Problemas de programación lineal multi-objetivo

Carlos Carazo de la Fuente

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: Pedro Mariano Mateo Collazos
26 de junio de 2022

Summary

In the real world, decisions are rarely made according to only one criterion, but to several. Besides, it is common to see that some of them interfere with the others, making it difficult to find a clear answer that satisfies every objective in a reasonable way. It is natural then to consider the multiobjective problems associated to these decisions, and to look for the most suitable solutions for them. This is what motivates the project's main theme: multiobjective optimization.

This topic belongs to the Operations Research field, and it was developed on the basis of what is taught in the subject under the same name. Hence, it is needed to know about convex sets, single-objective linear programming and duality, being especially important the understanding of how the simplex algorithm works. Basic knowledge about Java programming language (Arrays, classes, collections and interfaces) was also required in order to implement the algorithms explained later.

The purpose of this work is to increase the knowledge of the mentioned field of study, since multicriteria optimization is an important matter that can provide helpful solutions to real life problems, and cannot be addressed during the Mathematics degree due to a lack of time. It consists of a thorough study of the mentioned topic, including:

- A general introduction to multiobjective problems. The necessary elements in these are: the feasible set, which is the set of points that meet the constraints of the problem, the vector of objective functions, and the order relationship defined in the feasible set of the objective space. The latter will play a big role in these problems, by allowing us to compare different points so we can decide which ones are the most relevant.

  The concept of efficiency is probably the most important in the project. A solution in the feasible set will be efficient if there is no other feasible solution that is more optimal than this for every objective. The weakening or stiffening of this definition will lead to weak and strict efficiency, respectively. Finding the efficient solutions of a problem will be the ultimate goal.

- An approach to the most common methods to solve this type of problems. The first of all and the simplest one, weighted sum scalarization, is based on solving an uniobjective problem where the objective function is a linear combination of all the objective functions. The next one, ε-constraint method, consists in optimizing one of the objective functions while considering the rest as constraints of the problem. At last, the hybrid method will combine features of both of the previous ones.

- The particularization of some of the initial concepts and principles to linear problems, i.e., problems with linear objective functions and constraints.

- A review of a biobjective version of the simplex algorithm based in parametric programming, after a set of auxiliary results to help understand the way it works and to prove its mathematical correctness. It will consist of three phases:

  1. First of all, it is required to find a basic feasible solution of the uniobjective problem that appears considering only the first objective function. If it cannot be obtained directly, an auxiliary problem involving artificial variables will be solved.

2. After that, the usual simplex algorithm is used to get an optimal solution and an optimal base of the mentioned problem.
3. At last, we get all the efficient solutions and bases through parametric programming: These will be the solutions of the problem having $c(\lambda) := \lambda c^1 + (1-\lambda)c^2$ as the objective function, where $\lambda \in [0,1]$, and c^1, c^2 are the objective functions of the original problem.

To ilustrate its functioning, an example is solved step by step using this algorithm.

- The study of a general version of the simplex algorithm, valid for linear problems with two or more objective functions, along with the necessary theoretical background, with a special attention to how efficient bases are related to each other. As the previous one, it will be made up of three stages:

  1. The first basic feasible solution of the multiobjective problem has to be found, for which an auxiliary problem can be solved (In a similar way to the first phase of the biobjective simplex).
  2. That feasible solution is used to solve an auxiliary dual problem, which will provide a weighting vector. By multiplying this by the matrix of objective functions and considering the initial constraints, we get a uniobjective problem whose optimal solution and base (that can be obtained via the simplex algorithm, again) will be the first efficient solution and base of the initial problem.
  3. The last phase consists in the following: Auxiliary problems related to the base that has been gotten in the second phase have to be solved, to know which nonbasic variables can enter the base. Then, all the possible pivot steps are tested with the entering criterion. Every one that passes this test must be an efficient base too, so the associated solution is computed and stored (along with the base). Once this is done, the same steps have to be done for every other base obtained by a feasible pivot, until every efficient solution and base is found.

  The same example as before is solved manually, in order to help understand every phase of it.

After all these, both the biobjective and the general simplex algorithms are programmed in Java language so as to test their proper functioning while solving academic problems. At the end, several examples are solved using these implementations. The input data includes the constraints matrix and vector, and the vector of objective functions, and the output shows the efficients bases and solutions, besides the evaluation of the objective functions in these.

Due to space limitations, related topics such as the geometry of multiobjective linear programming efficient solutions could not be addressed.

Índice general

Capítulo 1

Optimización multiobjetivo

1.1. Introducción. Nociones de eficiencia

En el mundo real, a menudo se dan problemas que requieren minimizar o maximizar simultáneamente varias funciones objetivo, que a veces entran en conflicto unas con otras. La optimización multiobjetivo buscará resolver estos problemas, teniendo siempre en cuenta que, en general, no existirá una solución que optimice todos los objetivos (a diferencia de lo que ocurre en la optimización uniobjetivo, donde el valor óptimo, si se alcanza, es único). La solución habrá de elegirse, según convenga, de entre todas las que llamaremos soluciones eficientes. Para esta sección, la referencia principal serán los capítulos 1 y 2 del libro de Ehrgott [5].

Un problema de optimización multiobjetivo "$\min_{x \in \mathbf{X}}$" $f(x)$ quedará completamente caracterizado por:

- La *región factible* $\mathbf{X}$, que es el conjunto de puntos que satisface las restricciones del problema (Posibles desigualdades, igualdades y/o restricciones enteras).
- El vector *función objetivo* $f = (f_1, ..., f_p) : \mathbf{X} \to \mathbb{R}^p$, con $p > 1$. Como hemos comentado previamente, es aquí donde radica la principal diferencia con la optimización uniobjetivo, donde la función a minimizar es escalar.
- Una *relación de orden* "$\preceq$" en $\mathbb{R}^p$ (necesitamos establecer una relación de orden en $\mathbb{R}^p$ que nos permita comparar pares de elementos, y de esta forma poder calcular "$\min_{x \in \mathbf{X}}$" $f(x)$ de acuerdo a dicho orden).

Una vez establecidos estos elementos, se tratará de encontrar el punto o puntos de la región factible tales que su imagen por f es mínima según el orden $\preceq$ determinado, y seleccionar el óptimo de entre estos. Denotaremos por $\mathbf{Y} = f(\mathbf{X})$ al conjunto imagen de la región factible por la función f.

Veamos brevemente el concepto de relación de orden en un conjunto, y cuál es la que utilizaremos generalmente. Revisamos primero el concepto de relación binaria, del cual derivaremos el de relación de orden:

Definición 1. Dado un conjunto cualquiera $\mathbf{S}$, una relación binaria es un subconjunto $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$ de elementos que cumplen una determinada condición. Si dos elementos s^1, s^2 de $\mathbf{S}$ están relacionados, lo denotaremos como $s^1 \mathbf{R} s^2$.

Un tipo especial de relaciones binarias son los órdenes. A continuación mostraremos los dos que utilizaremos posteriormente: el orden parcial y el orden estricto.

Definición 2. Se dice que una relación binaria es de orden, y la denotaremos con $\preceq$ en lugar de $\mathbf{R}$, si es:

- Reflexiva, es decir, $s \preceq s$ para todo $s \in \mathbf{S}$

- Antisimétrica, es decir, si $s^1 \preceq s^2$ y $s^2 \preceq s^1$, entonces $s^1 = s^2$ (Para todo $s^1, s^2 \in \mathbf{S}$)
- Transitiva, es, decir, si $s^1 \preceq s^2$ y $s^2 \preceq s^3$, entonces $s^1 \preceq s^3$ (Para todo $s^1, s^2, s^3 \in \mathbf{S}$)

Definición 3. Se dice que una relación binaria es un orden estricto, y lo denotaremos con $<$, si es:

- Irreflexiva, es decir, $s \not< s$ para todo $s \in \mathbf{S}$
- Antisimétrica, es decir, si $s^1 < s^2$ y $s^2 < s^1$, entonces $s^1 = s^2$ (Para todo $s^1, s^2 \in \mathbf{S}$)
- Transitiva, es, decir, si $s^1 < s^2$ y $s^2 < s^3$, entonces $s^1 < s^3$ (Para todo $s^1, s^2, s^3 \in \mathbf{S}$)

Dado un orden estricto $<$ en $\mathbf{S}$, es posible construir un orden parcial $\leq$ en $\mathbf{S}$ de la siguiente manera:

$$a \leq b \Leftrightarrow a < b \;\; o \;\; a = b$$

Recíprocamente, dado un orden parcial $\leq$ en $\mathbf{S}$, se puede definir un orden estricto $<$ en $\mathbf{S}$ mediante:

$$a < b \Leftrightarrow a \leq b \;\; y \;\; a \neq b$$

Para elementos escalares, la relación de orden común es la de $\leq$, 'menor o igual que'. Pero como trataremos con funciones vectoriales, generalizamos la anterior a una relación de orden componente a componente, tal y como se muestra en la siguiente tabla:

Notación	Definición	Nombre
$y^1 \leqq y^2$	$y^1_k \leq y^2_k \;\; \forall k = 1, ..., p$	Orden componente a componente débil
$y^1 \leq y^2$	$y^1_k \leq y^2_k \;\; \forall k = 1, ..., p;\; y^1 \neq y^2$	Orden componente a componente
$y^1 < y^2$	$y^1_k < y^2_k \;\; \forall k = 1, ..., p$	Orden componente a componente estricto

Tabla 1.1: Órdenes en $\mathbb{R}^p$

Volviendo al problema de optimización multiobjetivo, los órdenes anteriores nos permiten definir los conceptos de solución eficiente, débilmente eficiente y estrictamente eficiente, o equivalentemente, óptimo de Pareto, óptimo de Pareto débil, y óptimo de Pareto estricto:

Definición 4. Una solución factible $\hat{x} \in \mathbf{X}$ se dice eficiente u óptimo de Pareto si no existe otra $x \in \mathbf{X}$ tal que $f(x) \leq f(\hat{x})$ (Orden $\leq$ de la tabla 1.1). Si $\hat{x}$ es eficiente, $f(\hat{x})$ se dice punto no dominado. Si x^1, $x^2 \in \mathbf{X}$ y $f(x^1) \leq f(x^2)$, decimos que x^1 domina a x^2 y que $f(x^1)$ domina a $f(x^2)$. El conjunto de todas las soluciones eficientes $\hat{x} \in \mathbf{X}$ se llama conjunto eficiente y se denota $\mathbf{X}_E$. El conjunto de todos los puntos no dominados $\hat{y} = f(\hat{x}) \in \mathbf{Y}$, donde $\hat{x} \in \mathbf{X}_E$, se llama conjunto no dominado y se denota $\mathbf{Y}_N$.

Definición 5. Una solución factible $\hat{x} \in \mathbf{X}$ se dice débilmente eficiente u óptimo de Pareto débil si no existe otra $x \in \mathbf{X}$ tal que $f(x) < f(\hat{x})$, es decir, $f_k(x) < f_k(\hat{x})$ para todo $k = 1, ..., p$ (Orden $<$ de la tabla 1.1). El punto $\hat{y} = f(\hat{x})$ se dice débilmente no dominado. Los conjuntos de puntos débilmente eficientes y débilmente no dominados se denotan $\mathbf{X}_{\omega E}$, y $\mathbf{Y}_{\omega N}$, respectivamente.

Definición 6. Una solución factible $\hat{x} \in \mathbf{X}$ se dice estrictamente eficiente u óptimo de Pareto estricto si no existe $x \in \mathbf{X}$, $x \neq \hat{x}$, tal que $f_k(x) \leqq f_k(\hat{x})$ para todo $k = 1, ..., p$ (Orden $\leqq$ de la tabla 1.1). El conjunto de puntos estrictamente eficientes se denota $\mathbf{X}_{sE}$.

Observación. De las definiciones anteriores se deduce que $\mathbf{Y}_N \subset \mathbf{Y}_{\omega N}$, y que $\mathbf{X}_{sE} \subset \mathbf{X}_E \subset \mathbf{X}_{\omega E}$.

Ejemplo 1. Si consideramos el conjunto $\mathbb{R}^2$, un punto $\hat{y} \in \mathbf{Y}$ será no dominado si no existe otro que esté a la vez "más abajo" y "más a la izquierda" que este. Gráficamente (Figura 1.1):

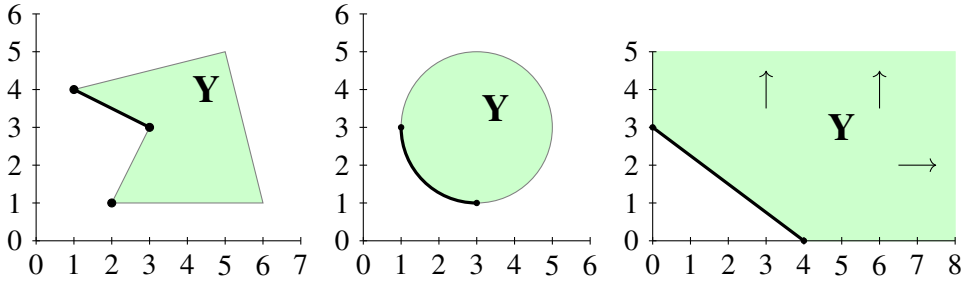

Figura 1.1: Puntos no dominados de varios conjuntos en $\mathbb{R}^2$

Ejemplo 2 (Ríos Insua, 1969). Sea el problema $\min_{x\in\mathbf{X}} f(x)$, donde $f:\mathbb{R}^2\to\mathbb{R}^2$ dada por $f(x_1,x_2)=(x_1,x_2)$ y $\mathbf{X}=\{(x_1,x_2)\in\mathbb{R}^2 : -4\le x_i\le 0 \text{ con } i=1,2 \text{ y } (x+2)^2+(y+2)^2\le 4 \text{ donde } -4\le x_1\le -2\}$. En este caso, notar que (Consultar figura 1.2):

$$\mathbf{X}_{sE}=\{(x_1,x_2)\in[-4,-2]\times[-4,-2] : (x+2)^2+(y+2)^2=4\}$$
$$\mathbf{X}_{\omega E}=\mathbf{X}_{sE}\bigcup\{(-4,x_2) : -2\le x_2\le 0\}\bigcup\{(x_1,-4) : -2\le x_1\le 0\}$$

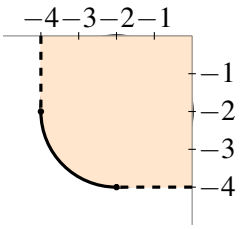

Figura 1.2: Puntos estrictamente (línea continua) y débilmente eficientes (línea discontinua) de $\mathbf{X}$

1.2. Métodos generales de resolución

En esta sección vamos a presentar varios métodos para resolver problemas de optimización lineal multiobjetivo basados en el enfoque tradicional, que consiste en realizar una escalarización del problema, esto es, reducirlo a otro uniobjetivo que esté de alguna manera relacionado con el problema original. Los tres métodos expuestos serán: el método de la suma ponderada, donde se buscará minimizar una combinación lineal de las componentes de la función objetivo, el método de las ε-restricciones, donde se minimizará uno de los objetivos y el resto serán añadidos como restricciones adicionales, y el método híbrido, que combinará las dos. Referimos al lector a los capítulos 3 y 4 del libro de Ehrgott [5].

1.2.1. Método de la suma ponderada

El método de la suma ponderada es el método más simple, y se basa en resolver el problema

$$\min_{x\in\mathbf{X}} \sum_{k=1}^{p} \lambda_k f_k(x), \tag{1.1}$$

donde $\lambda\in\mathbb{R}^p_{\geq}=\{\lambda\in\mathbb{R}^p : \lambda\geq 0\}$ es un vector de parámetros (pesos). Decimos que este nuevo problema es una suma ponderada del problema original. A continuación se presentan los resultados que nos permiten determinar qué condiciones deben cumplirse para que la resolución del problema anterior proporcione soluciones eficientes de nuestro problema, y recíprocamente, dada una solución eficiente, bajo qué condiciones será solución del problema (1.1).

Sea $\mathbf{Y} \subset \mathbb{R}^p$. Para un $\lambda \in \mathbb{R}^p_{\leqq}$ fijo, denotaremos por $\mathbf{S}(\lambda, \mathbf{Y}) := \{\hat{y} \in \mathbf{Y} : \langle \lambda, \hat{y} \rangle = \min_{y \in \mathbf{Y}} \langle \lambda, y \rangle\}$ al conjunto de puntos óptimos de $\mathbf{Y}$ con respecto a λ. Tras esto, definimos

$$\mathbf{S}(\mathbf{Y}) := \bigcup_{\lambda \in \mathbb{R}^p_{>}} \mathbf{S}(\lambda, \mathbf{Y})$$
$$\text{y } \mathbf{S}_0(\mathbf{Y}) := \bigcup_{\lambda \in \mathbb{R}^p_{\geq}} \mathbf{S}(\lambda, \mathbf{Y}).$$

Nótese que de la misma definición se sigue que $\mathbf{S}(\mathbf{Y}) \subset \mathbf{S}_0(\mathbf{Y})$.
En varios de los resultados necesitaremos suponer el siguiente tipo de convexidad:

Definición 7. Un conjunto $\mathbf{Y} \subset \mathbb{R}^p$ se dice $\mathbb{R}^p_{\leqq}$-convexo si $\mathbf{Y} + \mathbb{R}^p_{\leqq}$ es convexo.

Teorema 1.1. *Para cualquier conjunto $\mathbf{Y} \subset \mathbb{R}^p$, se tiene que $\mathbf{S}_0(\mathbf{Y}) \subset \mathbf{Y}_{\omega N}$.*

Demostración. Sea $\lambda \in \mathbb{R}^p_{\leq}$ e $\hat{y} \in \mathbf{S}(\lambda, \mathbf{Y})$. Entonces,

$$\sum_{k=1}^{p} \lambda_k \hat{y}_k \leq \sum_{k=1}^{p} \lambda_k y_k \text{ para todo } y \in \mathbf{Y}.$$

Suponer que $\hat{y} \notin \mathbf{Y}_{\omega N}$. Entonces, existe algún $y' \in \mathbf{Y}$ con $y'_k < \hat{y}_k$, $k = 1, ..., p$. Por tanto,

$$\sum_{k=1}^{p} \lambda_k y'_k \leq \sum_{k=1}^{p} \lambda_k \hat{y}_k,$$

porque al menos uno de los pesos λ_k debe ser positivo. Contradicción, de donde se sigue el resultado. □

Para conjuntos $\mathbb{R}^p_{\leqq}$-convexos, podemos probar la otra inclusión:

Teorema 1.2. *Si $\mathbf{Y}$ es $\mathbb{R}^p_{\leqq}$-convexo, entonces $\mathbf{Y}_{\omega N} = \mathbf{S}_0(\mathbf{Y})$.*

Demostración. Por el teorema 1.1 anterior, solo queda probar que $\mathbf{Y}_{\omega N} \subset \mathbf{S}_0(\mathbf{Y})$.
Notar que si $\hat{y} \in \mathbf{Y}_{\omega N}$, entonces $(Y + \mathbb{R}^p_{>} - \hat{y}) \cap (-\mathbb{R}^p_{>}) = \emptyset$. Por tratarse de dos conjuntos convexos disjuntos, podemos aplicar un teorema de separación y deducir que existe $\lambda \in \mathbb{R}^p \setminus \{0\}$ tal que $\langle \lambda, y + d - \hat{y} \rangle \geq 0 \geq \langle \lambda, -d' \rangle$ para todo $y \in \mathbf{Y}$, $d \in \mathbb{R}^p_{>}$, $d' \in \mathbb{R}^p_{>}$.

Como $\langle \lambda, -d' \rangle \leq 0$ para todo $d' \in \mathbb{R}^p_{>}$, podemos tomar $d' = e_k + \varepsilon e$, donde e_k es el k-ésimo vector de la base canónica, $e = (1, ..., 1)$ es un vector de unos y $\varepsilon > 0$ es arbitrariamente pequeño para deducir que $\lambda_k \geq 0$ para todo $k = 1, ..., p$.

