 4.1 *[84] Let the function $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m) : \left[0, \frac{\pi}{\gamma_m}\right] \to [c,d]$ be defined as*

$$\begin{aligned}
\mathcal{G}_1(t) &= \frac{c_1 + d_1}{2} - \frac{c_1 - d_1}{2} \cos \gamma_1 t, \\
\mathcal{G}_2(t) &= \frac{c_2 + d_2}{2} - \frac{c_2 - d_2}{2} \cos \gamma_2 t, \\
&\vdots \\
\mathcal{G}_m(t) &= \frac{c_m + d_m}{2} - \frac{c_m - dm}{2} \cos \gamma_m t.
\end{aligned}$$

where $\gamma_1, \gamma_2, \ldots, \gamma_m$ are the parameters specified by:

$$\begin{aligned}
\gamma_1 &= 1, \\
\gamma_2 &= \frac{\alpha}{\pi(|c_2| + |d_2|)}, \\
\gamma_3 &= \frac{\alpha^2}{\pi^2(|c_2| + |d_2|)(|c_3| + |d_3|)}, \\
&\vdots \\
\gamma_m &= \frac{\alpha^{m-1}}{\pi^{m-1}(|c_2| + |d_2|)(|c_3| + |d_3|) \cdots (|c_m| + |d_m|)}.
\end{aligned}$$

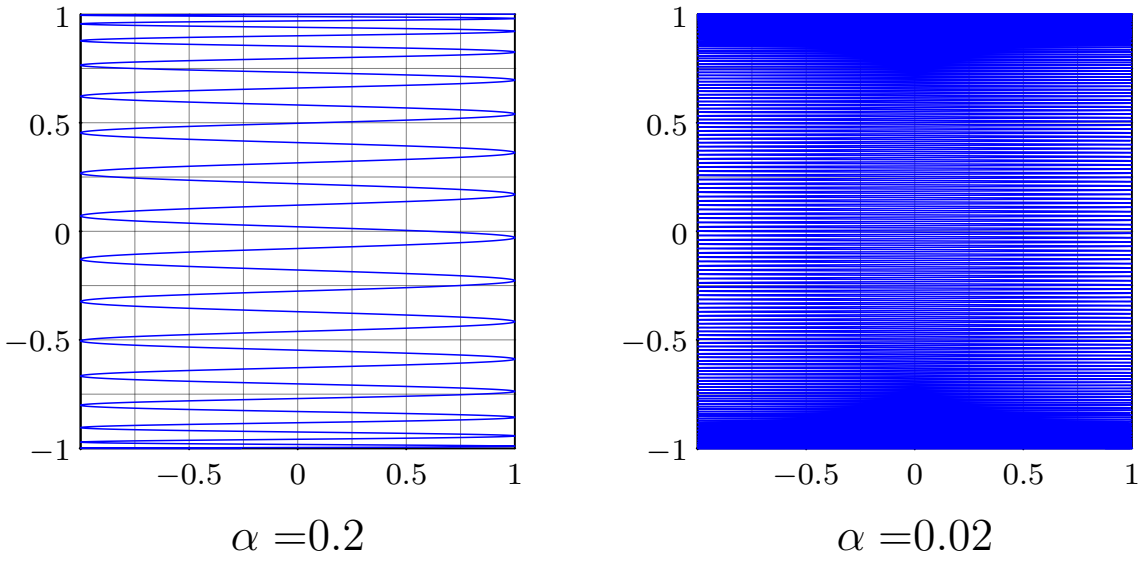

Figure 4.1: Densification of the square $[-1,1]^2$ using the curve $\mathcal{G}$.

Figure 4.1 illustrates the density of the square $[-1,1]^2$ using the curve $\mathcal{G}$ with $\alpha = 0.2, 0.02$.

According to Theorem 4.1, the curve $\mathcal{G}$ is α-dense in the domain $[c,d]$.

Figure 4.1 shows that the curve covers the designated area more effectively as the parameter α decreases. Smaller values of α yield a higher density of points along the curve, resulting in denser and more uniform coverage. Consequently, decreasing α enhances the curve's ability to fill voids.

4.3.2 Problem formulation

In this chapter, we will outline the steps of our proposed method for solving the following bilevel programming problem.

$$(P_2) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ a_1 \le x \le b_1, \\ G(x,y) \le 0, \\ y \in \Psi(x). \end{cases}$$

The set-valued mapping $\Psi : \mathbb{R}^n \rightrightarrows \mathbb{R}^m$ represents the optimal solution set of the lower-level problem:

$$(P_x) \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) \\ \text{s.t.} \\ a_2 \leq y \leq b_2, \end{cases}$$

where $a_1, b_1 \in \mathbb{R}^n$ are the lower and upper bounds on the decision variable x, and $a_2, b_2 \in \mathbb{R}^m$ are the lower and upper bounds on the decision variable y.

Our approach uses the LLVFR to convert the bilevel programming problem into a single-level optimization problem. The purpose is to apply PSO to solve the following problem:

$$(P_T) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ a_1 \leq x \leq b_1, \\ a_2 \leq y \leq b_2, \\ G(x,y) \leq 0, \\ f(x,y) - \varphi(x) \leq 0, \end{cases}$$

where

$$\varphi(x) = \min_y \{ f(x,y) \mid a_2 \leq y \leq b_2 \}.$$

Particle swarm optimization is commonly used to solve complex, non-linear optimization problems. However, due to the presence of the function $\varphi(x)$, which is not explicitly given, PSO cannot be applied directly. Therefore, our focus must shift to the optimal value function $\varphi(x)$. The fundamental approach to evaluating $\varphi(x)$ involves replacing it with a suitable approximation, as outlined below:

$$\varphi(x) \approx \min \Big\{ f(x,y_1), f(x,y_2), \ldots, f(x,y_p) \mid\ a_2 \leq y_i \leq b_2,\ i = 1, \ldots, p \Big\}.$$

where $\{y_1, y_2, \ldots, y_p\}$ denotes a collection of p points within the domain $[a_2, b_2]$.

To compute the approximate lower-level value function, it is essential to address the challenge posed by the multidimensional nature of the lower-level variable $y \in \mathbb{R}^m$. Therefore, we present a dimensionality reduction strategy that converts the multidimensional lower-level problem into a one-dimensional problem by utilizing a parametric α-dense curve, as suggested by Mora (1997) [53].