Por otra parte, tomando $d = \varepsilon e$ en $\langle \lambda, y + d - \hat{y} \rangle \geq 0$ se tiene que $\langle \lambda, y \rangle + \varepsilon \langle \lambda, e \rangle \geq \langle \lambda, \hat{y} \rangle$ para todo $y \in \mathbf{Y}$ y por lo tanto $\langle \lambda, y \rangle > \langle \lambda, \hat{y} \rangle$. Así, $\lambda \in \mathbb{R}^p_{\leq}$ e $\hat{y} \in \mathbf{S}(\lambda, \mathbf{Y}) \subset \mathbf{S}_0(\mathbf{Y})$. □

A continuación, relacionamos $\mathbf{S}(\mathbf{Y})$ con $\mathbf{Y}_N$.

Teorema 1.3. *Sea $\mathbf{Y} \subset \mathbb{R}^p$. Entonces, $\mathbf{S}(\mathbf{Y}) \subset \mathbf{Y}_N$.*

Demostración. Sea $\hat{y} \in \mathbf{S}(\mathbf{Y})$. Entonces, existe algún $\lambda \in \mathbb{R}^p_{>}$ tal que $\sum_{k=1}^{p} \lambda_k \hat{y}_k \leq \sum_{k=1}^{p} \lambda_k y_k$ para todo $y \in \mathbf{Y}$.
Suponer que $\hat{y} \notin \mathbf{Y}_N$. Por lo tanto, debe existir un $y' \in \mathbf{Y}$ con $y' \leq y$. Multiplicando componente a componente por los pesos, se llega a que $\lambda_k y'_k \leq \lambda_k \hat{y}_k$ para todo $k = 1, ..., p$, con una desigualdad estricta en una componente. Esta desigualdad estricta, junto al hecho de que todos los λ_k son positivos, implica que $\sum_{k=1}^{p} \lambda_k y'_k < \sum_{k=1}^{p} \lambda_k \hat{y}_k$, contradiciendo que $\hat{y} \in \mathbf{S}(\mathbf{Y})$. □

Corolario 1.1. *Si $\mathbf{Y}$ es $\mathbb{R}^p_{\leqq}$-convexo, entonces $\mathbf{Y}_N \subset \mathbf{S}(\mathbf{Y})$.*

Demostración. Es una consecuencia inmediata del teorema 1.2, puesto que $\mathbf{Y}_N \subset \mathbf{Y}_{\omega N} = \mathbf{S}(\mathbf{Y})$. □

Proposición 1.1. *Si $\hat{y}$ es el único elemento de $\mathbf{S}(\lambda, \mathbf{Y})$ para algún $\lambda \in \mathbb{R}^p_{\geq}$, entonces $\hat{y} \in \mathbf{Y}_N$.*

Demostración. Fijamos el $\lambda \in \mathbb{R}^p_{\geq}$ del enunciado, y suponemos que $\hat{y} \notin \mathbf{Y}_N$. Entonces, existe algún $y' \in \mathbf{Y}$ con $y'_k < \hat{y}_k$ para todo $k = 1, ..., p$. Por tanto,

$$\sum_{k=1}^{p} \lambda_k y'_k \leq \sum_{k=1}^{p} \lambda_k \hat{y}_k,$$

porque $\lambda \in \mathbb{R}^p_{\geq}$. Como $\hat{y} \in \mathbf{S}(\lambda, \mathbf{Y})$, debe darse la igualdad en la expresión anterior. Contradicción con que $\hat{y}$ sea el único elemento de $\in \mathbf{S}(\lambda, \mathbf{Y})$, luego $\hat{y} \in \mathbf{Y}_N$. □

Proposición 1.2. *Suponer que $\hat{x} \in \mathbf{X}$ es una solución óptima del problema* (1.1)*, con $\lambda \in \mathbb{R}^p_{\geq}$. Entonces, se tiene que:*

1. *Si $\lambda \in \mathbb{R}^p_{\geq}$, entonces $\hat{x} \in \mathbf{X}_{\omega E}$.*
2. *Si $\lambda \in \mathbb{R}^p_{>}$, entonces $\hat{x} \in \mathbf{X}_E$.*
3. *Si $\lambda \in \mathbb{R}^p_{\geq}$ y $\hat{x}$ es la única solución óptima de la suma ponderada, entonces $\hat{x} \in \mathbf{X}_{sE}$.*

Demostración. Los apartados se siguen de los teoremas 1.1, 1.3, y la proposición 1.1 con la unicidad de $\hat{x}$, respectivamente. □

Proposición 1.3. *Sea $\mathbf{X}$ un conjunto convexo, y sean f_k funciones convexas, $k = 1, ..., p$. Si $\hat{x} \in \mathbf{X}_{\omega E}$, entonces existe algún $\lambda \in \mathbb{R}^p_{\geq}$ tal que $\hat{x}$ es una solución óptima de* (1.1)*.*

Demostración. Es consecuencia del teorema 1.2. □

1.2.2. Método de las ε-restricciones

El método de las ε-restricciones fue introducido en 1971 por Yacov Y. Haimes y otros [1], y junto con el anterior es el más conocido a la hora de resolver problemas de optimización multiobjetivo. La idea de este consiste en minimizar únicamente uno de los objetivos (componentes de la función objetivo) del problema original, añadiendo el resto como nuevas restricciones al problema. Es decir, pasaremos de resolver el problema $\underset{x \in \mathbf{X}}{min}(f_1(x), ..., f_p(x))$ a resolver:

$$\begin{aligned} &\min_{x \in \mathbf{X}} f_j(x) \\ &\text{sujeto a } \ f_k(x) \leq \varepsilon_k, k = 1, ..., p, k \neq j, \end{aligned} \tag{1.2}$$

donde $\varepsilon \in \mathbb{R}^p$. A continuación vamos a ver algunos resultados importantes en relación a este método:

Proposición 1.4. *Sea $\hat{x}$ una solución óptima de* (1.2) *para algún $j \in \{1, ..., p\}$. Entonces, $\hat{x}$ es débilmente eficiente.*

Demostración. Suponer que $\hat{x} \notin \mathbf{X}_{\omega E}$. Entonces, existe algún $x \in \mathbf{X}$ tal que $f_k(x) < f_k(\hat{x})$ para todo $k = 1, ..., p$. En particular, $f_j(x) < f_j(\hat{x})$. Como $f_k(x) < f_k(\hat{x}) \leq \varepsilon_k$ para $k \neq j$, la solución x es factible para el problema (1.2). Contradicción con que $\hat{x}$ sea una solución óptima de (1.2). Luego $\hat{x} \in \mathbf{X}_{\omega E}$. □

Proposición 1.5. *Sea $\hat{x}$ la única solución óptima de* (1.2) *para algún $j \in \{1, ..., p\}$. Entonces, $\hat{x}$ es estrictamente eficiente (y con ello, eficiente).*

Demostración. Suponer que existe algún $x \in \mathbf{X}$ tal que $f_k(x) \leq \varepsilon_k$ para todo $k \neq j$. Si además $f_j(x) \leq f_j(\hat{x})$, debe ocurrir que $f_j(x) = f_j(\hat{x})$, porque $\hat{x}$ es una solución óptima de (1.2). Luego x es una solución óptima de (1.2). Y como $\hat{x}$ era única, debe ser $x = \hat{x}$ y $\hat{x} \in \mathbf{X}_{sE}$. □

Teorema 1.4. *Una solución factible $\hat{x} \in \mathbf{X}$ es eficiente si y solo si existe un $\hat{\varepsilon} \in \mathbb{R}^p$ tal que $\hat{x}$ es una solución óptima de* (1.2) *para todo $j = 1, ..., p$.*

Demostración. $\Rightarrow$) Sea $\hat{\varepsilon} = f(\hat{x})$. Suponer que $\hat{x}$ no es una solución óptima de (1.2) para algún j. Entonces, debe haber algún $x \in \mathbf{X}$ con $f_j(x) < f_j(\hat{x})$ y $f_k(x) \leq \hat{\varepsilon}_k = f_k(\hat{x})$ para todo $k \neq j$, con lo que $\hat{x} \notin \mathbf{X}_E$.

$\Leftarrow$) Suponer que $\hat{x} \notin \mathbf{X}_E$. Entonces, hay algún índice $j \in \{1, ..., p\}$ y una solución factible $x \in \mathbf{X}$ tal que $f_j(x) < f_j(\hat{x})$ y $f_k(x) \leq \varepsilon_k$ para $k \neq j$. Por tanto, $\hat{x}$ no puede ser una solución óptima de (1.2) para ningún ε para el cual sea factible. □

Por último, vamos a ver un teorema que relaciona las soluciones del método de las ε-restricciones con las del método de la suma ponderada. Pero antes, veamos un resultado auxiliar del libro de Mangasarian [6], páginas 63-65:

Teorema 1.5. *Sea Γ un conjunto convexo no vacío en $\mathbb{R}^n$, sea f una función m-dimensional convexa en Γ, y sea h una función k-dimensional lineal en $\mathbb{R}^n$. Si el sistema $\begin{cases} f(x) < 0 \\ h(x) = 0 \end{cases}$ no tiene solución $x \in \Gamma$, entonces existen $p \in \mathbb{R}^m$ y $q \in \mathbb{R}^k$ tales que $p \geq 0$, $(p,q) \neq 0$, $pf(x) + qh(x) \geq 0$ para todo $x \in \Gamma$.*

Demostración. Sean los conjuntos $\Lambda(x) = \{(y,z) : y \in \mathbb{R}^m,\ z \in \mathbb{R}^k,\ y > f(x),\ z = h(x)\}$ para $x \in \Gamma$, y $\Lambda = \bigcup_{x \in \Gamma} \Lambda(x)$. Por hipótesis, Λ no contiene al origen $0 \in \mathbb{R}^{m+k}$. Y como $\Lambda(x)$ es convexo y f es una función convexa, entonces para $(y^1, z^1), (y^2, z^2) \in \Lambda(x)$ y $0 \leq \lambda \leq 1$ se tiene que:

$$(1-\lambda)y^1 + \lambda y^2 > (1-\lambda)f(x^1) + \lambda f(x^2) \geq f((1-\lambda)x^1 + \lambda x^2)$$
$$(1-\lambda)z^1 + \lambda z^2 > (1-\lambda)h(x^1) + \lambda h(x^2) \geq h((1-\lambda)x^1 + \lambda x^2)$$

Como Λ es un conjunto convexo que no contiene al origen, se sigue por el lema 3.2.2 del mismo libro [6] (página 47) que existen $p \in \mathbb{R}^m$ y $q \in \mathbb{R}^k, (p,q) \neq 0$ tales que $(u,v) \in \Lambda \Rightarrow pu + qv \geq 0$. Y como cada u_i puede tomarse tan grande como se desee, $p \geq 0$.
Sea $\varepsilon > 0$, $u = f(x) + e\varepsilon$, $v = h(x)$, $x \in \Gamma$, donde e es un vector de unos en $\mathbb{R}^m$. Por tanto, $(u,v) \in \Lambda(x) \subset \Lambda$, y $pu + qv = pf(x) + \varepsilon pe + qh(x) \geq 0$ para $x \in \Gamma$, de donde $pf(x) + qh(x) \geq -\varepsilon pe$ para $x \in \Gamma$.
Ahora, si $\inf_{x \in \Gamma}\{pf(x) + qh(x)\} = -\delta < 0$, tomando un ε tal que $\varepsilon pe < \delta$ se tiene que $\inf_{x \in \Gamma}\{pf(x) + qh(x)\} = -\delta < -\varepsilon pe$, que contradice que $pf(x) + qh(x) \geq -\varepsilon pe$ para todo $x \in \Gamma$. Luego $\inf_{x \in \Gamma}\{pf(x) + qh(x)\} \geq 0$. □

Teorema 1.6. *Sea $\mathbf{X} \subset \mathbb{R}^n$ un conjunto convexo, y sean $f_k : \mathbb{R}^n \to \mathbb{R}$ funciones convexas, $k = 1, ..., p$. Si el sistema $f_k < 0$, $k = 1, ..., p$ no tiene solución $x \in \mathbf{X}$, entonces existen $\lambda_k \geq 0$ con $\sum_{k=1}^p \lambda_k = 1$ tales que se satisface $\sum_{k=1}^p \lambda_k f_k(x) \geq 0$ para todo $x \in \mathbf{X}$.*

Demostración. Se sigue del teorema 1.5 anterior, eliminando la condición $h(x) = 0$. □

Teorema 1.7 (Chankong y Haimes, 1983).

1. *Suponer que $\hat{x}$ es una solución óptima de $min_{x \in \mathbf{X}} \sum_{k=1}^p \lambda_k f_k(x)$. Si $\lambda_j > 0$ para todo $j \in \{1, ...p\}$, existe $\hat{\varepsilon}$ tal que $\hat{x}$ también es una solución óptima de* (1.2).

2. *Suponer que $\mathbf{X}$ es un conjunto convexo y que $f_k : \mathbb{R}^n \to \mathbb{R}$ son funciones convexas. Si $\hat{x}$ es una solución óptima de* (1.2) *para algún $j \in \{1, ...p\}$, existe $\hat{\lambda} \in \mathbb{R}^p_{\geq}$ tal que $\hat{x}$ es una solución óptima de $min_{x \in \mathbf{X}} \sum_{k=1}^p \hat{\lambda}_k f_k(x)$.*

Demostración.

1. Como en la demostración anterior, vemos que podemos tomar $\hat{\varepsilon} = f(\hat{x})$. De la optimalidad de $\hat{x}$ para un problema de suma ponderada, tenemos que $\sum_{k=1}^p \lambda_k (f_k(x) - f_k(\hat{x})) \geq 0$ para todo $x \in \mathbf{X}$. Suponer que $\hat{x}$ no es óptima para (1.2) con $\hat{\varepsilon}$. La contradicción se sigue de que para cualquier $x' \in \mathbf{X}$ con $f_j(x') < f_j(\hat{x})$ y $f_k(x') \leq f_k(\hat{x})$ para $k \neq j$, se tiene que $\lambda_j (f_j(x') - f_j(\hat{x})) + \sum_{k \neq j} \lambda_k (f_k(x') - f_k(\hat{x})) < 0$, porque $\lambda_j > 0$ para todo $j \in \{1, ...p\}$.

2. Suponer que $\hat{x}$ es una solución óptima de (1.2). Entonces, no hay ningún $x \in \mathbf{X}$ satisfaciendo $f_j(x) < f_j(\hat{x})$ y $f_k(x) \leq f_k(\hat{x}) \leq \varepsilon_k$ para $k \neq j$. Por la convexidad de f_k, podemos aplicar el teorema 1.6 anterior y concluir que debe haber algún $\hat{\lambda} \in \mathbb{R}^p_{\geq}$ tal que $\sum_{k=1}^p \hat{\lambda}_k (f_k(x) - f_k(\hat{x})) \geq 0$ para todo $x \in \mathbf{X}$. Como $\hat{\lambda} \in \mathbb{R}^p_{\geq}$, tenemos que $\sum_{k=1}^p \hat{\lambda}_k f_k(x) \geq \sum_{k=1}^p \hat{\lambda}_k f_k(\hat{x})$ para todo $x \in \mathbf{X}$. Por tanto, $\hat{\lambda}$ es el vector de pesos buscado.

□

1.2.3. Método híbrido

Este método será una combinación de los dos anteriores: El problema a resolver será una suma ponderada, y además tendrá restricciones en todos los objetivos. Consideremos el siguiente problema:

$$\begin{aligned} &\min_{x \in \mathbf{X}} \sum_{k=1}^{p} \lambda_k f_k(x) \\ &\text{sujeto a } \; f_k(x) \leq f_k(x^0),\, k = 1, ..., p, \end{aligned} \tag{1.3}$$

donde $x \in \mathbf{X}$, $\lambda \in \mathbb{R}^p_{\geq}$ y x^0 es un punto factible arbitrario del problema original.

Teorema 1.8 (Guddat et al., 1985). *Sea $\lambda \in \mathbb{R}^p_{>}$. Una solución factible $x^0 \in \mathbf{X}$ es una solución óptima del problema* (1.3) *si y solo si $x^0 \in \mathbf{X}_E$.*

Demostración. Sea $x^0 \in \mathbf{X}$ eficiente. Entonces, no existe ningún $x \in \mathbf{X}$ tal que $f(x) \leq f(x^0)$. Por tanto, cualquier solución de (1.3) cumple que $f(x) = f(x^0)$ y es una solución óptima.

Recíprocamente, sea x^0 una solución óptima de (1.3). Si hubiera algún $x \in \mathbf{X}$ tal que $f(x) \leq f(x^0)$, los pesos positivos implicarían que $\sum_{k=1}^p \lambda_k f_k(x) < \sum_{k=1}^p \lambda_k f_k(x^0)$. Por tanto, x^0 es eficiente. □

Ejemplo 3. Resolvemos el siguiente problema de optimización bicriterio, propuesto al final del capítulo 4 del libro de Ehrgott [5], utilizando la versión de prueba del software Lingo 19.0 [11]:

$$\begin{aligned} &\min \; -6x_1 - 4x_2 \\ &\min \; -x_1 \\ &\text{sujeto a} \quad x_1 + x_2 \leq 100 \\ &\qquad\quad 2x_1 + x_2 \leq 150 \\ &\qquad\quad x_1, x_2 \geq 0. \end{aligned}$$

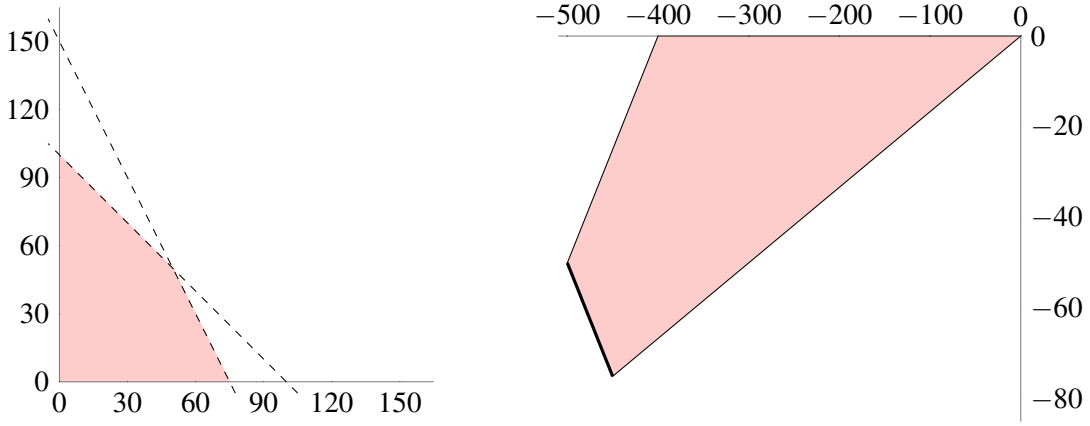

Figura 1.3: Conjunto factible del problema y su imagen por la función objetivo

Como puede apreciarse en la figura 1.3, el conjunto de puntos no dominados es el segmento que une los puntos (-500,-50) y (-450,-75).

- Por el método de la suma ponderada, el problema pasa a ser:

$$\begin{aligned} \min\ & \lambda_1(-6x_1-4x_2)+\lambda_2(-x_1) \\ \text{sujeto a}\ & x_1+x_2 \le 100 \\ & 2x_1+x_2 \le 150 \\ & x_1,x_2 \ge 0. \end{aligned}$$

  Con $\lambda_1=\lambda_2=\dfrac{1}{2}$ obtenemos la solución $(x_1,x_2)=(50,50)$ con valor $(-493,-50)$, y como $\lambda_1,\lambda_2>0$, por el apartado 2 de la proposición 1.2 sabemos que esta solución es eficiente para el problema original.

  Con $\lambda_1=\dfrac{1}{4},\ \lambda_2=\dfrac{3}{4}$ obtenemos la solución $(x_1,x_2)=(75,0)$ con valor $(-450,-75)$, que por el mismo motivo es eficiente para el problema original.

- Por el método de las ε-restricciones con $j=1$ y $\varepsilon=0$, el problema pasa a ser:

$$\begin{aligned} \min\ & -6x_1-4x_2 \\ \text{sujeto a}\ & x_1+x_2 \le 100 \\ & 2x_1+x_2 \le 150 \\ & -x_1 \le 0 \\ & x_1,x_2 \ge 0. \end{aligned}$$

  Obtenemos la solución $(x_1,x_2)=(50,50)$, que no es única porque alguno de los costos marginales finales es 0. Por la proposición 1.4, es al menos débilmente eficiente para el problema original (Sabemos que de hecho es eficiente, por el método anterior).

  De nuevo por este método, tomando $j=2$ y $\varepsilon=0$ se tiene el problema

$$\begin{aligned} \min\ & -x_1 \\ \text{sujeto a}\ & x_1+x_2 \le 100, \\ & 2x_1+x_2 \le 150 \\ & -6x_1-4x_2 \le 0 \\ & x_1,x_2 \ge 0 \end{aligned}$$

  cuya solución es $(x_1,x_2)=(75,0)$, la otra solución eficiente que ya conocemos.

- Por último, utilizando el método híbrido, resolvemos el siguiente problema habiendo tomado el punto factible arbitrario $x^0=(20,20)$:

$$\begin{aligned} \min\ & \lambda_1(-6x_1-4x_2)+\lambda_2(-x_1) \\ \text{sujeto a}\ & x_1+x_2 \le 100 \\ & 2x_1+x_2 \le 150 \\ & -6x_1-4x_2 \le -200 \\ & -x_1 \le -20 \\ & x_1,x_2 \ge 0. \end{aligned}$$

  Y de nuevo se obtienen las mismas soluciones eficientes: $(x_1,x_2)=(50,50)$ para $\lambda_1=\lambda_2=\dfrac{1}{2}$, y $(x_1,x_2)=(75,0)$ para $\lambda_1=\dfrac{1}{4},\ \lambda_2=\dfrac{3}{4}$.

Capítulo 2

Algoritmo simplex para problemas multiobjetivo lineales

2.1. Programación lineal multiobjetivo

Para el contenido de este capítulo, salvo la última sección, se toman como referencia los capítulos 6 y 7 del libro de Ehrgott [5]. Este estará dedicado a la aplicación del algoritmo simplex a problemas de optimización multiobjetivo, por lo que será imprescindible el conocimiento del mismo (y de otros temas como dualidad, estudiados en la asignatura *Investigación Operativa*). Para poder aplicarlo, estos deberán ser lineales, es decir, las funciones objetivo deberán ser de la forma $f_k(x) = c_k^T x,\ k = 1, ..., p$, con $c_k \in \mathbb{R}^n$, y el conjunto de restricciones $g_j(x) \leq 0$ de la forma $Ax \leq b$ (que quedará como $Ax = b$ tras la adición de lo que se denominan variables de holgura). Así pues, deberán tener el siguiente formato:

$$\begin{aligned} \min \quad & Cx \\ \text{sujeto a} \quad & Ax = b \\ & x \geqq 0 \end{aligned} \tag{2.1}$$

Notar que entonces el conjunto factible del espacio de soluciones será $\mathbf{X} = \{x \in \mathbb{R}^n : Ax = b,\ x \geqq 0\}$, y el conjunto factible en el espacio objetivo será $\mathbf{Y} = C\mathbf{X} = \{Cx : x \in \mathbf{X}\}$. Además, las nociones de eficiencia vistas en el capítulo anterior quedarán como sigue:

Definición 8. Sea $\hat{x} \in \mathbf{X}$ una solución factible del problema (2.1), e $\hat{y} = C\hat{x}$.

1. $\hat{x}$ se dice débilmente eficiente si no existe $x \in \mathbf{X}$ tal que $Cx < C\hat{x}$. En tal caso, $\hat{y} = C\hat{x}$ se dice débilmente no dominada.

2. $\hat{x}$ se dice eficiente si no existe $x \in \mathbf{X}$ tal que $Cx \leq C\hat{x}$. En tal caso, $\hat{y} = C\hat{x}$ se dice no dominada.

Nota. Si $\mathbf{X}_k := \{\hat{x} \in \mathbf{X} :\ c_k^T \hat{x} \leq c_k^T x \text{ para todo } x \in \mathbf{X}\}$ es el conjunto de soluciones óptimas de la k-ésima función objetivo, supondremos durante todo el capítulo que

$$\cap_{k=1}^{p} \mathbf{X}_k = \emptyset, \tag{2.2}$$

es decir, que no existe ninguna solución factible que optimice todos los objetivos simultáneamente. Así, problema (2.1) es verdaderamente un problema multiobjetivo.

2.2. Algoritmo simplex para problemas de optimización lineal biobjetivo

2.2.1. Programación lineal y eficiencia

Antes de presentar la versión del algoritmo simplex para problemas de optimización de dos componentes, debemos conocer algunos resultados teóricos.

Definimos la versión lineal del problema de la suma ponderada:

$$\begin{array}{ll} \min & \lambda^T Cx \\ \text{sujeto a} & Ax = b \\ & x \geqq 0, \end{array} \tag{2.3}$$

donde $\lambda \in \mathbb{R}^p_{\geq}$.

Teorema 2.1. *Sea $\hat{x} \in \mathbf{X}$ una solución óptima del problema* (2.3).

1. *Si $\lambda \geq 0$, entonces $\hat{x}$ es débilmente eficiente.*
2. *Si $\lambda > 0$, entonces $\hat{x}$ es eficiente.*

Demostración. Es un caso particular de la proposición 1.2. □

Lema 2.2. *Una solución factible $x^0 \in \mathbf{X}$ es eficiente si y solo si el problema lineal*

$$\begin{array}{ll} \max & e^T z \\ \text{sujeto a} & Ax = b \\ & Cx + Iz = Cx^0 \\ & x, z \geqq 0, \end{array} \tag{2.4}$$

donde $e^T = (1, \ldots, 1) \in \mathbb{R}^p$, $z \in \mathbb{R}^p_{\leqq}$ e I es la matriz identidad $p \times p$, tiene una solución óptima $(\hat{x}, \hat{z})$ con $\hat{z} = 0$.

Demostración. Sea $(x,z) \in \mathbf{X} \times \mathbb{R}^p_{\leqq}$ una solución factible de (2.4). Entonces se tiene que $Cx + Iz = Cx^0$, y por tanto $z = Cx^0 - Cx \geqq 0$ por la no negatividad de z. Si la $\hat{x}$ de una solución óptima $(\hat{x}, \hat{z})$ es eficiente, entonces no existe $x \in \mathbf{X}$ tal que $Cx \leqq C\hat{x}$, por lo que debe ser $\hat{z} = 0$. Por otra parte, si $\hat{x}$ no es eficiente, debe existir $x \in \mathbf{X}$ tal que $Cx \leq Cx^0$. Pero entonces hay una z con $z_k > 0$ para al menos una k, contradiciendo la optimalidad de $(\hat{x}, 0)$. □

Lema 2.3. *Una solución factible $x^0 \in \mathbf{X}$ es eficiente si y solo si el problema lineal*

$$\begin{array}{ll} \min & u^T b + w^T Cx^0 \\ \text{sujeto a} & u^T A + w^T C \geqq 0 \\ & w \geqq e \\ & u \in \mathbb{R}^m \end{array} \tag{2.5}$$

tiene una solución óptima $(\hat{u}, \hat{w})$ con $\hat{u}^T b + \hat{w}^T Cx^0 = 0$.

Demostración. Nótese que (2.5) es el dual del problema (2.4). Por tanto, $(\hat{x}, \hat{z})$ es una solución óptima de (2.4) si y solo si el problema (2.5) tiene una solución óptima $(\hat{u}, \hat{w})$ tal que $e^T \hat{z} = \hat{u}^T b + \hat{w}^T Cx^0 = 0$. □

Una vez visto el lema 2.3, podemos probar que todas las soluciones de un problema (2.1) pueden ser encontradas resolviendo un problema de suma ponderada (2.3):

Teorema 2.4 (Isermann, 1974). *Una solución factible $x^0 \in \mathbf{X}$ es una solución eficiente del problema de optimización lineal multiobjetivo* (2.1) *si y solo si existe $\lambda \in \mathbb{R}^p_{>}$ tal que $\lambda^T Cx^0 \leq \lambda^T Cx$ para todo $x \in \mathbf{X}$.*

Demostración. Es consecuencia de las proposiciones 1.2 y 1.3. Veamos también una demostración con argumentos de dualidad:
$\Leftarrow$) Sabemos por el teorema 2.1 que una solución óptima de un problema lineal de suma ponderada con

pesos positivos es eficiente.

$\Rightarrow$) Sea $x^0 \in \mathbf{X}_E$. Del lema 2.3 se sigue que el problema (2.5) tiene una solución óptima $(\hat{u}, \hat{w})$ tal que

$$\hat{u}^T b = -\hat{w}^T C x^0. \tag{2.6}$$

Es fácil de ver que esta misma $\hat{u}$ es también una solución óptima del problema

$$\min \{u^T b : u^T A \geqq -\hat{w}^T C\}, \tag{2.7}$$

que es justo (2.5) con $w = \hat{w}$ fijado. Por tanto, existe una solución óptima de

$$\max \{-\hat{w}^T C x : Ax = b,\ x \geqq 0\}, \tag{2.8}$$

que es el dual de (2.7). Como por dualidad débil $u^T b \geq -\hat{w}^T C x$ para todas las soluciones factibles u de (2.7) y todas las soluciones factibles x de (2.8), y ya sabemos que $\hat{u}^T b = -\hat{w}^T C x^0$ por (2.6), se sigue que x^0 es una solución óptima de (2.8). Finalmente, notamos que (2.8) es equivalente a

$$\min \{\hat{w}^T C x : Ax = b,\ x \geqq 0\},$$

y que $\hat{w} \geqq e > 0$ por las restricciones de (2.5). Por lo tanto, x^0 es una solución óptima de la suma ponderada (2.3) con $\lambda = \hat{w}$ como vector de pesos. □

Proposición 2.1. *Sea $x^0 \in \mathbf{X}$. Entonces, el problema* (2.4) *es factible y se cumple lo siguiente:*

1. *Si $(\hat{x}, \hat{z})$ es una solución óptima de* (2.4)*, entonces $\hat{x}$ es una solución eficiente del problema multi-objetivo* (2.1)*.*
2. *Si* (2.4) *es no acotado, entonces $\mathbf{X}_E = \emptyset$.*

Demostración. Si x^0 es una solución factible de (2.1), entonces $Ax^0 = b$. Tomando también $z = 0$, se tiene que (x^0, z) cumple todas las restricciones de (2.4), con lo que el conjunto factible de este problema es no vacío, es decir, el problema es factible.

1. Sea $(\hat{x}, \hat{z})$ es una solución óptima de (2.4). Como se ha visto en la demostración del lema 2.2, debe ocurrir que $\hat{z} = 0$, por ser $(\hat{x}, \hat{z})$ óptima para (2.4). Luego, por este mismo lema, x^0 será eficiente para (2.1), y así $C\hat{x} = Cx^0 - I\hat{z} = Cx^0$, con lo que $\hat{x}$ es eficiente por serlo x^0.
2. Si (2.4) es no acotado, entonces existe alguna componente z_k de z tal que $z_k \to \infty$. Por tanto, no existe una solución óptima $(\hat{x}, \hat{z})$ con $\hat{z} = 0$, luego por el lema 2.2, se tiene que x^0 no es eficiente para (2.1) $\forall\, x^0 \in \mathbf{X}$, es decir, $\mathbf{X}_E = \emptyset$.

□

2.2.2. Algoritmo simplex para problemas biobjetivo

Tras ver los resultados anteriores, podemos extender el algoritmo simplex a problemas lineales con funciones objetivo de dos componentes, mediante programación lineal paramétrica:

Sea el problema lineal biobjetivo

$$\begin{array}{ll} \min & ((c^1)^T x, (c^2)^T x) \\ \text{sujeto a} & Ax = b \\ & x \geqq 0 \end{array} \tag{2.9}$$

Por el teorema 2.4, sabemos que encontrar las soluciones eficientes de (2.9) es equivalente a resolver el problema lineal

$$\min \{\lambda_1 (c^1)^T x + \lambda_2 (c^2)^T x : \ Ax = b, \ x \geqq 0\}$$

para todo $\lambda \in \mathbb{R}^2_{>}$. Sin pérdida de generalidad (dividiendo la función objetivo por $\lambda_1 + \lambda_2$), podemos suponer que $(\lambda_1, \lambda_2) = (\lambda, 1-\lambda)$. Definiendo la función objetivo paramétrica

$$c(\lambda) := \lambda c^1 + (1-\lambda) c^2,$$

tendremos que resolver

$$\min \{c(\lambda)^T x : \ Ax = b, \ x \geqq 0\}, \tag{2.10}$$

que es un problema lineal paramétrico. Buscamos determinar qué valores de λ son relevantes, para lo cual comenzamos considerando una base factible $\mathfrak{B}$ y el vector de costos reducidos de la función objetivo paramétrica

$$\bar{c}(\lambda) := \lambda \bar{c}^1 + (1-\lambda) \bar{c}^2. \tag{2.11}$$

Suponer que $\hat{\mathfrak{B}}$ es una base óptima para el problema lineal (2.10) para algún $\lambda = \hat{\lambda}$. Entonces, debe ser $\bar{c}(\lambda) \geqq 0$. Distinguimos dos casos:
Si $\bar{c}^2 \geq 0$, entonces de (2.11) se sigue que $\bar{c}(\lambda) \geqq 0$ para todo $\lambda < \hat{\lambda}$ y así $\hat{\mathfrak{B}}$ es una base óptima para todo $0 \leq \lambda \leq \hat{\lambda}$.
En otro caso, existe al menos un $i \in \mathbf{N}$ tal que $\bar{c}^2_i < 0$. Por tanto, mientras el problema no sea degenerado (esto es, no existan soluciones múltiples), hay algún valor $\lambda < \hat{\lambda}$ tal que $\bar{c}(\lambda)_i = 0$, es decir:

$$\lambda \bar{c}^1_i + (1-\lambda)\bar{c}^2_i = 0 \ \Rightarrow \ \lambda(\bar{c}^1_i - \bar{c}^2_i) + \bar{c}^2_i = 0 \ \Rightarrow \ \lambda = \frac{-\bar{c}^2_i}{\bar{c}^1_i - \bar{c}^2_i}.$$

Definimos $\mathfrak{I} = \{i \in \mathbf{N} : \ \bar{c}^2_i < 0, \ \bar{c}^1_i \geq 0\}$, y

$$\lambda' := \max_{i \in \mathfrak{I}} \frac{-\bar{c}^2_i}{\bar{c}^1_i - \bar{c}^2_i} \tag{2.12}$$

La base $\mathfrak{B}$ es óptima para el problema (2.10) para todo $\lambda \in [\lambda', \hat{\lambda}]$. En cuanto $\lambda \leq \lambda'$, la base óptima cambia, por lo que la variable x_s que entra en la base tiene que ser elegida tal que el máximo en (2.12) se alcanza para $i = s$.

Por lo anterior, podemos resolver (2.10) primero para $\lambda = 1$ (asumiendo que tiene solución), y luego, de forma iterativa, ir encontrando variables que entran en la base y nuevos valores λ según (2.12), hasta que $\bar{c}^2 \geqq 0$. Notar que si $\mathfrak{B}$ es una base óptima para el problema con $\lambda = 1$, entonces no puede serlo para $\lambda = 0$, por la suposición (2.2) hecha al comienzo del capítulo. Luego inicialmente $\mathfrak{I} \neq \emptyset$.
En cada iteración, para determinar una nueva base óptima, se elige como columna pivote (variable que entra x_s) el índice s para el cual se alcanza el valor crítico λ' de (2.12). La fila pivote (variable que sale x_r) se elige mediante el mismo criterio cociente del algoritmo simplex para un objetivo. La variable x_s se cambia por x_r en la base. De este modo, se genera una secuencia de valores críticos λ tales que $1 = \lambda^1 > ... > \lambda^l = 0$, y bases óptimas $\mathfrak{B}^1, ..., \mathfrak{B}^{l-1}$ que definen soluciones factibles básicas de (2.10) para todo λ: $\mathfrak{B}^i$ es una base óptima de (2.10) para todo $\lambda \in [\lambda^i, \lambda^{i+1}]$, $i = 1, ...l$.

En la práctica, resolveremos el siguiente problema auxiliar al comienzo del algoritmo:

$$\begin{array}{ll} \min & e^T z \\ \text{sujeto a} & Ax + z = b \\ & x, z \geqq 0 \end{array} \tag{2.13}$$

Este nos proporcionará una base óptima de la que partir, o nos hará llegar a la conclusión de que el problema es no factible.

Presentamos así un esquema del algoritmo simplex para problemas bicriterio lineales:

Input: Datos A, b, C de un problema lineal biobjetivo.
Output: Secuencia de λ-valores y soluciones eficientes.
Fase 1: Resolver el problema auxiliar (2.13) usando el algoritmo simplex.
if *el valor óptimo es positivo* **then**
  PARAR, $\mathbf{X} = \emptyset$.
else
  sea $\mathfrak{B}$ una base óptima.
end
Fase 2: Resolver el problema auxiliar (2.10) para $\lambda = 1$ partiendo de la base $\mathfrak{B}$ encontrada en la Fase 1, produciendo una base óptima $\hat{\mathfrak{B}}$. Recalcular $\tilde{A}$ y $\tilde{b}$.
Fase 3:
while $\mathfrak{I} = \{i \in \mathbf{N} : \bar{c}_i^2 < 0, \bar{c}_i^1 \geq 0\} \neq \emptyset$ **do**
  $\lambda := \max_{i \in \mathfrak{I}} \frac{-\bar{c}_i^2}{\bar{c}_i^1 - \bar{c}_i^2}$
  $s \in \operatorname{argmin}\{i \in \mathfrak{I} : \frac{-\bar{c}_i^2}{\bar{c}_i^1 - \bar{c}_i^2}\}$
  $r \in \operatorname{argmin}\{j \in \mathfrak{B} : \frac{\tilde{b}_j}{\tilde{A}_{sj}},\ \tilde{A}_{sj} > 0\}$
  Tomar $\mathfrak{B} := (\mathfrak{B} \backslash \{r\}) \cup \{s\}$ y actualizar $\tilde{A}$ y $\tilde{b}$.
end

Algoritmo 1: Simplex paramétrico para problemas lineales biobjetivo

2.2.3. Un ejemplo práctico

Ilustramos el funcionamiento del algoritmo anterior aplicándolo al siguiente problema tomado de un artículo relacionado con el tema [9]:

$$\begin{aligned} &\min\ -3x_1 - x_2 \\ &\min\ -x_1 - 4x_2 \\ &\text{sujeto a}\quad -x_1 + x_2 \leq 2 \\ &\qquad\qquad x_1 + x_2 \leq 7 \\ &\qquad\qquad x_1 + 2x_2 \leq 10 \\ &\qquad\qquad x_1, x_2 \geq 0. \end{aligned}$$

Tras añadir las variables de holgura (una sumando en cada restricción, por tratarse de restricciones de tipo $\leq$, y siempre con costo $\bar{0}$), los datos del problema quedarán:

$$A = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 7 \\ 10 \end{pmatrix}, \qquad C = \begin{pmatrix} -3 & -1 & 0 & 0 & 0 \\ -1 & -4 & 0 & 0 & 0 \end{pmatrix}.$$

Fase 1: Puesto que ya encontramos columnas de A que formen la matriz identidad (columnas 3,4 y 5), no será necesario aplicar la fase 1 del algoritmo, pues $\mathbf{x}^T = (0,0,2,7,10)$ (base $\{x_3, x_4, x_5\}$) ya será una solución factible básica del problema $\{\min\ c^1,\ \text{s.a.}\ Ax = b\}$.