Applying this approach, the approximation of the lower-level problem becomes:

$$(P'_x) \quad \begin{cases} \underset{t}{\text{Minimize}}\ f(x, \mathcal{G}(t)) \\ \text{s.t.} \\ 0 \leq t \leq T. \end{cases}$$

where $\mathcal{G}(t)$ refers to the curve described previously, and

$$f(x, \mathcal{G}(t)) = f\left(x, \mathcal{G}_1(t), \mathcal{G}_2(t), \ldots, \mathcal{G}_n(t)\right)$$

Our objective is to allocate the points $\{y_1, y_2, \ldots, y_p\}$ uniformly along the parametric curve $\mathcal{G}$, with p being the total number of points inside the feasible region $[a_2, b_2]$. This distribution is depicted in Figures 4.2, 4.3, and 4.4. The parameter p can be adjusted to enhance the distribution, thereby improving the approximation accuracy as required.

Increasing the number of points p enhances the accuracy of the approximation of the set

$$S = \{y \in \mathbb{R}^m \mid a_2 \leq y \leq b_2\},$$

since a larger value of p provides a denser and more uniform coverage of the feasible region by points along $\mathcal{G}$, consequently, the parametric curve yields a more accurate and comprehensive representation of the feasible set.

Furthermore, as the parameter α decreases, the curve $\mathcal{G}$ describes the feasible region with greater precision, thereby improving the quality of the approximation. This interplay between the number of points p and the density controlled by α is crucial for balancing computational efficiency and solution quality when approximating the lower-level problem.

When the lower-level value function $\varphi(x)$ is approximated, it is replaced by the following estimate:

$$\varphi_{\text{app}}(x) = \min_{j=1,\ldots,p} \left\{ f\left(x, \mathcal{G}(t_j)\right) \right\},$$

where $\{t_1, t_2, \ldots, t_p\}$ is a subdivision of the interval $\left[0, \frac{\pi}{\gamma_m}\right]$. To ensure a sufficiently accurate approximation, the subdivision must satisfy

$$\max_{1 \leq i \leq p-1} (t_{i+1} - t_i) < \varepsilon,$$

for some small $\varepsilon > 0$.

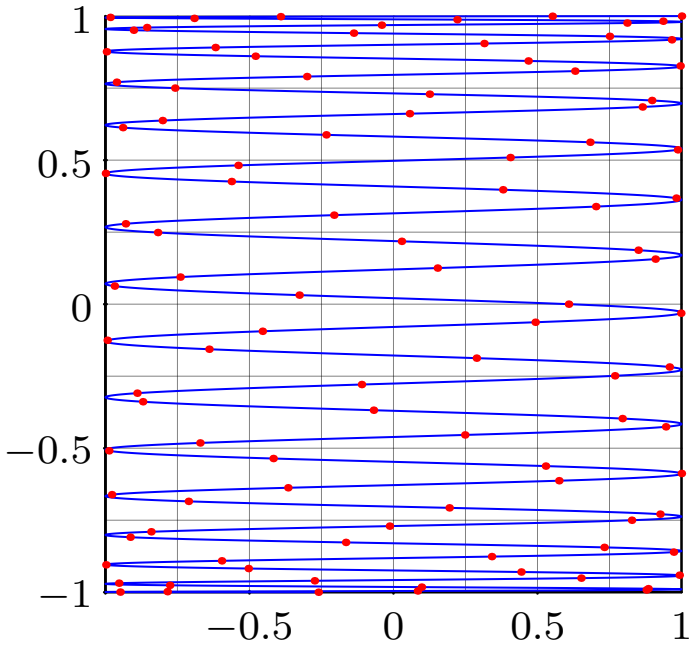

Figure 4.2: Distribution points of the curve for 100 points with $\alpha = 0.2$.

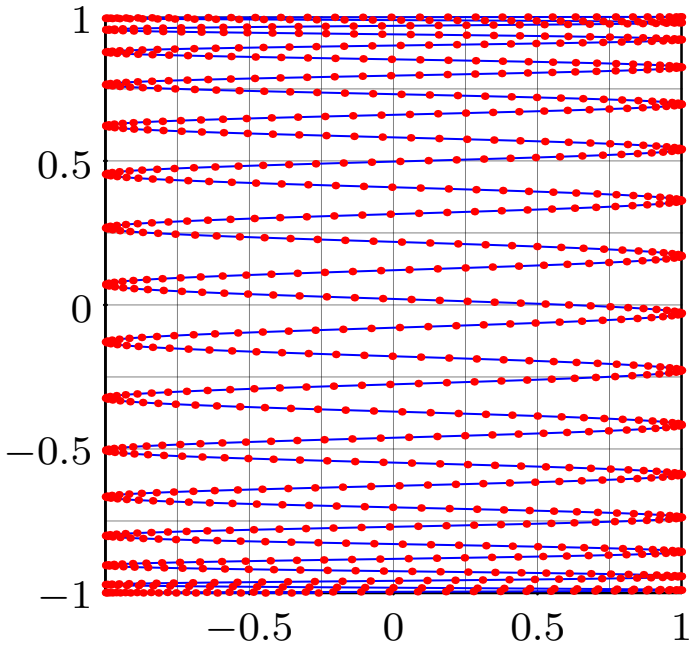

Figure 4.3: Distribution points of the curve for 1000 points with $\alpha = 0.2$.

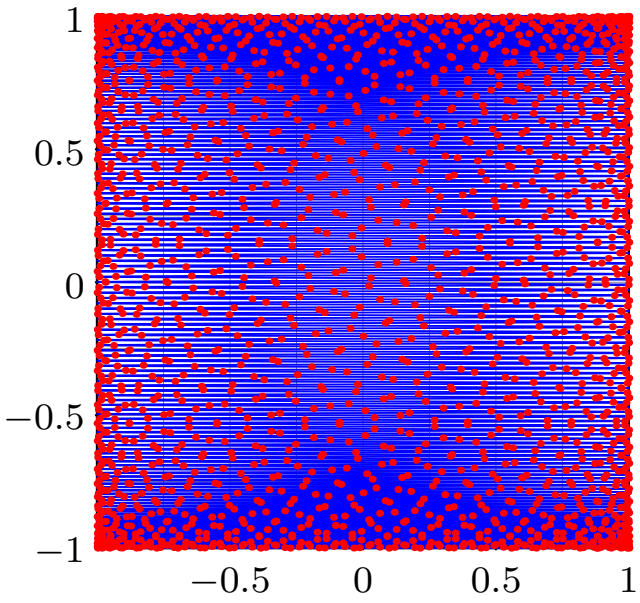

Figure 4.4: Distribution points of the curve for 2000 points with $\alpha = 0.02$.