Fase 2: Podemos entonces resolver este problema aplicando el algoritmo simplex habitual:

	x_1	x_2	x_3	x_4	x_5	
x_3	-1	1	1	0	0	2
x_4	1	1	0	1	0	7
x_5	1	2	0	0	1	10
	-3	-1	0	0	0	

	x_1	x_2	x_3	x_4	x_5	
x_3	0	2	1	1	0	9
x_1	1	1	0	1	0	7
x_5	0	1	0	-1	1	3
	0	2	0	3	0	

Fase 3: Comenzamos así la tercera fase del algoritmo:

Iteración 1: Actualizamos la segunda fila de costos marginales según esta base óptima (la primera fila ya lo está):

$$\bar{c}^2 = c^2 - c^2_{\mathfrak{B}} A^{-1}_{\mathfrak{B}} A = (-1,-4,0,0,0) - (0,-1,0) \begin{pmatrix} 0 & 2 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 1 \end{pmatrix} = (0,-3,0,1,0)$$

$\mathfrak{I} = \{i \in \mathbf{N} : \bar{c}^2_i < 0, \bar{c}^1_i \geq 0\} = \{2\}$

$\lambda = \max_{i \in \mathfrak{I}} \frac{-\bar{c}^2_i}{\bar{c}^1_i - \bar{c}^2_i} = \frac{-(-3)}{2-(-3)} = \frac{3}{5}$

$s = \text{argmin}\{i \in \mathfrak{I} : \frac{-\bar{c}^2_i}{\bar{c}^1_i - \bar{c}^2_i}\} = 2 \Rightarrow$ La variable x_2 entra en la base

$r = \text{argmin}\{j \in \mathfrak{B} : \frac{\tilde{b}_j}{\tilde{A}_{sj}}, \tilde{A}_{sj} > 0\} = \text{argmin}\{\frac{9}{2}\ (j=3), \frac{3}{1} = 3\ (j=5)\} = 5 \Rightarrow$ La variable x_5 sale de la base, con lo que la nueva base será $\{x_3, x_1, x_2\}$
Actualizamos los datos del problema tras esta iteración:

	x_1	x_2	x_3	x_4	x_5	
x_3	0	0	1	3	-2	3
x_1	1	0	0	2	-1	4
x_2	0	1	0	-1	1	3
	0	0	0	5	-2	
	0	0	0	-2	3	

Iteración 2: $\mathfrak{I} = \{4\}$

$\lambda = \frac{-(-2)}{5-(-2)} = \frac{2}{7}$

$s = 4 \Rightarrow$ La variable x_4 entra en la base

$r = \text{argmin}\{\frac{3}{3} = 1\ (j=3), \frac{4}{2} = 2\ (j=1)\} = 3 \Rightarrow$ La variable x_3 sale de la base, y así la nueva base será $\{x_4, x_1, x_2\}$ Actualizamos los datos del problema tras esta iteración:

	x_1	x_2	x_3	x_4	x_5	
x_4	0	0	1/3	1	-2/3	1
x_1	1	0	-2/3	0	1/3	2
x_2	0	1	1/3	0	1/3	4
	0	0	-5/3	0	4/3	
	0	0	2/3	0	5/3	

Iteración 3: $\mathfrak{I} = \emptyset$, el algoritmo se detiene.

Se tiene entonces que las bases eficientes del problema y sus respectivas soluciones eficientes son:

- $\{x_3, x_1, x_5\}$, $\mathbf{x}^T = (7,0,9,0,3), \ \forall \lambda \in [\frac{3}{5}, 1]$
- $\{x_3, x_1, x_2\}$, $\mathbf{x}^T = (4,3,3,0,0), \ \forall \lambda \in [\frac{2}{7}, \frac{3}{5}]$
- $\{x_4, x_1, x_2\}$, $\mathbf{x}^T = (2,4,0,1,0), \ \forall \lambda \in [0, \frac{2}{7}]$

Para ubicar las soluciones en el conjunto factible original, ignórense las componentes correspondientes a las variables de holgura (3, 4 y 5).

2.3. Algoritmo simplex general para problemas de optimización lineal multiobjetivo

Esta sección está dedicada a presentar el algoritmo general para el cálculo de soluciones factibles básicas eficientes cuando el problema a optimizar tiene más de 2 objetivos. Obviamente, este algoritmo también será válido para resolver problemas biobjetivo, pero ante un problema de este tipo resultará más conveniente utilizar el anterior por su mayor sencillez (y menor coste computacional).

2.3.1. Bases eficientes y relaciones entre ellas

En primer lugar, estudiamos cómo las bases eficientes de un problema están relacionadas entre sí. Utilizaremos la notación $\bar{C} = C - C_{\mathfrak{B}}A_{\mathfrak{B}}^{-1}A$ para la matriz de costos marginales con respecto a la base $\mathfrak{B}$, y $R := \bar{C}_N$ para referirnos a la matriz formada por las columnas no básicas de la anterior. Del mismo modo que en el algoritmo simplex, se tendrá que $\bar{C}_{\mathfrak{B}}$, la parte básica de $\bar{C}$, será la matriz nula.

Definición 9. Una base factible $\mathfrak{B}$ se dice eficiente si es una base óptima de (2.3) para algún $\lambda \in \mathbb{R}^p_>$.

A continuación se presentan resultados enfocados al pivotaje entre bases eficientes. Diremos que un pivote es factible si la solución que se obtiene tras el pivotaje es factible.

Definición 10. Dos bases $\mathfrak{B}$ y $\tilde{\mathfrak{B}}$ se dicen adyacentes si una puede ser obtenida a partir de la otra mediante un pivotaje de un paso.

Sea $\lambda^T R \geqq 0$, $\lambda^T r^j$ la forma general del sistema de ecuaciones que usamos en la sección anterior para calcular los valores críticos λ, de los cuales derivábamos (2.12). Escogíamos s tal que $\bar{c}(\lambda) \geq 0$, $\bar{c}(\lambda)_s = 0$.

Definición 11. 1. Sea $\mathfrak{B}$ una base eficiente, y $\mathbf{N}$ el conjunto de índices de las variables no básicas. Una variable x_j, $j \in \mathbf{N}$ se dice eficiente no básica en $\mathfrak{B}$ si existe un $\lambda \in \mathbb{R}^p_>$ tal que $\lambda^T R \geqq 0$ y $\lambda^T r^j = 0$, donde r^j es la columna de R correspondiente a la variable x_j.
2. Sea $\mathfrak{B}$ una base eficiente y x_j una variable eficiente no básica. Un pivote factible de $\mathfrak{B}$, con x_j la variable que sale, se dice pivote eficiente con respecto a $\mathfrak{B}$ y x_j.

Proposición 2.2. *Sea $\mathfrak{B}$ una base eficiente. Entonces, existe una variable eficiente no básica en $\mathfrak{B}$.*

Demostración. Como $\mathfrak{B}$ es una base eficiente, existe algún $\lambda > 0$ tal que $\lambda^T R \geqq 0$. Por lo tanto, el conjunto $\mathfrak{L} := \{\lambda > 0 : \lambda^T R \geqq 0\}$ es no vacío. Tenemos que ver que existe algún $\lambda \in \mathfrak{L}$ y $j \in \mathbf{N}$ tal que $\lambda^T r^j = 0$.
Observamos en primer lugar que no hay ninguna columa r de R tal que $r \leq 0$. También debe haber al menos una columna con elementos positivos y negativos, por la suposición (2.2). Sea ahora $\lambda^* \in \mathfrak{L}$, en particular $\lambda^T R \geqq 0$. Sea $\lambda' \in \mathbb{R}^p_>$ tal que $\mathfrak{I} := \{i \in \mathbf{N} : \lambda'^T r^j < 0\} \neq \emptyset$. Tal λ debe existir, pues R tiene al menos un elemento negativo.
Definimos $\phi : \mathbb{R} \to \mathbb{R}^{|N|}$ mediante

$$\phi_i(t) := (t(\lambda^{*T} + (1-t)\lambda'^T)r^i,\ i \in \mathbf{N}.$$

Por tanto, $\phi(0) = \lambda'^T R$ y $\phi(1) = \lambda^{*T} R \geqq 0$. Para cada $i \in \mathbf{N}\backslash\mathfrak{I}$ se tiene que $\phi_i(t) \geq 0$ para todo $t \in [0,1]$. Para todo $i \in \mathfrak{I}$ existe algún $t_i \in [0,1]$ tal que

$$\phi_i(t) \begin{cases} < 0 & si \quad t \in [0, t_i) \\ = 0 & si \quad t = t_i \\ \geq 0 & si \quad t \in [t_i, 1]. \end{cases}$$

Con $t^* = \max\{t_i : i \in \mathfrak{I}\}$, tenemos que $\phi_i(t^*) \geqq 0$ y que $\phi_i(t^*) = 0$ para algún $i \in \mathfrak{I}$. Luego $\hat{\lambda} := t\lambda^* + (1-t)\lambda' \in \mathfrak{L}$, concluyendo la demostración. □

Lema 2.5. *Sea $\mathfrak{B}$ una base eficiente y x_j una variable eficiente no básica. Entonces, cualquier pivote eficiente de $\mathfrak{B}$ lleva a una base eficiente adyacente $\hat{\mathfrak{B}}$.*

Demostración. Sea x_j la variable que entra en la base $\mathfrak{B}$. Como x_j es una variable eficiente no básica, existe $\lambda \in \mathbb{R}^p_>$ con $\lambda^T R \geqq 0$ y $\lambda^T r^j = 0$. Luego x_j es una variable no básica con costo marginal 0 en (2.3), y por tanto los costos marginales de (2.3) no cambiarán cuando x_j entre en la base. Sea $\hat{\mathfrak{B}}$ la base resultante tras cualquier pivotaje factible sobre $\mathfrak{B}$ donde x_j es la variable que entra. Entonces $\lambda^T R \geqq 0$ y $\lambda^T r^j = 0$ para $\hat{\mathfrak{B}}$, con lo que $\hat{\mathfrak{B}}$ es una base óptima para (2.3) y por tanto una base eficiente adyacente. □

Necesitamos un método para comprobar si una variable no básica x_j de una base eficiente $\mathfrak{B}$ es eficiente. Este consistirá en resolver un problema lineal:

Teorema 2.6 (Evans y Steuer, 1973). *Sea $\mathfrak{B}$ una base eficiente y x_j una variable no básica. Entonces, la variable x_j es eficiente no básica si y solo si el problema lineal*

$$\begin{aligned} \max \quad & e^T v \\ \text{sujeto a} \quad & Rz - r^j\delta + Iv = 0 \\ & z, \delta, v \geqq 0 \end{aligned} \tag{2.14}$$

tiene valor óptimo 0.

Demostración. Por la definición 11, x_j es una variable eficiente no básica si el problema lineal

$$\begin{aligned} \min \quad & 0^T\lambda = 0 \\ \text{sujeto a} \quad & R^T\lambda \geqq 0 \\ & (r^j)^T\lambda = 0 \\ & I\lambda \geq e \\ & \lambda \geqq 0 \end{aligned} \tag{2.15}$$

tiene valor óptimo 0, es decir, si es factible. Las dos primeras restricciones de (2.15) juntas son equivalentes a $R^T\lambda \geqq 0,\ (r^j)^T\lambda \leq 0$, o equivalentemente $R^T\lambda \geqq 0,\ (-r^j)^T\lambda \geq 0$, lo que resulta en el problema

$$\begin{aligned} \min \quad & 0^T\lambda = 0 \\ \text{sujeto a} \quad & R^T\lambda \geqq 0 \\ & -(r^j)^T\lambda \geq 0 \\ & I\lambda \geq e \\ & \lambda \geqq 0 \end{aligned} \tag{2.16}$$

El problema dual de (2.16) es

$$\begin{aligned} \max \quad & e^T v \\ \text{sujeto a} \quad & Rz - r^j\delta + Iv + It = 0 \\ & z, \delta, v, t \geqq 0, \end{aligned} \tag{2.17}$$

y puesto que una solución óptima de (2.17) siempre tendrá a t con valor 0, este problema es equivalente a

$$\begin{aligned} \max \quad & e^T v \\ \text{sujeto a} \quad & Rz - r^j\delta + Iv = 0 \\ & z, \delta, v \geqq 0, \end{aligned} \tag{2.18}$$

que es el del enunciado. □

Es importante notar que el problema de prueba (2.14) siempre es factible, puesto que siempre puede tomarse $(z, \delta, v) = 0$. La demostración anterior también muestra que (2.14) solo puede tener una solución óptima con $v = 0$ (el valor objetivo de (2.15) es 0), o ser no acotado. Con esta observación, concluimos que:

- x_j es un variable eficiente no básica si y solo si (2.14) es acotado y tiene valor óptimo 0.
- x_j es una variable "no eficiente" no básica si y solo si (2.14) es no acotado.

El algoritmo simplex funciona moviéndose a lo largo de bases adyacentes hasta que se encuentra una óptima. Buscamos aprovechar este principio para identificar todas las bases eficientes, es decir, queremos movernos de una base eficiente a otra. Por tanto, tenemos que probar que efectivamente podemos restringirnos únicamente a bases adyacentes, o lo que es lo mismo, que las bases eficientes están conectadas (en términos de adyacencia).

Definición 12. Dos bases eficientes $\mathfrak{B}$ y $\hat{\mathfrak{B}}$ se dicen conectadas si una puede ser obtenida a partir de la otra efectuando solo pivotajes eficientes.

Teorema 2.7 (**Steuer, 1985**). *Todas las bases eficientes están conectadas.*

Demostración. Sean $\mathfrak{B}$ y $\hat{\mathfrak{B}}$ dos bases eficientes. Sean $\lambda, \hat{\lambda} \in \mathbb{R}^p_>$ vectores de pesos positivos para los cuales $\mathfrak{B}$ y $\hat{\mathfrak{B}}$ son bases óptimas para el problema (2.3) con λ y con $\hat{\lambda}$, respectivamente. Consideremos el problema lineal paramétrico con función objetivo

$$c(\phi) = \phi\hat{\lambda}^T C + (1-\phi)\lambda^T C$$

con $\phi \in [0,1]$.
Sea $\hat{\mathfrak{B}}$ la primera base (para ϕ = 1). Después de varios pivotajes óptimos, obtenemos una base $\tilde{\mathfrak{B}}$ que es óptima para (2.3) con λ. Como $\lambda^* = \phi\hat{\lambda} + (1-\phi)\lambda \in \mathbb{R}^p_>$ para todo $\phi \in [0,1]$, todas las bases intermedias son óptimas para (2.3) con algún $\lambda^* \in \mathbb{R}^p_>$, es decir, son bases eficientes. Entonces, todos los pivotes paramétricos y óptimos son pivotes eficientes. Si $\tilde{\mathfrak{B}} = \mathfrak{B}$, ya está. Si no, $\mathfrak{B}$ puede ser obtenida a partir de $\tilde{\mathfrak{B}}$ mediante pivotes eficientes (es decir, pivotes eficientes para (2.3) con λ), pues tanto $\mathfrak{B}$ como $\tilde{\mathfrak{B}}$ son bases óptimas para este problema. □

2.3.2. Algoritmo simplex general para problemas multiobjetivo

Los resultados anteriores nos permiten pasar de una base eficiente a otra, por lo que para formular un algoritmo simplex multiobjetivo solo falta encontrar un modo de encontrar una base eficiente de la cual partir. Dado el problema

$$\min\{Cx : Ax = b,\ x \geqq 0\},$$

únicamente puede darse una de las tres situaciones siguientes:

- El problema es no factible, es decir, $\mathbf{X} = \emptyset$.
- El problema es factible pero no tiene soluciones eficientes, es decir, $\mathbf{X}_E = \emptyset$.
- El problema es factible y tiene soluciones eficientes, es decir, $\mathbf{X}_E \neq \emptyset$.

El algoritmo simplex multiobjetivo tratará con cada una de estas situaciones en tres fases, que presentamos a continuación:

Fase 1: Se determina una solución factible básica, o el algoritmo se detiene con la conclusión de que $\mathbf{X} = \emptyset$. Esta fase no involucra a la matriz C de la función objetivo, y de nuevo puede usarse el problema auxiliar (2.13).

Fase 2: Se determina una base eficiente inicial, o el algoritmo se detiene con la conclusión de que $\mathbf{X}_E = \emptyset$. Para ello:

La solución de un problema de suma ponderada con $\lambda > 0$ proporcionará una base eficiente, suponiendo que tal problema sea acotado. Si esta información no se conoce a priori, es necesario buscar un procedimiento que concluya que $\mathbf{X}_E = \emptyset$ o que devuelva un λ apropiado para el cual la suma ponderada tenga solución óptima. Suponiendo que $\mathbf{X} \neq \emptyset$, la fase 1 devuelve una solución factible básica $x^0 \in \mathbf{X}$, que puede ser eficiente o no. Procedemos en dos pasos: Primero, se resuelve el problema auxiliar (2.5) para determinar si $\mathbf{X}_E = \emptyset$. Por la proposición 2.1 y la dualidad, $\mathbf{X}_E \neq \emptyset$ si y solo si (2.5) devuelve un vector de pesos $\hat{w}$ apropiado, análogamente al argumento usado en la demostración del teorema 2.4. Por la proposición 2.1, el problema $\min\{Cx : Ax = b, x \geqq 0\}$ tiene una solución eficiente si y solo si el problema

$$\max \{e^T z : \ Ax = b, \ Cx + Iz = Cx^0, \ x, z \geqq 0\} \tag{2.19}$$

tiene solución óptima. Además, $\hat{x}$ en una solución óptima de (2.4) es eficiente. Sin embargo, no sabemos si $\hat{x}$ es una solución factible básica del problema multiobjetivo, luego en general no podemos elegir $\hat{x}$ para comenzar la fase 3 del algoritmo.
En vez de eso, aplicamos dualidad: (2.19) tiene solución óptima si y solo si su dual

$$\min \{u^T b + w^T C x^0 : \ u^T A + w^T C \geqq 0, \ w \geqq e\} \tag{2.20}$$

tiene una solución óptima $(\hat{u}, \hat{w})$ con $\hat{u}^T b + \hat{w} C x^0 = e^T \hat{z}$. Entonces, $\hat{u}$ también es una solución óptima del problema

$$\min \{u^T b : \ u^T A \geqq -\hat{w}^T C\}, \tag{2.21}$$

que es (2.20) con $w = \hat{w}$ fijo. Como en la demostración del teorema 2.4, el dual de (2.21) tiene solución óptima, y por tanto solución factible básica óptima, que es eficiente. El dual de (2.21) es equivalente al problema de suma ponderada

$$\min \{\hat{w}^T C x : \ Ax = b, \ x \geqq 0\}. \tag{2.22}$$

Se sigue que los problemas (2.20) y (2.3) con $\lambda = \hat{w}$ son las herramientas necesarias en la fase 2. Si (2.20) es no factible, $\mathbf{X}_E = \emptyset$. En otro caso, una solución óptima de (2.20) proporciona un vector de pesos apropiado $\lambda = \hat{w}$ para el cual (2.3) tiene una solución factible básica óptima, que es una solución factible básica eficiente del problema multiobjetivo.

Fase 3: Se pivota entre distintas bases eficientes para determinarlas todas y encontrar las direcciones de no acotación de $\mathbf{X}_E$.