Alternatively, a uniform subdivision may be employed by setting

$$t_i = ih, \quad \text{where} \quad h = \frac{\pi}{p\,\gamma_m}, \quad i = 0, 1, \ldots, p.$$

In this case, the step size h decreases as p increases, resulting in a finer approximation. Using this approximation $\varphi_{\text{app}}(x)$ and employing an exact penalty function for the last two constraints of problem (P_T), the problem becomes:

$$(P_\mu) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) + \mu\left(\sum_{i=1}^{m} \max\{0, G_i(x,y)\} + \max\left\{f(x,y) - \varphi_{app}(x), 0\right\}\right) \\ \text{s.t.} \\ a_1 \le x \le b_1, \\ a_2 \le y \le b_2, \end{cases}$$

where μ is a penalty parameter.

Now that we have transformed the original lower-level problem into a tractable form using a one-dimensional parametric representation, we can apply the PSO algorithm to solve the resulting approximation. In what follows, we provide a detailed explanation of the PSO and GWO algorithms as they are applied in our context:

4.3.3 BL-GWO Algorithm

Algorithm 3: BL-GWO Algorithm

Input: Population size n, maximum iterations $itermax$, penalty parameter $\mu > 0$, density $\alpha > 0$, number of distribution points p.

Output: Best solution X_α.

Step 1: Initialization;

Randomly initialize the gray wolf population

$$P = \{(x_i, y_i) : i = 1, \ldots, n\}.$$

Generate distribution points $z_i = \mathcal{G}(t_i)$, $i = 1, \ldots, p$;

Define the approximate value function: $\varphi_{\text{app}}(x) = \min\limits_{1 \le i \le p} \{f(x, z_i)\}$;

Define the penalty function:

$H(x, y) = \max\{f(x, y) - \varphi_{\text{app}}(x), 0\} + \sum_{i=1}^{m} \max\{0, G_i(x, y)\}$;

Define the objective function: $L(x, y) = F(x, y) + \mu H(x, y)$;

Evaluate $L(x_i, y_i)$ for all individuals in P;

Identify the three best solutions: X_α (best), X_β (second-best), X_σ (third-best);

Step 2: Main Loop;

for $k = 1$ **to** $itermax$ **do**

  foreach *candidate solution* $X \in P$ **do**

    Update coefficients $a^{(k)}$, A, and C using equations (4.5), (4.1), (4.2);

    Update the position of X using equation (4.6);

    Compute the fitness value L for the updated solution;

  end

  Update X_α, X_β, and X_σ based on the new population;

end

return X_α;

4.3.4 BL-PSO Algorithm

Algorithm 4: BL-PSO algorithm

Input: Population size n, maximum iterations $itermax$, penalty parameter $\mu > 0$, density $\alpha > 0$, number of distribution points p, acceleration coefficients c_1, c_2, inertia weight w.

Output: Global best solution g^{best}.

Step 1: Initialization;

Randomly initialize the particle population

$$P = \{(x_i, y_i) : i = 1, \ldots, n\}.$$

with initial velocities set to zero for all components;

Generate distribution points $z_i = \mathcal{G}(t_i)$, $i = 1, \ldots, p$;

Define the approximate value function: $\varphi_{\text{app}}(x) = \min\limits_{1 \le i \le p} \{f(x, z_i)\}$;

Define the penalty function:

$H(x, y) = \max\{f(x, y) - \varphi_{\text{app}}(x), 0\} + \sum_{i=1}^{m} \max\{0, G_i(x, y)\}$;

Define the objective function: $L(x, y) = F(x, y) + \mu H(x, y)$;

Determine the initial global best solution g_{best};

Step 2: Main Loop;

for $k = 1$ **to** $itermax$ **do**

  foreach *particle* $X \in P$ **do**

    Update the velocity and position of X using equations (4.7) and (4.8);

    Compute the fitness value $L(x_i, y_i)$;

    Update the personal best position P^{best} of the particle;

    Update the global best position g^{best} if necessary;

  end

end

return g^{best};

4.3.5 Analysis of computational results

In this subsection, we demonstrate the effectiveness of the proposed algorithm by applying it to several nonlinear bilevel programming problems. These test cases are chosen to highlight the robustness and applicability of the method to complex problem structures. To further validate our approach, we compare the solutions obtained with our algorithm to those reported in the literature.

In addition, we evaluate the performance of the two algorithms, BL-GWO and BL-PSO, in terms of both computational time and solution quality. For the BL-PSO algorithm, the following parameter settings are used: acceleration coefficients $c_1 = c_2 = 1.2$, inertia weight $w = 0.8$, population size $n = 1000$, and curve density $\alpha = 0.01$. The parameters μ, max, and p are set to the same values as those used in [15]. All implementations are carried out using the Julia programming language, version 1.10.1.

Test Problems

Problem 1.[55]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = \left(x + \frac{1}{2}\right)^2 + \frac{1}{2}y^2 \\ \text{s.t.} \\ x \in [-1,1] \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{-1 \le y \le 1}{\operatorname{argmin}} \frac{1}{2}xy^2 + \frac{1}{4}y^4$$

Problem 2.[55]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = y \\ \text{s.t.} \\ x \in [-1,1] \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{-0.8 \le y \le 1}{\operatorname{argmin}} x\left(16y^4 + 2y^3 - 8y^2 - \frac{3}{2}y + \frac{1}{2}\right)$$

Problem 3.[26]

$$\begin{cases} \underset{x,y \ge 0}{\text{Minimize}}\ F(x,y) = (x-1)^2 + (y-1)^2 \\ \text{s.t.} \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{y}{\operatorname{argmin}}\ 0.5y^2 + 500y - 50xy$$

Problem 4.[55]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = x^2 - y \\ \text{s.t.} \\ x \in [0,1] \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{0\le y\le 3}{\operatorname{argmin}} \left((y-1-0.1x)^2 - 0.5 - 0.5x\right)^2$$

Problem 5.[55]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = x_1y_1 + x_2y_2^2 + x_1x_2y_3^3 \\ \text{s.t.} \\ x \in [-1,1] \\ 0.1 - x_1^2 \le 0, \\ -2.5 + y_1^2 + y_2^2 + y_3^2 \le 0, \\ 1.5 - y_1^2 - y_2^2 - y_3^2 \le 0, \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{-1\le y\le 1}{\operatorname{argmin}} x_1y_1^2 + x_2y_2^2 + (x_1 - x_2)y_3^2$$

Problem 6.[55]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = (x-3)^2 + (y-2)^2 \\ \text{s.t.} \\ x \in [0,8] \\ -2x + y \le 1, \\ x - 2y \le -2, \\ x + 2y \le 14, \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{0\le y\le 10}{\operatorname{argmin}} (y-5)^2$$

Problem 7.[68]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = (x_1-30)^2 + (x_2-20)^2 - 20y_1 + 20y_2 \\ \text{s.t.} \\ x_2 - 15 \le 0 \\ x_1 + x_2 - 25 \le 0 \\ -x_1 - 2x_2 + 30 \le 0 \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{0\le y\le 10}{\operatorname{argmin}} (x_1 - y_1)^2 + (x_2 - y_2)^2$$

Problem 8.[49]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ F(x,y) = \displaystyle\sum_{i=1}^{10}(|x_i-1|+|y_i|) \\ \text{s.t.} \qquad \text{where} \quad \Psi(x) = \underset{-\pi\le y\le\pi}{\text{argmin}} \exp\left(\left[1+\displaystyle\sum_{i=1}^{10}(y_i^2/4000)-\prod_{i=1}^{10}\cos(y_i/\sqrt{i})\right]\sum_{i=1}^{10}x_i^2\right) \\ y\in\Psi(x) \end{cases}$$

Results

Each experiment is performed over 20 independent runs, and the best solution among these runs is selected as the corresponding global optimum.