Presentamos así una versión en pseudocódigo del algoritmo simplex general, válido para problemas lineales multiobjetivo:

Input: Datos A, b, C de un problema lineal multiobjetivo.
Output: Lista de bases (y soluciones) eficientes $\mathfrak{L}_2$.
Initialization: Tomar $\mathfrak{L}_1 := \emptyset$, $\mathfrak{L}_2 := \emptyset$.
Fase 1: Resolver el problema lineal $\min\{e^T z : Ax + Iz = b,\ x,\ z \geqq 0\}$.
if *el valor óptimo es distinto de 0* **then**
  PARAR, $\mathbf{X} = \emptyset$.
else
  sea x^0 una solución factible básica del problema lineal multiobjetivo.
end
Fase 2: Resolver el problema lineal $\min\{u^T b + w^T C x^0 : u^T A + w^T C \geqq 0,\ w \geqq e\}$.
if *este problema es no factible* **then**
  PARAR, $\mathbf{X}_E = \emptyset$.
else
  sea $(\hat{u}, \hat{w})$ una solución óptima.
end
Encontrar una base óptima $\mathfrak{B}$ del problema lineal $\min\{\hat{w}^T C x : Ax = b,\ x \geqq 0\}$.
$\mathfrak{L}_1 := \{\mathfrak{B}\}$, $\mathfrak{L}_2 := \emptyset$.
Fase 3:
while $\mathfrak{L}_1 \neq \emptyset$ **do**
  Elegir $\mathfrak{B}$ en $\mathfrak{L}_1$, tomar $\mathfrak{L}_1 := \mathfrak{L}_1 \setminus \{\mathfrak{B}\}$, $\mathfrak{L}_2 := \mathfrak{L}_2 \cup \{\mathfrak{B}\}$.
  Calcular $\tilde{A}, \tilde{b}$ y R según $\mathfrak{B}$.
  $\mathscr{E}\mathbf{N} := \mathbf{N}$.
  for $j \in \mathbf{N}$ **do**
    Resolver el problema lineal $\max\{e^T v : Ry - r^j \delta + Iv = 0;\ y,\ \delta,\ v \geqq 0\}$.
    if *este problema es no acotado* **then**
      $\mathscr{E}\mathbf{N} := \mathscr{E}\mathbf{N} \setminus \{j\}$.
    end
  end
  for $j \in \mathscr{E}\mathbf{N}$ **do**
    for $i \in \mathfrak{B}$ **do**
      if $\mathfrak{B}' = (\mathfrak{B} \setminus \{i\} \cup \{j\})$ *es factible y* $\mathfrak{B}' \notin \mathfrak{L}_1 \cup \mathfrak{L}_2$ **then**
        $\mathfrak{L}_1 := \mathfrak{L}_1 \cup \mathfrak{B}'$
      end
    end
  end
end

Algoritmo 2: Simplex general para problemas lineales multiobjetivo

2.3.3. Un ejemplo práctico

Ponemos en práctica este algoritmo resolviendo el mismo problema que en la sección anterior.

Fase 1: Del mismo modo que antes, la fase 1 no es necesaria, pues ya partimos de la solución factible básica $\mathbf{x}^0 = (0,0,2,7,10)$ (base $\{x_3, x_4, x_5\}$).

Fase 2: Pasamos entonces a resolver el primer problema auxiliar de la fase 2. Puesto que no está en forma estándar (se necesita que las variables sean no negativas), lo modelamos para que así sea y poder aplicar el algoritmo simplex habitual:

- Aplicamos el cambio de variable $w \to \bar{w} = w - 1$, con lo que pasaremos de tener la restricción $w \geqq e$ a tener $\bar{w} \geqq 0$. Notar además que esto añadirá una constante a la función objetivo, que no afectará al problema.
- Expresamos el vector u, que es no restringido en $\mathbb{R}^m$, como $u = u^1 - u^2$, donde $u^1, u^2 \geqq 0$.

De este modo, pasaremos a resolver el problema de la derecha:

$$\begin{array}{ll} \min & u^T b + w^T C x^0 \\ \text{sujeto a} & u^T A + w^T C \geqq 0 \\ & w \geqq e \\ & u \in \mathbb{R}^m \end{array} \quad \rightharpoonup \quad \begin{array}{ll} \min & bu_1 - bu_2 + Cx^0 \bar{w} \\ \text{sujeto a} & A^T u^1 - A^T u^2 + C^T \bar{w} \geqq k, \\ & u^1, u^2, \bar{w} \geqq 0 \end{array}$$

donde la i-ésima fila de k es el opuesto de la suma de la i-ésima fila de la matriz C.

Así, tras estas modificaciones y la adición de las variables de holgura, los datos del primer problema auxiliar de la fase 2 quedarán como sigue (en formato de tabla de simplex):

u_1^1	u_2^1	u_3^1	u_1^2	u_2^2	u_3^2	w_1	w_2	x_9	x_{10}	x_{11}	x_{12}	x_{13}	
-1	1	1	1	-1	-1	-3	-1	-1	0	0	0	0	4
1	1	2	-1	-1	-2	-1	-4	0	-1	0	0	0	5
1	0	0	-1	0	0	0	0	0	0	-1	0	0	0
0	1	0	0	-1	0	0	0	0	0	0	-1	0	0
0	0	1	0	0	-1	0	0	0	0	0	0	-1	0
2	7	1	-2	-7	-1	0	0	0	0	0	0	0	

Como de esta matriz de coeficientes no podemos extraer la matriz identidad de dimensión 5, es necesario aplicarle a este problema su propia fase 1 para obtener la primera solución factible básica (Se añaden 5 variables artificiales con costo 1 y se hacen 0 los costos de las demás variables, se resuelve por el método simplex, se eliminan las columnas correspondientes a las variables artificiales y se reinsertan los costos originales). Tras efectuar esta fase, se tiene la siguiente tabla con la información de la primera solución factible básica del primer problema auxiliar:

	u_1^1	u_2^1	u_3^1	u_1^2	u_2^2	u_3^2	w_1	w_2	x_9	x_{10}	x_{11}	x_{12}	x_{13}	
x_8	0	0	0	0	0	0	-5/2	1	-1	1/2	-3/2	1/2	0	3/2
x_{13}	0	0	0	0	0	0	-11/2	0	-2	1/2	-5/2	3/2	1	11/2
x_1	1	0	0	-1	0	0	0	0	0	0	-1	0	0	0
x_2	0	1	0	0	-1	0	0	0	0	0	0	-1	0	0
x_3	0	0	1	0	0	-1	-11/2	0	-2	1/2	-5/2	3/2	0	11/2
	0	0	0	0	0	0	55	0	20	-5	27	-8	0	

Partiendo de ella, aplicamos el algoritmo simplex habitual para hallar una solución óptima:

	u_1^1	u_2^1	u_3^1	u_1^2	u_2^2	u_3^2	w_1	w_2	x_9	x_{10}	x_{11}	x_{12}	x_{13}	
x_{12}	0	0	0	0	0	0	-5	2	-2	1	-3	1	0	3
x_{13}	0	0	0	0	0	0	2	-3	1	-1	2	0	1	1
x_1	1	0	0	-1	0	0	0	0	0	0	-1	0	0	0
x_2	0	1	0	0	-1	0	-5	2	-2	1	-3	0	0	3
x_3	0	0	1	0	0	-1	2	-3	1	-1	2	0	0	1
	0	0	0	0	0	0	15	16	4	3	3	0	0	

Descartando las variables de holgura, la solución óptima es entonces $(\hat{u_1}, \hat{u_2}, \hat{\bar{w}}) = (0,3,1,0,0,0,0,0)$, y deshaciendo el cambio de variable (sumando 1 a las componentes de $\hat{\bar{w}}$): $(\hat{u_1}, \hat{u_2}, \hat{w}) = (0,3,1,0,0,0,1,1)$.

Conociendo esta información, podemos resolver ahora el segundo problema auxiliar de la fase 2, $\min\{\hat{w}^T Cx : Ax = b,\ x \geqq 0\}$.

Calculamos $\hat{w}^T C = \begin{pmatrix}1 & 1\end{pmatrix}\begin{pmatrix}-3 & -1 & 0 & 0 & 0\\ -1 & -4 & 0 & 0 & 0\end{pmatrix} = \begin{pmatrix}-4 & -5 & 0 & 0 & 0\end{pmatrix}$ y almacenamos los datos de este problema en una tabla en formato de simplex (izquierda) y tras cuatro iteraciones obtenemos la solución óptima que aparece a la derecha:

	x_1	x_2	x_3	x_4	x_5	
x_3	-1	1	1	0	0	2
x_4	1	1	0	1	0	7
x_5	1	2	0	0	1	10
	-4	-5	0	0	0	

	x_1	x_2	x_3	x_4	x_5	
x_2	0	1	0	-1	1	3
x_3	0	0	1	3	-2	3
x_1	1	0	0	2	-1	4
	0	0	0	3	1	

Se tiene así el primer par de base y solución eficiente del problema original: $\mathfrak{B} = \{x_2, x_3, x_1\}$, $\mathbf{x}^T = (4,3,3,0,0)$, y podemos dar comienzo a la fase 3 del algoritmo partiendo de $\mathfrak{L}_1 = \{\{x_2, x_3, x_1\}\}$.

<u>Fase 3:</u> Iteración 1: Consideramos la base $\{x_2, x_3, x_1\}$ para los cálculos siguientes y la almacenamos en $\mathfrak{L}_2$. Así, $\mathfrak{L}_1 = \emptyset,\ \mathfrak{L}_2 = \{\{x_2, x_3, x_1\}\}$

$\mathscr{E}\mathbf{N} = \mathbf{N} = \{4,5\}$ (Índices de las variables no básicas)

Seleccionamos $j = 4$, y resolvemos el problema $\max\{e^T v : Ry - r^j\delta + Iv = 0;\ y,\ \delta,\ v \geqq 0\}$. Recordamos que la matriz R estará formada por las columnas no básicas de la matriz

$$\bar{C} = C - C_{\mathfrak{B}} A_{\mathfrak{B}}^{-1} A = \begin{pmatrix}-3 & -1 & 0 & 0 & 0\\ -1 & -4 & 0 & 0 & 0\end{pmatrix} - \begin{pmatrix}-1 & 0 & -3\\ -4 & 0 & -1\end{pmatrix}\begin{pmatrix}1 & 1 & -1\\ 1 & 0 & 1\\ 2 & 0 & 1\end{pmatrix}^{-1}\begin{pmatrix}-1 & 1 & 1 & 0 & 0\\ 1 & 1 & 0 & 1 & 0\\ 1 & 2 & 0 & 0 & 1\end{pmatrix} =$$

$$= \begin{pmatrix}0 & 0 & 0 & 5 & -2\\ 0 & 0 & 0 & -2 & 3\end{pmatrix},$$

con lo que los datos de este problema quedarán como se muestra en la tabla izquierda:

	x_1	x_2	x_3	x_4	x_5	
x_4	5	-2	-5	1	0	0
x_5	-2	3	2	0	1	0
	0	0	0	1	1	

	x_1	x_2	x_3	x_4	x_5	
x_4	0	11/2	0	1	5/2	0
x_3	-1	3/2	1	0	1/2	0
	0	11/2	0	0	3/2	

Como podemos extraer la matriz identidad de las columnas 4 y 5, la base inicial será $\{x_4, x_5\}$. Se recalculan los costos marginales según esta base y se aplica el algoritmo simplex, obteniendo la solución óptima de la parte derecha, con lo que el problema no es no acotado, y por tanto mantenemos el índice $j = 4$ en $\mathscr{E}\mathbf{N}$.

Tomamos ahora $j = 5$ y resolvemos el mismo problema asociado (izquierda) y de nuevo, tras recalcular los costos marginales y aplicar el simplex habitual, se llega a la solución óptima mostrada en la parte derecha:

	x_1	x_2	x_3	x_4	x_5	
x_4	5	-2	2	1	0	0
x_5	-2	3	-3	0	1	0
	0	0	0	-1	-1	

	x_1	x_2	x_3	x_4	x_5	
x_5	11/2	0	0	3/2	1	0
x_2	-5/2	1	-1	-1/2	0	0
	11/2	0	0	1/2	0	

y por tanto tampoco eliminamos el índice $j = 5$ de $\mathscr{E}\mathbf{N}$.

Ahora, para ver qué pivotajes pueden llevarse a cabo para obtener bases eficientes adyacentes, nos basaremos en el siguiente criterio de factibilidad:

Dada una base $\mathfrak{B}$ en la que acaba de realizarse un pivotaje, sea $A_{\mathfrak{B}}$ la matriz formada por las columnas básicas de la matriz de coeficientes A. Entonces, dicho pivotaje es factible si y solo si $A_{\mathfrak{B}}$ es regular y $\tilde{b} = A_{\mathfrak{B}}^{-1} b \geqq 0$.

Analizamos entonces todos los pivotajes posibles entre las variables x_4 y x_5 con todas las variables de la base actual, $\{x_2, x_3, x_1\}$:

- Si pivotamos la variable x_4 (entra en la base) por x_2 (sale de la base), se tiene que

$$A_{\mathfrak{B}} = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

  es regular, pero $\tilde{b}^T = (-3, 10, 12)$ no es no negativo, con lo que este pivotaje no es factible. Por tanto la base $\{x_4, x_3, x_1\}$ no es eficiente para el problema inicial y no se almacena en $\mathfrak{L}_1$.

- Pivotando x_4 por x_3, se tiene que

$$A_{\mathfrak{B}} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

  es regular y además $\tilde{b}^T = (4, 1, 2)$ es no negativo, con lo que este pivotaje será factible. Así, la base $\{x_2, x_4, x_1\}$ y su solución asociada $\mathbf{x}^T = (2, 4, 0, 1, 0)$ serán eficientes para el problema. Como esta base no está en $\mathfrak{L}_1$ ni en $\mathfrak{L}_2$, se almacena en $\mathfrak{L}_1$, y así $\mathfrak{L}_1 = \{\{x_2, x_4, x_1\}\}$.

- Pivotando x_4 por x_1,

$$A_{\mathfrak{B}} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 0 & 0 \end{pmatrix}$$

  es regular, pero $\tilde{b}^T = (5, -3, 2)$ no es no negativo, con lo que este pivotaje no es factible.

- Pivotando x_5 por x_2,

$$A_{\mathfrak{B}} = \begin{pmatrix} 0 & 1 & -1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

  es regular y $\tilde{b}^T = (3, 9, 7) \geqq 0$, con lo que este pivotaje será factible. La base $\{x_5, x_3, x_1\}$ y su solución asociada $\mathbf{x}^T = (7, 0, 0, 9, 3)$ serán eficientes para el problema, y como esta base no está en $\mathfrak{L}_1$ ni en $\mathfrak{L}_2$, se almacena en $\mathfrak{L}_1$ y ahora $\mathfrak{L}_1 = \{\{x_2, x_4, x_1\}, \{x_5, x_3, x_1\}\}$.

- Pivotando x_5 por x_3,

$$A_{\mathfrak{B}} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & 1 \\ 2 & 1 & 1 \end{pmatrix}$$

  es regular, pero $\tilde{b}^T = (9/2, -3/2, 5/2)$ no es no negativo, así que este pivotaje no es factible.

- Por último, pivotando x_5 por x_1,

$$A_{\mathfrak{B}} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

  es regular, pero $\tilde{b}^T = (7, -5, -4)$ no es no negativo, con lo que este pivotaje no es factible.

Por su similitud con la anterior, las iteraciones 2 y 3 del algoritmo se han añadido al anexo A. Como en la tercera iteración no hemos añadido ninguna base nueva a $\mathfrak{L}_1 = \emptyset$, el algoritmo se detiene. Se han obtenido entonces 3 bases y soluciones eficientes para el problema:

$\{x_2, x_3, x_1\}$, $\mathbf{x}^T = (4,3,3,0,0)$, $\{x_2, x_4, x_1\}$, $\mathbf{x}^T = (2,4,0,1,0)$ y $\{x_5, x_3, x_1\}$, $\mathbf{x}^T = (7,0,0,9,3)$

Nótese que las bases 2 y 3 están conectadas únicamente con la base 1, es decir, no es posible obtener la 3 partiendo de la 2 (o viceversa) sin pasar por la base 1. Y como antes, si se quiere situar las soluciones en el conjunto factible de partida, omítanse las componentes correspondientes a las variables de holgura (x_3, x_4 y x_5).

2.4. Implementación en lenguaje Java

Se han implementado los dos algoritmos presentados en el capítulo 2; dicha implementación permite constatar el conocimiento y entendimiento de dichos algoritmos, además de proporcionar una herramienta para la resolución de problemas lineales multiobjetivo académicos. La implementación no se ha realizado por tanto teniendo como objetivo la eficiencia, velocidad y posibilidad de resolución de problemas de grandes dimensiones, sino el desarrollo de una herramienta sencilla y que sirva de apoyo para dichos problemas académicos, que bien pudiera utilizarse en una hipotética asignatura dedicada a la programación lineal multiobjetivo.

Para la mencionada implementación, se ha escogido el lenguaje de programación Java (a través del entorno de desarrollo *Apache NetBeans IDE 13*), introducido en la asignatura *Informática II* del grado, y la librería EJML (Efficient Java Matrix Library) [10], que permite la manipulación sencilla de matrices al tener implementadas operaciones básicas con las mismas como sumas, restas, productos o cálculo de determinantes, traspuestas e inversas. Otros métodos a destacar de esta librería serían la extracción de submatrices o la resolución de sistemas lineales.

Para el primer algoritmo se han implementado dos clases de Java: una de ellas (simplexBiObjective) representará cada problema particular (almacenará sus datos, y contendrá todas las operaciones que se llevarán a cabo con estos), y la otra (SimplexBiObjetivoMain) contendrá al método *main*, y servirá para instanciar (crear) y resolver cada problema: El programa leerá a partir de un fichero de texto los datos del problema, creará el objeto de la primera clase y sucesivamente hará "llamadas" a los métodos (funciones) doPhase1(), doPhase2() y doPhase3() para que se lleven a cabo las fases 1, 2 y 3 del algoritmo para el problema dado. Cada uno de estos métodos se servirá a su vez de otros métodos auxiliares:
En la fase 1 intervendrán los métodos whichEntersPhase1(), whichLeavesPhase1(int in), doPivotPhase1(int in, int out), isPhase1Optimal() y finishingPhase1(). En la fase 2, criterioEntrada(), criterioSalida(int ENTRA) y recalcula(), correspondientes a los pasos del algoritmo simplex habitual. Por último, en la fase 3, criterioEntrada2(), recalcula2(), calculaVFO(SimpleMatrix solucion) y calculaVFO(double[] solucion).

Para el segundo algoritmo, las clases implementadas son cuatro: SimplexMultiObjetivoMain, simplexMultiObjective, simplexUniObjective y Base. Las dos primeras tendrán el mismo cometido que en el anterior: crear y resolver el problema con los datos obtenidos de un fichero de texto. La clase simplexUniObjective servirá para instanciar y resolver cada uno de los problemas auxiliares uniobjetivo que se plantean a lo largo del algoritmo, y contará con métodos similares a los de las fases 1 y 2 del algoritmo anterior. Por último, cada objeto Base representará una base eficiente del problema en la fase 3.
La fase 1 del algoritmo se efectuará creando y resolviendo el problema adecuado con la clase simplexUniObjective. En la fase 2, la preparación (y resolución, mediante un objeto de tipo simplexUniObjective) del primer problema a resolver se llevará a cabo en el método doPhase2Aux1(). La del segundo, en doPhase2Aux2() (partiendo de la solución del anterior, y de nuevo a través de la clase simplexUniObjective). Finalmente, en doPhase3() tendrá lugar la fase 3 al completo (salvo las llamadas a getL2(), calculaVFO(SimpleMatrix solucion), calculaVFO(double[] solucion) y isInL(Base t, TreeSet<Base> L)), y también se servirá de objetos simplexUniObjective.

En ambos programas, la primera componente del argumento *args* del método *main* es obligatoria y contiene el nombre del fichero de texto del cual se leen los datos del problema. Si hay un segundo argumento, se tomará como el nombre del fichero en el que se escribirán los datos de salida del programa. Es importante destacar que en la salida, los índices de las variables comienzan a contarse a partir de 0. En los anexos B y C, respectivamente, se encuentra todo el código perteneciente a las implementaciones anteriores. A continuación y por último, mostramos el planteamiento de 5 problemas extraídos de la literatura. Estos se han resuelto utilizando los programas desarrollados, y las soluciones halladas pueden encontrarse en el anexo D.

Problema 1: (Ref. [2])

$$\begin{aligned} \text{mín} \quad & x_1, \ \text{mín} \ x_2 \\ \text{s. a:} \quad & 2x_1 + x_2 \geq 4 \\ & x_1 + x_2 \geq 3 \\ & x_1 + 2x_2 \geq 4 \\ & -x_1 \geq -5 \\ & -x_2 \geq -5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Problema 2: (Ref. [4])

$$\begin{aligned} \text{máx} \quad & 3x_1 + x_2 + 2x_3 + x_4 \\ \text{máx} \quad & -x_1 + 5x_2 + x_3 + 2x_4 \\ \text{s. a:} \quad & 2x_1 + x_2 + 4x_3 + 3x_4 \geq 60 \\ & 3x_1 + 4x_2 + x_3 + 2x_4 \geq 60 \\ & x_1 + 2x_2 + 3x_3 + 4x_4 \geq 50 \\ & 4x_1 + 3x_2 + 2x_3 + 1x_4 \geq 50 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Problema 3: (Ref. [4]) Es el problema 2, con una función objetivo adicional: máx $x_1 - x_2 + 2x_3 + 4x_4$.

Problema 4: (Ref. [7])

$$\begin{aligned} \text{máx} \quad & x_1 + 2x_2 - x_3 + 3x_4 + 2x_5 + x_7 \\ \text{máx} \quad & x_2 + x_3 + 2x_4 + 3x_5 + x_6 \\ \text{máx} \quad & x_1 + x_3 - x_4 + -x_6 - x_7 \\ \text{s. a:} \quad & x_1 + 2x_2 + x_3 + x_4 + 2x_5 + x_6 + 2x_7 \leq 16 \\ & -2x_1 - x_2 + x_4 + 2x_5 + x_7 \leq 16 \\ & -x_1 + x_3 + 2x_5 - 2x_7 \leq 16 \\ & x_2 + 2x_3 - x_4 + x_5 - 2x_6 - x_7 \leq 16 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0 \end{aligned}$$

Problema 5: (Ref. [8])

$$\begin{aligned} \text{máx} \quad & 3x_1 - 7x_2 + 4x_3 + x_4 - x_6 - x_7 + 8x_8 \\ \text{máx} \quad & 2x_1 + 5x_2 + x_3 - x_4 + 6x_5 + 8x_6 + 3x_7 - 2x_8 \\ \text{máx} \quad & 5x_1 - 2x_2 + 5x_3 + 6x_5 + 7x_6 + 2x_7 + 6x_8 \\ \text{máx} \quad & 4x_2 - x_3 - x_4 - 3x_5 + x_8 \\ \text{máx} \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \\ \text{s. a:} \quad & x_1 + 3x_2 - 4x_3 + x_4 - x_5 + x_6 + x_7 + x_8 \leq 40 \\ & 5x_1 + 2x_2 + 4x_3 - x_4 + 3x_5 + 7x_6 + 2x_7 + 7x_8 \leq 84 \\ & 4x_2 - x_3 - x_4 - 3x_5 + x_8 \leq 18 \\ & -3x_1 - 4x_2 + 8x_3 + 2x_4 + 3x_5 - 4x_6 + 5x_7 - x_8 \leq 100 \\ & 12x_1 + 8x_2 - x_3 + 4x_4 + x_6 + x_7 \leq 40 \\ & -x_1 - x_2 - x_3 - x_4 - x_5 - x_6 - x_7 - x_8 \leq -12 \\ & 8x_1 - 12x_2 - 3x_3 + 4x_4 - x_5 \leq 30 \\ & -5x_1 - 6x_2 + 12x_3 + x_4 - x_7 + x_8 \leq 100 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \geq 0 \end{aligned}$$

Bibliografía

[1] HAIMES, Y. Y., LASDON, L. S., AND WISMER, D. A., *On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization, IEEE Transactions on Systems, Man, and Cybernetics SMC-1, 3, July 1971, 296–297.*

[2] JUSTO PUERTO, *A moment method to solve multiobjective linear programs, 2013, 6.*

[3] M. JESÚS RÍOS INSUA, SIXTO RÍOS INSUA, *Procesos de decisión multicriterio (EUDEMA Universidad), 1989, 108-109.*

[4] MARIA JOÃO ALVES, CARLOS HENGGELER ANTUNES, JOÃO CLÍMACO, *Interactive MOLP Explorer—A Graphical-Based Computational Tool for Teaching and Decision Support in Multi-Objective Linear Programming Models, 2014, 319-320.*

[5] MATTHIAS EHRGOTT, *Multicriteria Optimization Second edition, Springer, 2005.*

[6] OLVI L. MANGASARIAN, *Nonlinear Programming, Society for Industrial and Applied Mathematics, Philadelphia, 1969.*

[7] P. L. YU, M. ZELENY, *The techniques of linear multiobjective programming, RAIRO - Operations Research - Recherche Opérationnelle, Tome 8 , 1974, no. V3, 64-69.*

[8] P. L. YU, M. ZELENY, *The Set of All Nondominated Solutions in Linear Cases and a Multicriteria Simplex Method, 1975, 454-455.*

[9] PASCHAL B NYIAM AND ABDELLAH SALHI, *A comparative study of two key algorithms in multiple objective linear programming, Journal of Algorithms & Computational Technology, 2019, Volume 13, 1-18.*

[10] `http://ejml.org/wiki/index.php?title=Main_Page`

[11] `https://www.lindo.com/index.php/products/lingo-and-optimization-modeling`

Anexo A

Iteración 2: Extraemos la base $\{x_2, x_4, x_1\}$ de $\mathfrak{L}_1$ para los cálculos siguientes y la almacenamos en $\mathfrak{L}_2$, obteniéndose $\mathfrak{L}_1 = \{\{x_5, x_3, x_1\}\},\ \mathfrak{L}_2 = \{\{x_2, x_3, x_1\}, \{x_2, x_4, x_1\}\}$.

$\mathscr{E}\mathbf{N} = \mathbf{N} = \{3, 5\}$ (Índices de las variables no básicas)

Seleccionamos $j = 3$, y resolvemos el problema $\max\{e^T v : Ry - r^j\delta + Iv = 0;\ y,\ \delta,\ v \geqq 0\}$, donde, como antes, R estará formada por las columnas no básicas de la matriz

$$\bar{C} = C - C_{\mathfrak{B}} A_{\mathfrak{B}}^{-1} A = \begin{pmatrix} 0 & 0 & -5/3 & 0 & 4/3 \\ 0 & 0 & 2/3 & 0 & 5/3 \end{pmatrix},$$

es decir, los datos del problema quedarán como sigue:

	x_1	x_2	x_3	x_4	x_5	
x_4	-5/3	4/3	5/3	1	0	0
x_5	2/3	5/3	-2/3	0	1	0
	0	0	0	1	1	

La base inicial de este problema será $\{x_4, x_5\}$. Se recalculan los costos marginales según esta base y se aplica el algoritmo simplex, obteniéndose la siguiente solución óptima

	x_1	x_2	x_3	x_4	x_5	
x_4	0	11/2	0	1	5/2	0
x_1	1	5/2	-1	0	3/2	0
	0	11/2	0	0	3/2	

Luego como el problema tiene solución óptima, mantenemos el índice $j = 3$ en $\mathscr{E}\mathbf{N}$.

Tratamos de resolvemos el mismo problema para $j = 5$:

	x_1	x_2	x_3	x_4	x_5	
x_4	-5/3	4/3	-4/3	1	0	0
x_5	2/3	5/3	-5/3	0	1	0
	0	0	0	1	1	

Tras algunas iteraciones, se llega a la conclusión de que este problema es no acotado, con lo que se elimina el índice $j = 5$ de $\mathscr{E}\mathbf{N}$, resultando en $\mathscr{E}\mathbf{N} = \{3\}$.

Veamos ahora qué pivotajes entre la variable x_3 y las variables de la base actual $\{x_2, x_4, x_1\}$ son factibles, según el criterio visto antes:

- Si pivotamos la variable x_3 por x_2 , se tiene que

$$A_{\mathfrak{B}} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

es regular, pero $\tilde{b}^T = (12,-3,10)$ no es no negativo, con lo que este pivotaje no es factible y la base $\{x_3,x_4,x_1\}$ no se almacena en $\mathfrak{L}_1$.

- Pivotando x_3 por x_4 se obtiene la base anterior a la actual, la cual ya sabemos que es eficiente y ya está almacenada en $\mathfrak{L}_2$.
- Si pivotamos la variable x_3 por x_1 , se tiene que

$$A_{\mathfrak{B}} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 2 & 0 & 0 \end{pmatrix}$$

es regular, pero $\tilde{b}^T = (5,2,-3)$ no es no negativo, con lo que este pivotaje no es factible.

Iteración 3: Consideramos ahora la base $\{x_5,x_3,x_1\}$ para los cálculos siguientes y la almacenamos en $\mathfrak{L}_2$, obteniéndose $\mathfrak{L}_1 = \emptyset,\ \mathfrak{L}_2 = \{\{x_2,x_3,x_1\},\{x_2,x_4,x_1\},\{x_5,x_3,x_1\}\}$.

$\mathscr{E}\mathbf{N} = \mathbf{N} = \{2,4\}$ (Índices de las variables no básicas)

Seleccionamos $j = 2$, y de nuevo resolvemos el problema $\max\{e^T v : Ry - r^j\delta + Iv = 0;\ y,\ \delta,\ v \geqq 0\}$, con R formada por las columnas no básicas de

$$\bar{C} = C - C_{\mathfrak{B}}A_{\mathfrak{B}}^{-1}A = \begin{pmatrix} 0 & 2 & 0 & 3 & 0 \\ 0 & -3 & 0 & 1 & 0 \end{pmatrix}.$$

es decir, los datos del problema quedarán como sigue:

	x_1	x_2	x_3	x_4	x_5	
x_4	2	3	-2	1	0	0
x_5	-3	1	3	0	1	0
	0	0	0	1	1	

Recalculamos los costos marginales según la base $\{x_4,x_5\}$ y aplicamos el algoritmo simplex, obteniéndose la solución óptima:

	x_1	x_2	x_3	x_4	x_5	
x_5	0	11/2	0	3/2	1	0
x_3	-1	-3/2	1	-1/2	0	0
	0	11/2	0	1/2	0	

El problema no es no acotado, y por tanto se mantiene el índice $j = 2$ en $\mathscr{E}\mathbf{N}$.

Aplicamos el algoritmo simplex al mismo problema asociado para $j = 4$, con datos:

	x_1	x_2	x_3	x_4	x_5	
x_4	2	3	-3	1	0	0
x_5	-3	1	-1	0	1	0
	0	0	0	1	1	

y tras unas iteraciones el algoritmo se detiene porque el problema es no acotado. Por tanto, eliminamos el índice $j = 4$ de $\mathscr{E}\mathbf{N}$, resultando $\mathscr{E}\mathbf{N} = \{2\}$.

Verificamos cuáles de los pivotajes posibles entre x_2 y las variables de $\{x_5,x_3,x_1\}$ son factibles:

- Si se pivota x_2 por x_5, se obtiene la primera base eficiente, ya almacenada en $\mathfrak{L}_2$.
- Pivotando la variable x_2 por x_3 , se tiene que

$$A_{\mathfrak{B}} = \begin{pmatrix} 0 & 1 & -1 \\ 0 & 1 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$

es regular, pero $\tilde{b}^T = (-3/2, 9/2, 5/2)$ no es no negativo, con lo que este pivotaje no es factible.

- Pivotando la variable x_2 por x_3 , se tiene que

$$A_{\mathfrak{B}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 2 \end{pmatrix}$$

es regular, pero $\tilde{b}^T = (-4, -5, 7)$ no es no negativo, con lo que este pivotaje no es factible.

Anexo B

Clase SimplexBiObjetivoMain.java:

```
package simplexbiobjetivo;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.ejml.simple.SimpleMatrix;
import org.apache.commons.lang3.time.StopWatch;

/*Esta clase lee los datos de un problema de programación lineal biobjetivo a
través de un fichero de texto y llama a la clase simplexBiObjective para que
trate de resolverlo*/
public class SimplexBiObjetivoMain {

   public static StopWatch contador;

   public static void main(String[] args) {

      contador = StopWatch.createStarted();

      //Lectura de los datos del problema a través de un fichero de texto (El nombre
          del mismo es el primer argumento de main)
      File fentrada = new File(args[0]);
      Scanner entrada = null;
      String linea = null;

      try {
         entrada = new Scanner(fentrada);
         //Se leen las dimensiones de A
         int m = entrada.nextInt();
         int n = entrada.nextInt();
         int o = 2; //2 objetivos, no es necesario indicarlo en el fichero
         //Se lee el tipo del problema (min o max)
         entrada.nextLine();
         String tipo = entrada.next();
         int dirOpt = 0;
         if (tipo.equals("max")) {
            dirOpt = simplexBiObjective.maximizacion;
         } else if (tipo.equals("min")) {
            dirOpt = simplexBiObjective.minimizacion;
         } else {
            System.out.println("Tipo de problema no válido");
            System.exit(0);
```

```
        }
        //Se lee la matriz A
        double a[][] = new double[m][n];
        for (int i = 0; i < m; i++) {
           entrada.nextLine();
           for (int j = 0; j < n; j++) {
              a[i][j] = entrada.nextDouble();
           }
        }
        //Se lee la matriz C
        double c[][] = new double[o][n];
        for (int i = 0; i < o; i++) {
           entrada.nextLine();
           for (int j = 0; j < n; j++) {
              c[i][j] = entrada.nextDouble();
           }
        }
        //Se lee el vector b
        double b[] = new double[m];
        entrada.nextLine();
        for (int i = 0; i < m; i++) {
           b[i] = entrada.nextDouble();
        }
        //Se leen los tipos de las restricciones
        int tipoRest[] = new int[m];
        int tiporestr;
        entrada.nextLine();
        for (int i = 0; i < m; i++) {
           tiporestr = entrada.nextInt();
           if (tiporestr == -1) {
              tipoRest[i] = simplexBiObjective.menor;
           } else if (tiporestr == 0) {
              tipoRest[i] = simplexBiObjective.igualdad;
           } else if (tiporestr == 1) {
              tipoRest[i] = simplexBiObjective.mayor;
           } else {
              System.out.println("Tipo de restricción no válido");
              System.exit(0);
           }
        }

        //Creación del problema
        simplexBiObjective sbo = new simplexBiObjective(a, b, c, tipoRest, dirOpt);

        //Si la función main recibe un segundo argumento, los datos de salida se
            escribirán en el fichero con ese nombre
        if (args.length == 2) {
           sbo.creaFichero(args[1]);
        }

        //Resolución del problema
        sbo.solve();
     } catch (FileNotFoundException ex) {
        Logger.getLogger(SimplexBiObjetivoMain.class.getName()).log(Level.SEVERE,
            null, ex);
     }
   }
}
```