Table 4.1 compares the solutions obtained by the proposed algorithm and those reported in related references, showing the best solutions (x^*, y^*) along with their associated upper-level objective value F^*. Table 4.2 provides a comparative analysis of the two evolutionary algorithms, BL-PSO and BL-GWO.

Comments

First, it is important to note that problems 1, 2, 4, 5, and 6 are taken from the benchmark study presented in [55], which provides established theoretical solutions for these cases. A comparison of our algorithm's results with these theoretical benchmarks shows that our method consistently achieves exact or near-exact solutions, demonstrating its reliability and accuracy on these test problems.

An exciting finding arises with Problem 5; while the reference [55] reports a single global optimum, our algorithm identifies two distinct global solutions.

Finally, for problems 7 and 8, although no theoretical optima are explicitly reported in the literature, our algorithm produces highly satisfactory results. Taken together, these encouraging results reinforce our algorithm's strength and versatility across a range of challenging bilevel optimization problems.

Table 4.2 summarises the comparative results between BL-GWO and BL-PSO. While both algorithms perform similarly in terms of solution quality, BL-GWO requires more computational time than BL-PSO. Although BL-GWO is effective in solving, it can increase processing overhead. In contrast, BL-PSO exhibits faster convergence, making it more efficient in terms of runtime.

Table 4.1: **Comparison of the best solutions obtained by the proposed algorithm and those reported in the literature for the test problems.**

Problems	**BL-PSO**		**Reference**	
	(x^*, y^*)	F^*	(x^*, y^*)	F^*
Problem 1	$(-0.2513, -0.5)/(-0.2513, 0.5)$	0.1868	$(-0.25, \pm 0.5)$	0.1875
Problem 2	$(4.688 \times 10^{-11}, -0.8)$	-0.8	$(0, -0.8)$	-0.8
Problem 3	$(1, 0)$	1	$(1, 0)$	1
Problem 4	$(0.2113, 1.8006)$	-1.7560	$(0.210, 1.79)$	-1.755
Problem 5	$(-1, -1, 1, -1, -0.7071)$ / $(-1, -1, 1, 1, -0.7070)$	-2.3535	$(-1, -1, 1, 1, -0.707)$	-2.35
Problem 6	$(3, 4.9921)$	8.9528	$(3, 5)$	9
Problem 7	$(20, 4.9999, 10, 4.8255)$	221.511	$(20, 5, 9.775, 4.959)$	228.7
Problem 8	$(0.9964, 0.9999,$ $1.0006, 0.9999,$ $0.9998, 1.0000$ $0.9995, 0.9999,$ $1.0004, 1.0002)$ $(-0.0011, 0.0056,$ $-7.395 \times 10^{-5}, -0.0013,$ $0.0001, 0.0004,$ $-0.0003, 0.0024,$ $0.0014, -0.0001)$	0.0189	$(1.000087, 1.000387,$ $1.000230, 1.000338,$ $1.000190, 0.999098,$ $1.000254, 0.999878,$ $1.000146, 1.000592,$ $(3.56 \times 10^{-6}, -2.11 \times 10^{-7},$ $7.38 \times 10^{-7}, 5.02 \times 10^{-7},$ $-5.38 \times 10^{-7}, -1.26 \times 10^{-6},$ $-9.99 \times 10^{-7}, -2.30 \times 10^{-6},$ $-9.08 \times 10^{-8}, 1.71 \times 10^{-6})$	3.26×10^{-3}

Performance comparison of our algorithm with that proposed by [82]

This part provides a comparative analysis of the performance of our suggested algorithm and the approach proposed by [82]. For this comparison, problems 3, 6, and 7 were chosen using the parameter values $n = 40$, max $= 100$, $p = 1000$, and $\alpha = 0.01$. The results are presented in Table 4.3.

Although our approach is based on approximating the value function of the lower-level problem, which yields approximate solutions, it provides a notable advantage in computational efficiency. In particular, our approach significantly reduces the overall computational time compared to the technique presented by [82]. This improvement

Table 4.2: **Comparative performance analysis of BL-PSO and BL-GWO**

Problems	BL-GWO			BL-PSO		
	(x^*, y^*)	F^*	CPU Time(s)	(x^*, y^*)	F^*	CPU Time(s)
Problem 1	$(-0.2513, 0.5)$	0.1868	6.876	$(-0.2513, 0.5)$	0.1868	4.854
Problem 2	$(4.0833 \times 10^{-20}, -0.8)$	-0.8	8.6631	$(4.688 \times 10^{-11}, -0.8)$	-0.8	5.407
Problem 3	$(1, 0)$	1	4.312	$(1, 0)$	1	3.208
Problem 4	$(0.2115, 1.8006)$	-1.7559	5.064	$(0.2113, 1.8006)$	-1.7560	3.222
Problem 5	$(-1, -1, 1, -1, -0.7070)$	-2.3535	4.721	$(-1, -1, 1, -1, -0.7071)$	-0.3535	3.250
Problem 6	$(3, 4.9921)$	8.9526	3.412	$(3, 4.9921)$	8.9528	2.438
Problem 7	$(19.9998, 5.0018, 10, 4.8298)$	221.546	25.538	$(20, 4.9999, 10, 4.8255)$	221.511	16.540
Problem 8	$(0.9996, 0.9993,$ $0.9995, 1.0003,$ $1.0000, 1.0002,$ $0.9997, 0.9990,$ $1.0007, 0.9995)$ $(0.0071, 0.0027,$ $0.0036, -0.0129,$ $-0.0355, 0.0015,$ $0.0086, -0.0126$ $-0.0031, 0.0015)$	0.094	307.993	$(0.9964, 0.9999,$ $1.0006, 0.9999,$ $0.9998, 1.0000,$ $0.9995, 0.9999,$ $1.0004, 1.0002)$ $(-0.0011, 0.0056,$ $-7.395 \times 10^{-5}, -0.0013,$ $0.0001, 0.0004,$ $-0.0003, 0.0024,$ $0.0014, -0.0001)$	0.0189	235.029

Table 4.3: **Comparison between the proposed algorithm and that of [82]**

Problems	BL-PSO		[82]	
	(x^*, y^*)	CPU Time(s)	(x^*, y^*)	CPU Time(s)
Problem 3	$(1.0, 0.0)$	0.1566	$(1.0, -9.974 \times 10^{-9})$	50.1985
Problem 6	$(3.0003, 4.9921)$	0.1001	$(3.0, 4.9999)$	46.2765
Problem 7	$(19.9978, 5.0017, 10.0, 4.7580)$	0.4016	$(19.8371, 5.1091, 10.00, 5.1091)$	83.8749

in efficiency is particularly beneficial in practical applications where rapid decision making is essential, and small approximations in the solution are acceptable trade-offs for faster results.