Clase simplexBiObjective.java:

```
package simplexbiobjetivo;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.logging.Level;
import java.util.logging.Logger;
import static javax.lang.model.type.TypeKind.NULL;
import org.ejml.simple.SimpleMatrix;

/*Esta clase recibe la información de un problema de programación lineal
biobjetivo y trata de resolverlo*/
public class simplexBiObjective {

   private int recursosNegativosError = 1;
   public static final int igualdad = 0;
   public static final int menor = -1;
   public static final int mayor = 1;
   public static final int minimizacion = 1;
   public static final int maximizacion = 2;
   public static final int problemaNoFactible = 1;
   public static final int problemaFactible = 0;
   public static final double epsilon = 1E-10;
   private int m, n;
   private final int nObj = 2;
   private SimpleMatrix A;
   private SimpleMatrix b;
   private SimpleMatrix C;
   private SimpleMatrix C_BPhase1;
   private SimpleMatrix cFase1;
   private SimpleMatrix cMFase1;
   private SimpleMatrix C_M; //Costos marginales
   private SimpleMatrix C_B; //costos básicos
   private int base[];
   private int numHolguras;
   private int numArtificiales;
   private boolean phase1needed = false;
   private int nPhase1;

   private SimpleMatrix AOriginal;
   private SimpleMatrix bOriginal;
   private SimpleMatrix COriginal;
   private SimpleMatrix c1Original;

   //Otros datos que se modificarán en cada iteración
   private SimpleMatrix B2;
   private SimpleMatrix c1_2;
   private SimpleMatrix c2_2;
   private SimpleMatrix c_B2;

   double lambdahat = 1;

   private int dirOptimization = 1;

   private int escribeEnFichero = 0;
```

```
PrintWriter out = null;

public simplexBiObjective(double a[][], double bb[], double cc[][], int
    typeConstraint[], int dirOptimizacion) {

   for (double elem : bb) {
      if (elem < 0) {
         System.out.println("El vector de recursos debe ser >=0");
         System.exit(recursosNegativosError);
      }
   }
   n = a[0].length;
   m = bb.length;
   base = new int[m];

   for (int i : typeConstraint) {
      switch (i) {
         case igualdad:
         numArtificiales++;
         break;
         case menor:
         numHolguras++;
         break;
         case mayor:
         numHolguras++;
         numArtificiales++;
         break;
      }
   }
   A = new SimpleMatrix(bb.length, n + numArtificiales + numHolguras);
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         A.set(i, j, a[i][j]);
      }
   }
   int colOffset = 0;
   for (int i = 0; i < m; i++) {
      switch (typeConstraint[i]) {
         case menor:
         A.set(i, n + colOffset, 1);
         base[i] = n + colOffset;
         colOffset++;
         break;
         case mayor:
         A.set(i, n + colOffset, -1);
         colOffset++;
         break;
      }
   }
   for (int i = 0; i < m; i++) {
      if (typeConstraint[i] == igualdad || typeConstraint[i] == mayor) {
         A.set(i, n + colOffset, 1);
         base[i] = n + colOffset;
         colOffset++;
      }
   }

   b = new SimpleMatrix(bb.length, 1, true, bb);
```

```
      C = new SimpleMatrix(2, n + numHolguras);
      C_M = new SimpleMatrix(2, n + numHolguras);

      //Si el problema es de máximo, se multiplica C por -1 para hacerlo de mínimo
      if (dirOptimizacion == maximizacion) {
         dirOptimization = maximizacion;

         for (int i = 0; i < cc.length; i++) {
            for (int j = 0; j < cc[0].length; j++) {
               cc[i][j] = -cc[i][j];
            }
         }
      }

      for (int i = 0; i < 2; i++) //BIobjetive
      {
         for (int j = 0; j < n; j++) {
            C.set(i, j, cc[i][j]);
         }
      }

      if (numArtificiales > 0) {//Es necesario fase 1
         cFase1 = new SimpleMatrix(1, n + numArtificiales + numHolguras);
         for (int j = n + numHolguras; j < n + numArtificiales + numHolguras; j++) {
            cFase1.set(0, j, 1);
         }
         C_BPhase1 = new SimpleMatrix(1, m);
         for (int i = 0; i < m; i++) {
            if (base[i] >= n + numHolguras) {
               C_BPhase1.set(0, i, 1);
            }
         }
         //Se calcula los costos marginales
         cMFase1 = cFase1.minus(C_BPhase1.mult(A));
         nPhase1 = n + numArtificiales + numHolguras;
         phase1needed = true;
      }

      //Datos originales del problema
      AOriginal = A.extractMatrix(0, A.numRows(), 0, n + numHolguras);
      bOriginal = b.copy();
      SimpleMatrix cmatriz = new SimpleMatrix(cc);
      SimpleMatrix ceros = new SimpleMatrix(cmatriz.numRows(), numHolguras);
      COriginal = cmatriz.concatColumns(ceros); //Matriz de costos, con 0 en las
          variables de holgura
      c1Original = COriginal.extractVector(true, 0); //Vector de costos del primer
          objetivo

      B2 = new SimpleMatrix(A.numRows(), A.numRows());
      c_B2 = new SimpleMatrix(1, bb.length, true, bb);
   }

   public void solve() {
      if (isPhase1needed()) {
         if (doPhase1() == simplexBiObjective.problemaFactible) {
         } else {
            System.out.println("Problema no factible");
            System.exit(simplexBiObjective.problemaNoFactible);
```

```
      }
   }
   finishingPhase1();

   //FASE 2: Algoritmo simplex para resolver el problema con lambda = 1
   doPhase2();

   //FASE 3: Obtención de valores lambda y soluciones eficientes para tales
   doPhase3();
}

/**
* ********************************************************************
* FASE 1
* ********************************************************************
*/
public boolean isPhase1needed() {
   return phase1needed;
}

int doPhase1() {
   int in;
   int out;
   int iteration = 1;
   while (!isPhase1Optimal()) {
      in = whichEntersPhase1();
      out = whichLeavesPhase1(in);
      if (out == -1) {
         System.out.println("Problema no factible");
         System.exit(problemaNoFactible);
      }
      doPivotPhase1(in, out);
   }
   return problemaFactible;

}

//-1 => solucion optima
int whichEntersPhase1() {
   double minValue = Double.MAX_VALUE;
   int minPos = -1;

   int noBasica[] = new int[nPhase1];
   for (int i = 0; i < noBasica.length; i++) {
      noBasica[i] = i;
   }
   for (int i = 0; i < base.length; i++) {
      noBasica[base[i]] = -1;
   }
   for (int i = 0; i < nPhase1; i++) {
      if (noBasica[i] == -1) {
         continue;
      }
      if (cMFase1.get(0, i) < minValue) {
         minValue = cMFase1.get(0, i);
         minPos = i;
      }
   }
```

```
   return minPos;
}

//-1 problema no acotado
int whichLeavesPhase1(int in) {
   double minVal = Double.MAX_VALUE;
   int minPos = -1;
   SimpleMatrix colIn = A.extractVector(false, in);

   for (int i = 0; i < m; i++) {
      if (colIn.get(i, 0) > 0 && b.get(i, 0) / colIn.get(i, 0) < minVal) {
         minVal = b.get(i, 0) / colIn.get(i, 0);
         minPos = i;
      }
   }
   return minPos;
}

void doPivotPhase1(int in, int out) {
   //Se actualizan la base, A, b y los costos marginales
   SimpleMatrix elem = SimpleMatrix.identity(m);
   double Yst = 1.0 / A.get(out, in);
   for (int i = 0; i < m; i++) {
      if (i != out) {
         elem.set(i, out, -A.get(i, in) * Yst);
      } else {
         elem.set(i, out, Yst);
      }
   }
   //Se actualiza A
   A = elem.mult(A);
   //Se actualiza b
   b = elem.mult(b);
   //Se actualiza la base
   base[out] = in;
   //Se actualiza el vector de costos de variables básicas
   C_BPhase1.set(0, out, cFase1.get(in));
   //Se actualizan los costos marginales
   cMFase1 = cFase1.minus(C_BPhase1.mult(A));
}

boolean isPhase1Optimal() {
   int noBasica[] = new int[nPhase1];
   for (int i = 0; i < noBasica.length; i++) {
      noBasica[i] = i;
   }
   for (int i = 0; i < base.length; i++) {
      noBasica[base[i]] = -1;
   }
   for (int i = 0; i < nPhase1; i++) {
      if (noBasica[i] == -1) {
         continue;//nuevo
      }
      if (cMFase1.get(0, i) < -epsilon) {
         return false;
      }
   }
```

```
        return true;

    }

    /* Se encarga de cargar los datos para el problema a resolver, ahora que
    ya sabemos que es factible y que ya tenemos una base inicial.
    Solo es necesario eliminar las variables artificiales, la parte correspondiente
    en la matriz A, y calcular los primeros costos marginales C_M */
    void finishingPhase1() {
        //Se quitan las columnas de las variables artificiales
        A = A.extractMatrix(0, m, 0, n + numHolguras);

        //Se calcula la matriz de costos básicos
        C_B = new SimpleMatrix(2, m);
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < m; j++) {
                C_B.set(i, j, C.get(i, base[j]));
            }
        }

        //Se calcula la matriz de costos marginales
        C_M = C.minus(C_B.mult(A));
        c1_2 = C_M.extractVector(true, 0);
        c2_2 = C_M.extractVector(true, 1);
    }

    /**
    * ************************************************************************
    * FASE 2 (Lambda=1), resolver con c1
    * ************************************************************************
    */
    public void doPhase2() {

        while (1 > 0) {
            //CRITERIO DE ENTRADA
            int[] critEntrada = this.criterioEntrada();
            if (critEntrada[0] == 1) {
                //FIN DE FASE 2 (SOLUCIÓN ÓPTIMA CON LAMBDA = 1)
                break;
            }
            int ENTRA = critEntrada[1];

            //CRITERIO DE SALIDA
            int[] critSalida = this.criterioSalida(ENTRA);
            if (critSalida[0] == 1) {
                System.out.println("Solución no acotada");
                System.exit(0);
            }
            int SALE = critSalida[1];

            //SE RECALCULAN A, b Y c:
            SimpleMatrix[] nuevosdatos = this.recalcula();
            B2 = nuevosdatos[0];
            c_B2 = nuevosdatos[1];
            A = nuevosdatos[2];
            b = nuevosdatos[3];
            c1_2 = nuevosdatos[4];
        }
```

```
}

public int[] criterioEntrada() { //La primera componente que devuelve es el
    criterio de parada
   int ENTRA = 0; //Variable que entra en la base
   double costoMin = c1_2.get(0);

   for (int i = 1; i < A.numCols(); i++) {
      double costo = c1_2.get(i);

      if (costo < costoMin) {
         ENTRA = i;
         costoMin = costo;
      }
   }
   if (costoMin >= 0) {
      return new int[]{1, 0};
   }

   return new int[]{0, ENTRA};
}

public int[] criterioSalida(int ENTRA) { //La primera componente que devuelve es el
    criterio de parada
   int SALE = 0; //Variable que sale de la base
   int cualSale = -1;
   SimpleMatrix columnaEntra = A.extractVector(false, ENTRA);
   double cociente = 999999999;

   for (int k = 0; k < A.numRows(); k++) {

      if (columnaEntra.get(k) != 0) {
         double division = b.get(k) / columnaEntra.get(k);

         if (division > 0 && division < cociente) {
            cociente = division;
            SALE = base[k];
            cualSale = k;
         }
      }
   }
   if (cualSale == -1) {
      return new int[]{1, 0};
   }

   base[cualSale] = ENTRA;
   return new int[]{0, SALE};
}

public SimpleMatrix[] recalcula() {

   /* Se calcula la matriz A_B, los nuevos costos básicos, la nueva matriz
   A, el nuevo vector b y los nuevos costos marginales*/
   for (int i = 0; i < base.length; i++) {
      SimpleMatrix columnaB = AOriginal.extractVector(false, base[i]);
      B2.insertIntoThis(0, i, columnaB);
   }
```

```
      for (int j = 0; j < base.length; j++) {
         c_B2.set(j, c1Original.get(base[j]));
      }

      A = B2.solve(AOriginal);
      b = B2.solve(bOriginal);
      c1_2 = c1Original.minus(c_B2.mult(A));

      return new SimpleMatrix[]{B2, c_B2, A, b, c1_2};
   }

   /**
   * ********************************************************************
   * FASE 3
   * ********************************************************************
   */
   public void doPhase3() {
      //Se calculan las matrices de costos básicos y marginales, para actualizar la
          segunda fila
      for (int i = 0; i < 2; i++) {
         for (int j = 0; j < m; j++) {
            C_B.set(i, j, C.get(i, base[j]));
         }
      }

      C_M = COriginal.minus(C_B.mult(A));
      c2_2 = C_M.extractVector(true, 1);

      if (escribeEnFichero == 1) {
         out.println("Bases eficientes, soluciones respectivas y valor de la función
             objetivo: ");
      }
      System.out.println("Bases eficientes, soluciones respectivas y valor de la
          función objetivo: ");

      while (1 > 0) {
         if (escribeEnFichero == 1) {
            out.print(Arrays.toString(base) + " : ");
         }
         System.out.print(Arrays.toString(base) + " : ");

         //CRITERIO DE ENTRADA
         int[] critEntrada = criterioEntrada2();
         if (critEntrada[0] == 1) {
            break;
         }
         int ENTRA = critEntrada[1];

         //CRITERIO DE SALIDA
         int[] critSalida = criterioSalida(ENTRA);
         if (critSalida[0] == 1) {
            System.out.println("Solución no acotada");
            System.exit(0);
         }
         int SALE = critSalida[1];

         //SE RECALCULAN A, b Y C:
         SimpleMatrix[] nuevosdatos = this.recalcula2();
```

```
            B2 = nuevosdatos[0];
            C_B = nuevosdatos[1];
            A = nuevosdatos[2];
            b = nuevosdatos[3];
            C_M = nuevosdatos[4];
            c1_2 = C_M.extractVector(true, 0);
            c2_2 = C_M.extractVector(true, 1);
        }

        if (escribeEnFichero == 1) {
            out.println("\nTiempo de ejecución: " +
                SimplexBiObjetivoMain.contador.getTime() + " milisegundos");
        }
        System.out.println("\nTiempo de ejecución: " +
            SimplexBiObjetivoMain.contador.getTime() + " milisegundos");
    }

    public int[] criterioEntrada2() { //La primera componente que devuelve es el
        criterio de parada
        double lambda;
        double lambdamax = 0;
        double componentec1;
        double componentec2;

        int ENTRA = -1; //Variable que entra en la base

        for (int i = 0; i < c2_2.numCols(); i++) {

            componentec1 = c1_2.get(i);
            componentec2 = c2_2.get(i);

            if (componentec2 < 0 && componentec1 >= 0) {

                lambda = (-componentec2) / (componentec1 - componentec2);

                if (lambda > lambdamax) {
                    lambdamax = lambda;
                    ENTRA = i;
                }

            }
        }

        double[] solucion = new double[A.numCols()];
        for (int j = 0; j < A.numCols(); j++) {
            for (int l = 0; l < base.length; l++) {
                if (base[l] == j) {
                    solucion[j] = b.get(l);
                }
            }
        }

        if (ENTRA == -1) {
            lambdamax = 0;
        }

        if (escribeEnFichero == 1) {
            out.println(Arrays.toString(solucion) + " : " +
```

```
            Arrays.toString(calculaVFO(solucion)) + ", para todo lambda en [" +
            lambdamax + "," + lambdahat + "]");
      }

      System.out.println(Arrays.toString(solucion) + " : " +
          Arrays.toString(calculaVFO(solucion)) + ", para todo lambda en [" +
          lambdamax + "," + lambdahat + "]");
      lambdahat = lambdamax;

      if (ENTRA == -1) {
         return new int[]{1, 0};
      }

      return new int[]{0, ENTRA};
   }

   public SimpleMatrix[] recalcula2() {
      //Nueva matriz B:
      for (int i = 0; i < base.length; i++) {
         SimpleMatrix columnaB = AOriginal.extractVector(false, base[i]);
         B2.insertIntoThis(0, i, columnaB);
      }

      //Nuevos costos básicos:
      for (int i = 0; i < 2; i++) {
         for (int j = 0; j < m; j++) {
            C_B.set(i, j, C.get(i, base[j]));
         }
      }

      //Nueva matriz A:
      A = B2.solve(AOriginal);

      //Nuevo vector b:
      b = B2.solve(bOriginal);

      //Nueva matriz de costos marginales:
      C_M = COriginal.minus(C_B.mult(A));

      return new SimpleMatrix[]{B2, C_B, A, b, C_M};
   }

   /**
   * ************************************************************************
   * VARIOS
   * ************************************************************************
   */
   public void creaFichero(String nombre) {
      escribeEnFichero = 1;
      try {
         out = new PrintWriter(new FileWriter(nombre), true);
      } catch (IOException ex) {
         Logger.getLogger(simplexBiObjective.class.getName()).log(Level.SEVERE, null,
             ex);
      }
   }

   public double[] calculaVFO(SimpleMatrix solucion) {
```

```
      double[] vfo = new double[C.numRows()];

      //Si el problema es de máximo, se multiplica C por -1 de nuevo
      if (dirOptimization == maximizacion) {
         SimpleMatrix menosC = C.negative();
         for (int m = 0; m < C.numRows(); m++) {
            SimpleMatrix objetivo_m = menosC.extractVector(true, m).mult(solucion);
            vfo[m] = objetivo_m.get(0, 0);
         }
      } else {
         for (int m = 0; m < C.numRows(); m++) {
            SimpleMatrix objetivo_m = C.extractVector(true, m).mult(solucion);
            vfo[m] = objetivo_m.get(0, 0);
         }
      }
      return vfo;
   }

   public double[] calculaVFO(double[] solucion) {
      SimpleMatrix solucion2 = new SimpleMatrix(solucion.length, 1, true, solucion);
      return calculaVFO(solucion2);
   }
}
```

Anexo C

Clase SimplexMultiObjetivoMain.java:

```
package simplexmultiobjetivo;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.lang3.time.StopWatch;
import java.util.TreeSet;

/*Esta clase lee los datos de un problema de programación lineal multiobjetivo a
través de un fichero de texto y llama a la clase simplexMultiObjective para que
trate de resolverlo*/
public class SimplexMultiObjetivoMain {

   public static StopWatch contador;

   public static void main(String[] args) {

      contador = StopWatch.createStarted();

      //Lectura de datos del problema a través de un fichero de texto(El nombre del
          mismo es el primer argumento de main)
      File fentrada = new File(args[0]);
      Scanner entrada = null;
      String linea = null;

      try {
         entrada = new Scanner(fentrada);
         //Se leen las dimensiones de A y el número de funciones objetivo
         int m = entrada.nextInt();
         int n = entrada.nextInt();
         int o = entrada.nextInt();
         //Se lee el tipo del problema (min o max)
         entrada.nextLine();
         String tipo = entrada.next();
         int dirOpt = 0;
         if (tipo.equals("max")) {
            dirOpt = simplexUniObjective.maximizacion;
         } else if (tipo.equals("min")) {
            dirOpt = simplexUniObjective.minimizacion;
```

```
} else {
   System.out.println("Tipo de problema no válido");
   System.exit(0);
}
//Se lee la matriz A
double a[][] = new double[m][n];
for (int i = 0; i < m; i++) {
   entrada.nextLine();
   for (int j = 0; j < n; j++) {
      a[i][j] = entrada.nextDouble();
   }
}
//Se lee la matriz C
double c[][] = new double[o][n];
for (int i = 0; i < o; i++) {
   entrada.nextLine();
   for (int j = 0; j < n; j++) {
      c[i][j] = entrada.nextDouble();
   }
}
//Se lee el vector b
double b[] = new double[m];
entrada.nextLine();
for (int i = 0; i < m; i++) {
   b[i] = entrada.nextDouble();
}
//Se leen los tipos de las restricciones
int tipoRest[] = new int[m];
int tiporestr;
entrada.nextLine();
for (int i = 0; i < m; i++) {
   tiporestr = entrada.nextInt();
   if (tiporestr == -1) {
      tipoRest[i] = simplexUniObjective.menor;
   } else if (tiporestr == 0) {
      tipoRest[i] = simplexUniObjective.igualdad;
   } else if (tiporestr == 1) {
      tipoRest[i] = simplexUniObjective.mayor;
   } else {
      System.out.println("Tipo de restricción no válido");
      System.exit(0);
   }
}

//Creación y resolución del problema
simplexMultiObjective smo = new simplexMultiObjective(a, b, c, tipoRest,
    dirOpt);
smo.solve();

//Escritura de los datos de salida en un fichero (En el caso de que se le
    pase un segundo argumento a la función main)
if (args.length == 2) {
   PrintWriter out = null;
   try {
      out = new PrintWriter(new FileWriter(args[1]), true);
      TreeSet<Base> L2 = smo.getL2();
      out.println("Bases eficientes, soluciones respectivas y valor de la
          función objetivo: ");
```

```
            for (Base bb : L2) {
               out.println(Arrays.toString(bb.getBase()) + " : " +
                   Arrays.toString(bb.getX()) + " : " +
                   Arrays.toString(bb.getVFO()));
            }
            out.println("\nTiempo de ejecución: " + contador.getTime() + "
                milisegundos");

         } catch (IOException ex) {
            Logger.getLogger(SimplexMultiObjetivoMain.class.getName()).log(Level.SEVERE,
                null, ex);
         }
      }
   } catch (FileNotFoundException ex) {
      Logger.getLogger(SimplexMultiObjetivoMain.class.getName()).log(Level.SEVERE,
          null, ex);
   }
}
}
```