4.4 BL-PSO Algorithm for solving semi-vectorial bilevel programming problem

4.4.1 Semi-vectorial bilevel programming problem

Semi-vectorial bilevel programming problems represent a special category of bilevel optimization problems in which the upper-level (leader) problem has a scalar objective function. In contrast, the lower-level (follower) problem has a vector-valued (multiobjective) objective function. This framework establishes a hierarchical decision-making process with two agents: the leader first makes a decision, followed by the follower, who solves a multicriteria problem influenced by the leader's choice. The general formulation of a semi-vectorial bilevel programming problem is given by:

$$\begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ G(x,y) \leq 0, \\ y \in \Psi_e(x), \end{cases}$$

where $\Psi_e(x)$ denotes the set of efficient solutions of the lower-level problem

$$\begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) := \Big(f_1(x,y), f_2(x,y), \cdots, f_q(x,y)\Big) \\ \text{s.t.} \\ g(x,y) \leq 0, \end{cases}$$

with $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $G : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_1}$, $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{p_2}$, $f_i : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $i = 1, \ldots, q$.

In this context, we focus on a specific class of semi-vectorial bilevel programming problems in which box constraints are explicitly incorporated into both hierarchical

levels. The problem takes the following form:

$$(SBP) \quad \begin{cases} \underset{(x,y)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ a_1 \leq x \leq b_1, \\ G(x,y) \leq 0, \\ y \in \Psi_e(x), \end{cases}$$

where $\Psi_e(x)$ is the set of efficient solutions of the lower level problem

$$(MP_x) \quad \begin{cases} \underset{y}{\text{Minimize}}\ f(x,y) := \Big(f_1(x,y), f_2(x,y), \cdots, f_q(x,y)\Big) \\ \text{s.t.} \\ a_2 \leq y \leq b_2, \end{cases}$$

4.4.2 Adapting the BL-PSO algorithm for semi-vectorial bilevel optimization

To adapt the BL-PSO algorithm for solving semi-vectorial bilevel programming (SBP) problems, we first reformulate the problem as a single-level optimization problem. Assuming that each function $f_i(x,y)$ is convex in y for a fixed x, we apply the weighted sum scalarization technique to the lower-level problem. This yields the following equivalent formulation:

$$(SBP_s) \quad \begin{cases} \underset{(x,y,\lambda)}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ a_1 \leq x \leq b_1, \\ 0 \leq \lambda \leq 1,\ \sum_{i=1}^{q} \lambda_i = 1, \\ G_i(x,y) \leq 0, \\ y \in \Psi(x,\lambda), \end{cases}$$

where:

$$\Psi(x,\lambda) = \min_{a_2 \leq y \leq b_2} \sum_{i=1}^{q} \lambda_i f_i(x,y).$$

This reformulation approach using scalarization has been adopted by several researchers (e.g., [36], [44]). The theoretical connection between the original bilevel

multicriteria problem (SBP) and its scalarized form (SBP_s) is discussed in [29] and clarified in [31].

By employing the lower-level value function reformulation, we obtain the following equivalent problem:

$$(SBP_s) \quad \begin{cases} \underset{x,y,\lambda}{\text{Minimize}}\ F(x,y) \\ \text{s.t.} \\ a_1 \le x \le b_1, \\ 0 \le \lambda \le 1,\ \sum_{i=1}^{q} \lambda_i = 1, \\ G_i(x,y) \le 0, \\ \sum_{i=1}^{q} \lambda_i f_i(x,y) - \Psi(x,\lambda) \le 0, \end{cases}$$

Now, by applying an exact penalization approach, we get

$$\begin{cases} \underset{(x,y,\lambda)}{\text{Minimize}}\ F(x,y) + \mu\left(\max\left\{\sum_{i=1}^{q} \lambda_i f_i(x,y) - \Psi(x,\lambda), 0\right\} + \sum_{i=1}^{m} \max\{0, G_i(x,y)\} + \left|\sum_{i=1}^{q} \lambda_i - 1\right|\right) \\ \text{s.t.} \\ a_1 \le x \le b_1, \\ a_2 \le y \le b_2, \\ 0 \le \lambda_i \le 1,\ i = 1,...,q. \end{cases}$$

Therefore, our proposed algorithm, BL-PSO, can effectively solve this problem. By exploiting the advantages of particle swarm optimization in the bilevel programming problem, BL-PSO provides a robust and efficient approach to deal with the hierarchical structure and complexity of semi-vectorial bilevel problems.

4.4.3 Computational tests

We present an illustrative example of a semi-vectorial bilevel programming problem to demonstrate the application and effectiveness of our proposed algorithm.

Problem 1.[40]

$$\begin{cases} \underset{x,y}{\text{Minimize}}\ (y_1 - 1)^2 + y_2^2 + x^2 \\ \text{s.t.} \\ x \in [-1, 2], \\ y \in \Psi(x) \end{cases} \qquad \text{where} \quad \Psi(x) = \underset{-1 \le y \le 2}{\text{argmin}}\left(y_1^2 + y_2^2,\ (y_1 - x)^2 + y_2^2\right).$$

We employ the weighted sum method to scalarize the lower-level multicriteria problem:

$$\begin{cases} \underset{y}{\text{Minimize}} \left(y_1^2 + y_2^2,\ (y_1 - x)^2 + y_2^2 \right) \\ \text{s.t.} \\ y \in [-1, 2], \end{cases}$$

where the weights λ_i, for $i = 1, 2$, satisfy the condition $\lambda_1 + \lambda_2 = 1$. By varying these weights, we generate the set of efficient solutions illustrated in Figure 4.5. Notably, since $y_2 = 0$ for all efficient points, the plot is presented in terms of y_1 and x only.

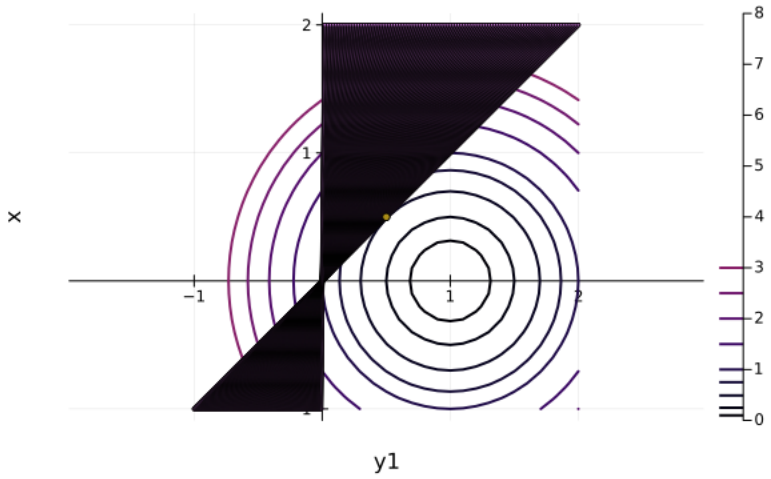

Figure 4.5: Level curves of F together with the set of efficient points for the lower-level problem at each fixed x.

The theoretical optimal solution is known to be $(x^*, y^*) = (0.5, 0.5, 0)$. Applying our BL-PSO algorithm, we obtain a solution close to this optimum $\left(0.48,\ 0.54,\ 5.46 \times 10^{-5}\right)$, with the weight vector $\lambda = (0, 1)$.

Based on the results from Problem 1, our proposed approach exhibits significant potential for extension to semi-vectorial bilevel optimization problems.

General Conclusion

This thesis has developed methods for solving bilevel optimization problems by analyzing new approaches designed to address the inherent challenges of these complex problems. At the core of this work are DC programming and innovative approximation techniques, which provide a robust framework for tackling bilevel scenarios where traditional methods often fall short.

One of the main contributions is the formulation of a DC programming-based approach for bilevel problems, particularly those characterized by non-convexity. The primary innovation is the regularization of the lower-level value function, enabling an effective reformulation as a DC program. This transformation provides a novel approach to non-convex bilevel problems where traditional methods are often inadequate. While DC programming generally produces local solutions, it has proven effective for non-convex problems.

Additionally, we introduced an innovative approximation technique for bilevel programming problems, approximating the value function using α-dense curves. By transforming the bilevel problem into a single-level problem and using space-filling curves for approximation, the resulting problem was solved using two evolutionary algorithms. Our approach demonstrated higher computational efficiency than previous studies (see [37], [82]). We also extended our method to semi-vectorial bilevel programming problems using the weighted sum method. Numerical experiments on bilevel programming problems demonstrated the efficiency and validity of our approach.

In summary, the approaches presented provide powerful tools for solving non-convex bilevel problems, and the future development of more advanced methods will further enhance their applicability in a wider range of bilevel optimization scenarios.

Bibliography

[1] Y. Abo-Elnaga and S. Nasr: Modified evolutionary algorithm and chaotic search for bi-level programming problems. Symmetry, 12 (2020), 767–796.

[2] E. Aiyoshi and K. Shimizu: A solution method for the static constrained Stackelberg problem via penalty method. IEEE Transactions on Automatic Control, 29(12) (1984), 1111–1114.

[3] G. B. Allende and G. Still: Solving bilevel programs with the KKT-approach. Mathematical Programming, 138 (2013), 309–332.

[4] M. J. Alves: Using MOPSO to solve multi-objective bi-level linear problems. In International Conference on Swarm Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), 332–339.

[5] M. J. Alves and J. P. Costa: An algorithm based on particle swarm optimization for multi-objective bi-level linear problems. Applied Mathematics and Computation, 247 (2014), 547–561.

[6] M. Amouzegar and K. Moshirvaziri: A penalty method for linear bilevel programming problems. Multilevel Optimization: Algorithms and Applications (1998), 251–271.

[7] G. Anandalingam and D. J. White: A solution method for the linear static Stackelberg problem using penalty functions. IEEE Transactions on Automatic Control, 35(10) (1990), 1170–1173.

[8] G. Anandalingam and D. J. White: A penalty function approach for solving bi-level linear programs. Journal of Global Optimization, 3 (1993), 397–419.

[9] J. S. Angelo, E. Krempser and H. J. Barbosa: Differential evolution for bilevel programming. In 2013 IEEE Congress on Evolutionary Computation. IEEE (2013), 470–477.

[10] J. S. Angelo, E. Krempser and H. J. C. Barbosa: Differential evolution assisted by a surrogate model for bi-level programming problems. IEEE Congress on Evolutionary Computation (CEC) (2014), 1784–1791.

[11] Z. Ankhili and A. Mansouri: An exact penalty on bi-level programs with linear vector optimization lower level. European Journal of Operational Research, 197 (2009), 36–41.

[12] A. Anzi and M. S. Radjef: A DC Algorithm for Solving Quadratic-linear Bilevel Optimization Problems. In Modelling, Computation and Optimization in Information Systems and Management Sciences. Proceedings of the 3rd International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences–MCO 2015–Part I. Springer International Publishing (2015), 119–129.

[13] J. Bard and J. Moore: A branch and bound algorithm for the bi-level programming problem. SIAM Journal on Scientific and Statistical Optimization, 52(3) (2003), 333–359.

[14] H. Benson: Existence of efficient solutions for vector maximization problems. Journal of Optimization Theory and Applications, 26(4) (1978), 569–580.

[15] W. Bouguern, S. Addoune and H. Debbiche: A novel approach for solving bi-level mono-objective and multi-objective programming problems using evolutionary algorithms. The Journal of Engineering and Exact Sciences, 10(9) (2024), 20661–20661.

[16] W. Bouguern, S. Addoune and H. Debbiche: Solving a Class of Bilevel Programming Problems by DC Programming and DCA. Nonlinear Dynamics and Systems Theory, 25(6) (2025), 596–607.

[17] H. Bonnel and J. Morgan: Semi-vectorial Bi-level Optimization Problem: Penalty Approach. Journal of Optimization Theory and Applications, 131 (2006), 365–382.

[18] H. I. Calvete, C. Gate and P. M. Mateo: A new approach for solving linear bi-level problems using genetic algorithms. European Journal of Operational Research, 188 (2008), 14–28.

[19] H. I. Calvete and C. Galé: Linear bi-level programs with multiple objectives at the upper level. Journal of Computational and Applied Mathematics, 234 (2010), 950–959.

[20] H. I. Calvete and C. Galé: On linear bi-level problems with multiple objectives at the lower level. Omega, 39 (2011), 33–40.

[21] Y. Chen and M. Florian: The nonlinear bilevel programming problem: Formulations, regularity and optimality conditions. Optimization, 32(3) (1995), 193–209.

[22] Y. Cherruault and A. Guillez: Une méthode pour la recherche du minimum global d'une fonctionnelle. CRAS Paris, 296 (1983), 175–178.

[23] B. Colson, P. Marcotte and G. Savard: An overview of bilevel optimization. Annals of Operations Research, 153 (2007), 235–256.

[24] K. Deb and A. Sinha: An efficient and accurate solution methodology for bi-level multi-objective programming problems using a hybrid evolutionary-local-search algorithm. Evolutionary Computation Journal, 18 (2010), 403–449.

[25] S. Dempe: A bundle algorithm applied to bilevel programming problems with non-unique lower level solutions. (2000).

[26] S. Dempe: Foundation of bilevel programming nonconvex optimization and its Applications. Springer (2002).

[27] S. Dempe, J. Dutta and B. Mordukhovich: New necessary optimality conditions in optimistic bi-level programming. Optimization (2007).

[28] S. Dempe and J. Dutta: Is bilevel programming a special case of mathematical programming with equilibrium constraints? Mathematical Programming, 131 (2012), 37–48.

[29] S. Dempe and A. Zemkoho: New optimality conditions for the semi-vectorial bilevel optimization problem. Journal of Optimization Theory and Applications, 157 (2013), 54–74.

[30] S. Dempe and P. Mehlitz: Semi-vectorial bi-level programming versus scalar bilevel programming. Optimization, 157 (2019), 657–679.

[31] S. Dempe and S. Franke: Solution of bi-level optimization problems using the KKT approach. Optimization, 68 (2019), 1471–1489.

[32] S. Dempe et al.: Bilevel programming problems. Energy Systems. Springer, Berlin, 10.978-3 (2015), 53–56.

[33] M. Ehrgott: Multicriteria optimization. Vol. 491. Springer Science & Business Media (2005).

[34] A. Fischer, A. B. Zemkoho and S. Zhou: Semismooth Newton-type method for bilevel optimization: global convergence and extensive numerical experiments. Optimization Methods and Software, 37(5) (2022), 1770–1804.

[35] L. L. Gao, J. J. Ye, H. Yin et al.: Moreau envelope based difference-of-weakly-convex reformulation and algorithm for bilevel programs. arXiv preprint arXiv:2306.16761 (2023).

[36] A. Gupta and Y. S. Ong: An evolutionary algorithm with adaptive scalarization for multi-objective bi-level programs. 2015 IEEE Congress on Evolutionary Computation (CEC) (2015).

[37] J. Han, G. Zhang, Y. Hu and J. Lu: A solution to bi/tri-level programming problems using particle swarm optimization. Information Sciences, 370 (2016), 519–537.

[38] R. Horst and N. V. Thoai: DC programming: overview. Journal of Optimization Theory and Applications, 103 (1999), 1–43.

[39] F. Jara-Moroni, J. S. Pang and A. Wächter: A study of the difference-of-convex approach for solving linear programs with complementarity constraints. Mathematical Programming, 169 (2018), 221–254.

[40] A. M. João, C. H. Antunes and P. Carrasqueira: A PSO approach to semi-vectorial bi-level programming: pessimistic, optimistic and deceiving solutions. Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (2015), 599–606.

[41] J. Kennedy and R. Eberhart: Particle swarm optimization. Proceedings of ICNN'95 - International Conference on Neural Networks, 4 (1995), 1942–1948.

[42] M. J. Kuo and C. C. Huang: Application of particle swarm optimization algorithm for solving bi-level linear programming problem. Computers & Mathematics with Applications, 58 (2009), 678–685.

[43] H. A. Le Thi and T. Pham Dinh: DC programming and DCA: thirty years of developments. Mathematical Programming, 169(1) (2018), 5–68.

[44] H. Li and L. Zhang: An efficient solution strategy for bi-level multi-objective optimization problems using multi-objective evolutionary algorithm. Soft Computing, 25 (2021), 8241–8261.

[45] X. Li, P. Tian and X. Min: A hierarchical particle swarm optimization for solving bi-level programming problems. Lecture Notes in Computer Science (2006), 1169–1178.

[46] G. H. Lin, M. Xu and J. J. Ye: On solving simple bi-level programs with a non-convex lower level program. Mathematical Programming (2014).

[47] Y. Lv, T. Hu, G. Wang and Z. Wan: A penalty function method based on Kuhn–Tucker condition for solving linear bilevel programming. Applied Mathematics and Computation, 188(1) (2007), 808–813.

[48] Y. Lv and Z. Wan: Solving linear bi-level multi-objective programming problem via exact penalty function approach. Journal of Inequalities and Applications, 2015 (2015), 1–12.

[49] L. Ma and G. Wang: A solving algorithm for nonlinear bi-level programming problems based on human evolutionary model. Algorithms (2020).

[50] R. Mathieu, L. Pittard and G. Anandalingam: Genetic algorithm based approach to bi-level linear programming. RAIRO-Operations Research, 28(1) (1994), 1–21.

[51] B. S. Mordukhovich and N. M. Nam: An Easy Path to Convex Analysis and Applications. Morgan & Claypool Publishers (2014).

[52] K. Moshirvaziri, M. A. Amouzegar and S. E. Jacobsen: Test problem construction for linear bilevel programming problems. Journal of Global Optimization, 8 (1996), 235–243.

[53] G. Mora and Y. Cherruault: Characterization and generation of α-dense curves. Computers and Mathematics with Applications, 33 (1997), 83–91.

[54] S. Mirjalili, S. M. Mirjalili and A. Lewis: Grey wolf optimizer. Advances in Engineering Software, 69 (2014), 46–61.

[55] A. Mitsos and P. Barton: A test set for bi-level programs. (2006).

[56] Y. S. Niu: On the convergence analysis of DCA. arXiv preprint arXiv:2211.10942 (2022).

[57] V. Oduguwa and R. Roy: Bi-level optimisation using genetic algorithm. In Proceedings 2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS 2002). IEEE (2002), 322–327.

[58] M. S. Osman, W. F. Abd El-Wahed, M. M. El Shafei and H. B. Abd El Wahab: An approach for solving multi-objective bi-level linear programming based on genetic algorithm. Journal of Applied Sciences Research, 6 (2010), 336–344.

[59] J. V. Outrata: On the numerical solution of a class of Stackelberg problems. Zeitschrift für Operations Research, 34(4) (1990), 255–277.

[60] T. Pham Dinh and H. A. Le Thi: Convex analysis approach to DC programming. Theory, algorithms and applications. Acta Mathematica Vietnamica, 22(1) (1997), 289–355.

[61] T. Pham Dinh and H. A. Le Thi: A DC optimization algorithm for solving the trust-region subproblem. SIAM Journal on Optimization, 8(2) (1998), 476–505.

[62] C. O. Pieume, P. Marcotte, L. P. Fotso and P. Siarry: Generating efficient solutions in bi-level multi-objective programming problems. American Journal of Operations Research, 3 (2013), 289.

[63] T. D. Quynh, P. D. Tao et al.: A DC programming approach for a class of bilevel programming problems and its application in portfolio selection. Numerical Algebra, Control and Optimization, 2(1) (2012), 167–185.

[64] M. S. Radjef and A. A. Radjef: Solving linear bilevel programming by DC algorithm. (2010).

[65] A. Ren and Y. Wang: A novel penalty function method for semi-vectorial bi-level programming problem. Applied Mathematical Modeling, 40 (2016), 135–149.

[66] S. Ruuska and K. Miettinen: Constructing evolutionary algorithms for bi-level multi-objective optimization. 2012 IEEE Congress on Evolutionary Computation (CEC-2012) (2012).

[67] C. Shi, J. Lu and G. Zhang: An extended Kth-best approach for linear bilevel programming. Applied Mathematics and Computation, 164(3) (2005), 843–855.

[68] K. Shimizu and E. Aiyoshi: A new computational method for Stackelberg and min-max problems by use of a penalty method. IEEE Transactions on Automatic Control, 26 (1981), 460–466.

[69] S. Srivastava and S. K. Sahana: Application of bat algorithm for transport network design problem. Applied Computational Intelligence and Soft Computing (2019).

[70] Q. Tang and Z. Fu: A bi-level programming model and algorithm for the static bike repositioning problem. Journal of Advanced Transportation (2019).

[71] G. M. Wang, X. J. Wang, Z. Wan et al.: An adaptive genetic algorithm for solving bi-level linear programming problem. Applied Mathematics and Mechanics, 28 (2007), 1605–1612.

[72] U. P. Wen and A. D. Huang: A simple tabu search method to solve the mixed-integer linear bilevel programming problem. European Journal of Operational Research, 88(3) (1996), 563–571.

[73] D. J. White and G. Anandalingam: A penalty function approach for solving bi-level linear programs. Journal of Global Optimization, 3 (1993), 397–419.

[74] M. Xu and J. J. Ye: A smoothing augmented Lagrangian method for solving simple bi-level programs. Computational Optimization and Applications, 59 (2014), 353–377.

[75] J. J. Ye and D. L. Zhu: Optimality conditions for bi-level programming problems. Optimization, 33 (1995), 9–27.

[76] J. J. Ye, X. Yuan, S. Zeng et al.: Difference of convex algorithms for bilevel programs with applications in hyperparameter selection. Mathematical Programming, 198(2) (2023), 1583–1616.

[77] Y. L. V.: An exact penalty function approach for solving the linear bi-level multi-objective programming problem. Filomat, 29 (2015), 773–779.

[78] T. Zhang, T. Hu, Y. Zheng and X. Guo: An improved particle swarm optimization for solving bi-level multi-objective programming problem. Journal of Applied Mathematics (2012).

[79] T. Zhang, Z. Chen and J. Chen: A cooperative coevolution PSO technique for complex bi-level programming problems and application to watershed water trading decision making problems. Journal of Nonlinear Sciences and Applications, 10 (2017), 2115–2132.

[80] Y. Zheng, Z. Wan and G. Wang: A fuzzy interactive method for a class of bi-level multi-objective programming problem. Expert Systems with Applications, 38 (2011), 10384–10388.

[81] L. H. Zhang, Q. Chen, Q. Zhang and Y. C. Jiao: Multi-objective differential evolution algorithm based on decomposition for a type of multi-objective bi-level programming problems. Knowledge-Based Systems, 107 (2016), 271–288.

[82] Z. Zhao and X. Gu: Particle swarm optimization based algorithm for bilevel programming problems. In Sixth International Conference on Intelligent Systems Design and Applications, Vol. 2. IEEE (2006), 951–956.

[83] S. Zhou, A. B. Zemkoho and A. Tin: BOLIB 2019: Bilevel Optimization LIBrary of test problems version 2. (2019).

[84] R. Ziadi and A. Bencherif-Madani: A mixed algorithm for smooth global optimization. Journal of Mathematical Modeling, 11 (2023), 207–228.

ملخص:

تتناول هذه الأطروحة تطوير طرق جديدة لحل مسائل البرمجة ذات المستويين والبرمجة متعددة المعايير، من خلال خوارزميات تعتمد على برمجة الفرق بين دالتين محدبتين مع التنظيم، إضافة إلى مقاربة ميتاهيوريستية تجمع بين تحسين سرب الجسيمات والذئب الرمادي مع استخدام منحنيات كثيفة لتبسيط الصياغة. وقد أظهرت النتائج العددية فعالية عالية من حيث الدقة وزمن الحساب مع توسيعها إلى مسائل متعددة الأهداف.

كلمات مفتاحية:
البرمجة ذات المستويين، التحسين متعدد المعايير، التحسين الشامل، المنحنيات الكثيفة α، الخوارزميات التطورية، برمجة الفرق بين دالتين محدبتين.

Abstract :

This thesis focuses on developing new methods for solving bilevel programming problems and multicriteria optimization problems, through algorithms based on Difference of Convex Functions (DC) programming with regularization, as well as a metaheuristic approach combining particle swarm optimization and grey wolf optimization, using dense curves to simplify the formulation. Numerical results demonstrate high effectiveness in terms of accuracy and computational time, and the approach is extended to multicriteria problems.

Keywords: Bilevel programming, multicriteria optimization, global optimization, α-dense curves, evolutionary algorithms, DC programming.

Résumé :

Cette thèse traite du développement de nouvelles méthodes pour résoudre les problèmes de programmation à deux niveaux et d'optimisation multicritère, à travers des algorithmes basés sur la programmation DC (différence de fonctions convexes) avec régularisation, ainsi qu'une approche métaheuristique combinant l'optimisation par essaim particulaire et l'optimisation par loup gris, en utilisant des courbes denses pour simplifier la formulation. Les résultats numériques démontrent une grande efficacité en termes de précision et de temps de calcul, et l'approche est étendue aux problèmes multicritère.

Mots-clés : Programmation à deux niveaux, optimisation multicritère, optimisation globale, courbes denses, algorithmes évolutionnaires, programmation DC, approche de régularisation.