Clase simplexMultiObjective.java:

```
package simplexmultiobjetivo;

import static java.lang.Boolean.TRUE;
import java.util.Arrays;
import java.util.Comparator;
import java.util.TreeSet;
import org.ejml.simple.SimpleMatrix;
import static simplexmultiobjetivo.SimplexMultiObjetivoMain.contador;

/*Esta clase recibe la información de un problema de programación lineal
multiobjetivo y trata de resolverlo*/
public class simplexMultiObjective {

   private int m, n, o;
   private int base[];
   private int numHolguras;

   private SimpleMatrix A;
   private SimpleMatrix b;
   private SimpleMatrix C;

   private SimpleMatrix Aoriginal;
   private SimpleMatrix boriginal;
   private SimpleMatrix Coriginal;
   private SimpleMatrix hatW;

   private simplexUniObjective fase1;
   private simplexUniObjective fase2_1;
   private simplexUniObjective fase2_2;
   private simplexUniObjective fase3;
   private simplexUniObjective fase3_j;

   private double[][] Ao;
   private double[] bo;
   private double co[][];
   private int[] typeConstraint;
   private int dirOptimization;

   private TreeSet L2;

   public simplexMultiObjective(double a[][], double bb[], double cc[][], int
       typeConst[], int dirOpt) {

      //Si el problema es de máximo, se multiplica C por -1 para hacerlo de mínimo
      if (dirOpt == simplexUniObjective.maximizacion) {
         for (int i = 0; i < cc.length; i++) {
            for (int j = 0; j < cc[0].length; j++) {
               cc[i][j] = -cc[i][j];
            }
         }
      }

      Ao = a;
      bo = bb;
      co = cc;
      typeConstraint = typeConst;
```

```
      dirOptimization = dirOpt;
      o = cc.length; //Número de objetivos

      Aoriginal = new SimpleMatrix(a);
      boriginal = new SimpleMatrix(bb.length, 1, true, bb);
      Coriginal = new SimpleMatrix(cc);

      hatW = new SimpleMatrix(o, 1);
   }

   public void solve() {
      /*FASE 1 del algoritmo: Se resuelve la fase 1 del metodo de la doble fase para
          saber si el problema es factible, y en caso de serlo tener
      una solución factible básica del problema*/
      fase1 = new simplexUniObjective(Ao, bo, co[0], typeConstraint, dirOptimization);
          //Se le pasa c[0], pero en la fase 1 del doble fase se ignora

      if (fase1.getPhase1needed()) {
         int res = fase1.doPhase1();
         if (res != simplexUniObjective.problemaFactible) {
            System.out.println("Problema no factible");
            System.exit(res);
         }
      }

      //Se obtienen de fase1 los elementos originales del problema
      A = fase1.getAoriginal();
      b = fase1.getbOriginal();
      C = new SimpleMatrix(co.length, fase1.getN() + fase1.getNumHolguras());
      SimpleMatrix CO = new SimpleMatrix(co);
      SimpleMatrix cerosCO = new SimpleMatrix(co.length, fase1.getNumHolguras());
      CO = CO.concatColumns(cerosCO);
      for (int i = 0; i < o; i++) {
         for (int j = 0; j < fase1.getN(); j++) {
            C.set(i, j, CO.get(i, j));
         }
      }

      //FASE 2: Si la FASE 1 ha proporcionado una SFB, resolvemos los problemas
          auxiliares
      doPhase2Aux1();
      doPhase2Aux2();

      //FASE 3: Partiendo de una solución óptima, encontramos las demás
      doPhase3();
   }

   /**
   * ************************************************************************
   * FASE 2
   * ************************************************************************
   */
   public void doPhase2Aux1() {
      //Función objetivo:
      SimpleMatrix u1 = b.transpose();
      SimpleMatrix x_0 = fase1.getX_0().transpose();
      SimpleMatrix w = C.mult(x_0).transpose();
      SimpleMatrix cUW = u1.concatColumns(u1.negative()).concatColumns(w);
```

```
    //Matriz de coeficientes:
    SimpleMatrix A1 = A.transpose();
    SimpleMatrix AUW = A1.concatColumns(A1.negative()).concatColumns(C.transpose());

    //Vector de recursos:
    SimpleMatrix bUW = new SimpleMatrix(C.numCols(), 1);

    SimpleMatrix fila = new SimpleMatrix(1, C.numCols());
    for (int k = 0; k < C.numCols(); k++) {
       fila = C.transpose().extractVector(TRUE, k);
       // TIENE EL SIGNO CAMBIADO
       bUW.set(k, -fila.elementSum());
    }

    //Cambiamos de signo las restricciones con recursos < 0:
    int[] tipoRestr = new int[A.numCols()]; //0 si es >=, 1 si es <=

    SimpleMatrix cerosFila = new SimpleMatrix(1, A.numRows() * 3);
    SimpleMatrix filaAUW = new SimpleMatrix(1, AUW.numCols());

    for (int l = 0; l < A.numCols(); l++) {
       if (bUW.get(l) < 0) {
          tipoRestr[l] = simplexUniObjective.menor;
          bUW.set(l, -bUW.get(l));
          filaAUW = AUW.extractVector(TRUE, l).negative();
          AUW.insertIntoThis(l, 0, filaAUW);
       } else {
          tipoRestr[l] = simplexUniObjective.mayor;
       }
    }

    //Fin de preparación del problema auxiliar 1 de la fase 2
    //Lo resolvemos:
    fase2_1 = simplexUniObjective.createSimplexUniObjective(AUW, bUW, cUW,
        tipoRestr, simplexUniObjective.minimizacion);
    fase2_1.solve();

    SimpleMatrix hatU = new SimpleMatrix(A.numRows(), 1);
    for (int i = 0; i < A.numRows(); i++) {
       hatU.set(i, 0, fase2_1.getX_0().get(0, i) - fase2_1.getX_0().get(0, i +
           A.numRows()));
    }

    for (int i = 0; i < o; i++) {
       hatW.set(i, 0, 1 + fase2_1.getX_0().get(0, i + 2 * A.numRows()));
    }
 }

 public void doPhase2Aux2() {
    //Función objetivo
    SimpleMatrix cAux2 = hatW.transpose().mult(Coriginal);
    //Tenemos en cuenta que x_0 es una SFB de este problema
    simplexUniObjective fase2_2 = (simplexUniObjective) fase1.clone();
    fase2_2.solveSoloFase2(cAux2);
    //Copiamos los datos con la solución óptima para la fase 3
    fase3 = (simplexUniObjective) fase2_2.clone();
 }
```

```
/**
* ************************************************************************
* FASE 3
* ************************************************************************
*/
public void doPhase3() {
   //Definimos las listas que almacenarán las bases
   TreeSet<Base> L1 = new TreeSet<>();
   TreeSet<Base> L2 = new TreeSet<>();

   Base basePrimera = new Base(fase3.getBase());
   basePrimera.setX(fase3.getX_0().transpose());
   basePrimera.setVFO(calculaVFO(fase3.getX_0().transpose()));
   L1.add(basePrimera);

   int[] baseActual;
   SimpleMatrix B2 = new SimpleMatrix(A.numRows(), A.numRows());
   SimpleMatrix b2 = new SimpleMatrix(A.numRows(), 1);
   int N = A.numCols() - A.numRows();

   while (L1.size() != 0) {
      Base aux = L1.pollFirst();
      baseActual = aux.getBase();
      double[] solucionActual = aux.getX();
      int[] baseActual2 = Arrays.copyOf(baseActual, baseActual.length);
      Base baseNueva = new Base(baseActual2);
      baseNueva.setX(solucionActual);
      baseNueva.setVFO(calculaVFO(solucionActual));

      if (!isInL(baseNueva, L2)) {
         L2.add(baseNueva);
      }

      //Recalculamos A, b y R=C_N según esta base:
      for (int i = 0; i < baseActual.length; i++) {
         SimpleMatrix columnaB = fase3.getAoriginal().extractVector(false,
             baseActual[i]);
         B2.insertIntoThis(0, i, columnaB);
      }

      A = B2.solve(fase3.getAoriginal());
      b = B2.solve(fase3.getbOriginal());

      SimpleMatrix C_B = new SimpleMatrix(C.numRows(), baseActual.length);
      for (int i = 0; i < baseActual.length; i++) {
         SimpleMatrix columnaC = C.extractVector(false, baseActual[i]);
         C_B.insertIntoThis(0, i, columnaC);
      }

      SimpleMatrix C_barra = C.minus(C_B.mult(A));

      SimpleMatrix R = new SimpleMatrix(C.numRows(), N);
      int basica = 0;
      int[] noBasicas = new int[N];
      int contador = 0;
      for (int i = 0; i < C.numCols(); i++) {
         for (int j = 0; j < baseActual.length; j++) {
```

```
            if (baseActual[j] == i) {
               basica = 1;
            }
         }

         if (basica != 1) {
            SimpleMatrix columnaCM = C_barra.extractVector(false, i);
            R.insertIntoThis(0, contador, columnaCM);
            noBasicas[contador] = i;
            contador++;
         }
         basica = 0;
      }

      for (int j = 0; j < N; j++) {
         //Resolvemos el problema auxiliar para cada j:
         SimpleMatrix r_j = R.extractVector(false, j);
         SimpleMatrix A3 = R.copy().concatColumns(r_j.negative()).concatColumns(
         SimpleMatrix.identity(R.numRows()));

         SimpleMatrix b3 = new SimpleMatrix(R.numRows(), 1);

         SimpleMatrix c3 = new SimpleMatrix(1, N + 1);
         SimpleMatrix unos = new SimpleMatrix(1, C.numRows());
         unos.fill(1);
         c3 = c3.concatColumns(unos);

         int[] tipoRestr = new int[R.numRows()];
         Arrays.fill(tipoRestr, simplexUniObjective.igualdad);

         fase3_j = simplexUniObjective.createSimplexUniObjective(A3, b3, c3,
             tipoRestr, simplexUniObjective.maximizacion);
         if (fase3_j.solve() == simplexUniObjective.problemaNoAcotado) {
            noBasicas[j] = -1;
         }
      }

      for (int j = 0; j < N; j++) {
         for (int i = 0; i < baseActual.length; i++) {

            if (noBasicas[j] != -1) {
               //Entra la variable nobasicas[j] por baseActual[i]
               baseActual[i] = noBasicas[j];

               simplexUniObjective fase3mod = (simplexUniObjective) fase3.clone();

               //Criterio de factibilidad: Si A_B es regular y (A_B)^(-1)*b >= 0, el
                   pivotaje es factible
               SimpleMatrix A_B = new SimpleMatrix(A.numRows(), A.numRows());
               for (int k = 0; k < baseActual.length; k++) {
                  SimpleMatrix columnaA_B =
                      fase3.getAoriginal().extractVector(false, baseActual[k]);
                  A_B.insertIntoThis(0, k, columnaA_B);
               }

               int ALMACENAR = 1;

               if (A_B.determinant() == 0) {
```

```
                    ALMACENAR = 0;
                }

                if (ALMACENAR == 1) {
                    b2 = A_B.invert().mult(fase3mod.getbOriginal());

                    for (int l = 0; l < b2.numRows(); l++) {
                        if (b2.get(l) < 0) {
                            ALMACENAR = 0;
                        }
                    }
                }

                if (ALMACENAR == 1) { //El pivotaje es factible
                    int[] baseActual3 = Arrays.copyOf(baseActual, baseActual.length);
                    Base base3 = new Base(baseActual3);

                    SimpleMatrix solucionCompleta = new SimpleMatrix(A.numCols(), 1);
                    for (int l = 0; l < solucionCompleta.numRows(); l++) {
                        for (int m = 0; m < baseActual.length; m++) {
                            if (baseActual3[m] == l) {
                                solucionCompleta.set(l, b2.get(m, 0));
                            }
                        }

                    }
                    base3.setX(solucionCompleta);
                    base3.setVFO(calculaVFO(solucionCompleta));

                    if (!isInL(base3, L1) && !isInL(base3, L2)) {
                        L1.add(base3);
                    }
                }
                baseActual[i] = baseActual2[i];
            }
        }
    }
}

this.L2 = L2;

//Se sacan por pantalla las bases y soluciones eficientes obtenidas
System.out.println("Bases eficientes, soluciones respectivas y valor de la
    función objetivo: ");
L2.forEach(bb -> {
    System.out.println(Arrays.toString(bb.getBase()) + " : " +
        Arrays.toString(bb.getX()) + " : " + Arrays.toString(bb.getVFO()));
});
System.out.println("\nTiempo de ejecución: " + contador.getTime() + "
    milisegundos");
}

/**
* ********************************************************************
* VARIOS
* ********************************************************************
*/
public TreeSet getL2() {
```

```
      return L2;
   }

   public double[] calculaVFO(SimpleMatrix solucion) {
      double[] vfo = new double[C.numRows()];

      //Si el problema es de máximo, se multiplica C por -1 de nuevo
      if (dirOptimization == simplexUniObjective.maximizacion) {
         SimpleMatrix menosC = C.negative();
         for (int m = 0; m < C.numRows(); m++) {
            SimpleMatrix objetivo_m = menosC.extractVector(true, m).mult(solucion);
            vfo[m] = objetivo_m.get(0, 0);
         }
      } else {
         for (int m = 0; m < C.numRows(); m++) {
            SimpleMatrix objetivo_m = C.extractVector(true, m).mult(solucion);
            vfo[m] = objetivo_m.get(0, 0);
         }
      }
      return vfo;
   }

   public double[] calculaVFO(double[] solucion) {
      SimpleMatrix solucion2 = new SimpleMatrix(solucion.length, 1, true, solucion);
      return calculaVFO(solucion2);
   }

   boolean isInL(Base t, TreeSet<Base> L) {
      int tt[] = Arrays.copyOf(t.getBase(), t.getBase().length);
      int bb[];
      Arrays.sort(tt);
      boolean esta = true;
      for (Base b : L) {
         bb = Arrays.copyOf(b.getBase(), b.getBase().length);
         Arrays.sort(bb);
         esta = true;
         for (int i = 0; i < tt.length; i++) {
            if (tt[i] != bb[i]) {
               esta = false;
               break;
            }
         }
         if (esta) {
            return esta;
         }
      }
      return false;
   }
}
```

Clase UniObjective.java:

```
package simplexmultiobjetivo;

import org.ejml.simple.SimpleMatrix;

/*Esta clase recibe la información de un problema de programación lineal y
trata de resolverlo (se define una instancia para cada problema distinto)*/
public class simplexUniObjective {

   private int recursosNegativosError = 1;
   public static final int igualdad = 0;
   public static final int menor = -1;
   public static final int mayor = 1;
   public static final int minimizacion = 1;
   public static final int maximizacion = 2;
   public static final int problemaNoAcotado = 2;
   public static final int problemaNoFactible = 1;
   public static final int problemaFactible = 0;
   public static final double epsilon = 1E-13;
   private int m, n;

   private int base[];
   private int numHolguras;
   private int numArtificiales;
   private boolean phase1needed = false;

   private SimpleMatrix A;
   private SimpleMatrix Aoriginal;
   private SimpleMatrix b;
   private SimpleMatrix bOriginal;
   private SimpleMatrix C;
   private SimpleMatrix C_M; //Costos marginales
   private SimpleMatrix C_B; //costos básicos

   //Fase uno de método de la doble fase
   private SimpleMatrix C_BPhase1;
   private SimpleMatrix cFase1;
   private SimpleMatrix cMFase1;
   private int nPhase1;

   private boolean solved = false;

   private simplexUniObjective() {
   }

   public simplexUniObjective(double a[][], double bb[], double cc[], int
       typeConstraint[], int dirOptimizacion) {

      for (double elem : bb) {
         if (elem < 0) {
            System.out.println("El vector de recursos debe ser >=0");
            System.exit(recursosNegativosError);
         }
      }
      m = bb.length;
      n = a[0].length;
      base = new int[m];
```

```
for (int i : typeConstraint) {
   switch (i) {
      case igualdad:
      numArtificiales++;
      break;
      case menor:
      numHolguras++;
      break;
      case mayor:
      numHolguras++;
      numArtificiales++;
      break;
   }
}
A = new SimpleMatrix(bb.length, getN() + getNumArtificiales() + numHolguras);
Aoriginal = new SimpleMatrix(bb.length, getN() + numHolguras);
for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++) {
      A.set(i, j, a[i][j]);
      Aoriginal.set(i, j, a[i][j]);
   }
}

int colOffset = 0;
for (int i = 0; i < m; i++) {
   switch (typeConstraint[i]) {
      case menor:
      A.set(i, n + colOffset, 1);
      Aoriginal.set(i, n + colOffset, 1);
      base[i] = n + colOffset;
      colOffset++;
      break;
      case mayor:
      A.set(i, n + colOffset, -1);
      Aoriginal.set(i, n + colOffset, -1);
      colOffset++;
      break;
   }
}
for (int i = 0; i < m; i++) {
   if (typeConstraint[i] == igualdad || typeConstraint[i] == mayor) {
      A.set(i, n + colOffset, 1);
      base[i] = n + colOffset;
      colOffset++;
   }
}

b = new SimpleMatrix(bb.length, 1, true, bb);
bOriginal = new SimpleMatrix(bb.length, 1, true, bb);
C = new SimpleMatrix(1, getN() + numHolguras);
C_M = new SimpleMatrix(1, getN() + numHolguras);
for (int j = 0; j < n; j++) {
   C.set(0, j, cc[j]);
}

if (dirOptimizacion == maximizacion) {
   C = C.negative();
```

```
    }
    if (numArtificiales > 0) {//Es necesario fase 1
       cFase1 = new SimpleMatrix(1, getN() + getNumArtificiales() + numHolguras);
       for (int j = n + numHolguras; j < n + numArtificiales + numHolguras; j++) {
          cFase1.set(0, j, 1);
       }
       C_BPhase1 = new SimpleMatrix(1, getM());
       for (int i = 0; i < m; i++) {
          if (base[i] >= n + numHolguras) {
             C_BPhase1.set(0, i, 1);
          }
       }
       //Calculamos los costos marginales
       cMFase1 = cFase1.minus(C_BPhase1.mult(A));
       nPhase1 = n + numArtificiales + numHolguras;
       phase1needed = true;
    }
 }

 public static simplexUniObjective createSimplexUniObjective(SimpleMatrix aa,
     SimpleMatrix bb, SimpleMatrix cc, int typeConstraint[], int dirOptimizacion) {
    double[][] a = new double[aa.numRows()][aa.numCols()];
    for (int i = 0; i < aa.numRows(); i++) {
       for (int j = 0; j < aa.numCols(); j++) {
          a[i][j] = aa.get(i, j);
       }
    }
    double[] b = new double[bb.numRows()];
    for (int i = 0; i < b.length; i++) {
       b[i] = bb.get(i, 0);
    }
    double[] c = new double[cc.numCols()];
    for (int j = 0; j < cc.numCols(); j++) {
       c[j] = cc.get(0, j);
    }

    return new simplexUniObjective(a, b, c, typeConstraint, dirOptimizacion);
 }

 /**
 * **********************
 * FASE 1 **********************
 */
 int doPhase1() {
    int in;
    int out;
    int iteration = 1;
    while (!isPhase1Optimal()) {
       in = whichEntersPhase1();
       out = whichLeavesPhase1and2(in);
       if (out == -1) {
          System.out.println("Problema no factible");
          return (problemaNoFactible);
       }
       doPivotPhase1(in, out);
    }
    return problemaFactible;
 }
```

```
//-1 => solucion optima
int whichEntersPhase1() {
   double minValue = Double.MAX_VALUE;
   int minPos = -1;

   int noBasica[] = new int[nPhase1];
   for (int i = 0; i < noBasica.length; i++) {
      noBasica[i] = i;
   }
   for (int i = 0; i < base.length; i++) {
      noBasica[base[i]] = -1;
   }
   for (int i = 0; i < nPhase1; i++) {
      if (noBasica[i] == -1) {
         continue;
      }
      if (cMFase1.get(0, i) < minValue) {
         minValue = cMFase1.get(0, i);
         minPos = i;
      }
   }

   return minPos;
}

//-1 problema no acotado
int whichLeavesPhase1and2(int in) {
   double minVal = Double.MAX_VALUE;
   int minPos = -1;
   SimpleMatrix colIn = A.extractVector(false, in);

   for (int i = 0; i < getM(); i++) {
      if (colIn.get(i, 0) > 0 && b.get(i, 0) / colIn.get(i, 0) < minVal) {
         minVal = b.get(i, 0) / colIn.get(i, 0);
         minPos = i;
      }
   }
   return minPos;
}

boolean isPhase1Optimal() {
   int noBasica[] = new int[nPhase1];
   for (int i = 0; i < noBasica.length; i++) {
      noBasica[i] = i;
   }
   for (int i = 0; i < base.length; i++) {
      noBasica[base[i]] = -1;
   }
   for (int i = 0; i < nPhase1; i++) {
      if (noBasica[i] == -1) {
         continue;
      }
      if (cMFase1.get(0, i) < -epsilon) {
         return false;
      }
   }
   return true;
```

```
}

void doPivotPhase1(int in, int out) {
   //Se actualiza la base, A, b y los costos marginales
   SimpleMatrix elem = SimpleMatrix.identity(getM());
   double Yst = 1.0 / A.get(out, in);
   for (int i = 0; i < getM(); i++) {
      if (i != out) {
         elem.set(i, out, -A.get(i, in) * Yst);
      } else {
         elem.set(i, out, Yst);
      }
   }
   //Se actualiza A
   A = elem.mult(A);
   //Se actualiza b
   b = elem.mult(b);
   //Se actualiza base: cambiamos base[out] por in
   base[out] = in;
   //Se actualiza el vector de costos de variables básicas
   C_BPhase1.set(0, out, cFase1.get(in));
   //Se actualizan los costos marginales
   cMFase1 = cFase1.minus(C_BPhase1.mult(A));
}

/**
* **********************
* FASE 2 **********************
*/
/* Se encarga de cargar los datos para el problema a resolver, ahora que
ya sabemos que es factible y que ya tenemos una base inicial.
Solo es necesario eliminar las variables artificiales, la parte correspondiente
en la matriz A, y calcular los primeros costos marginales C_M */
void preparingPhase2() {
   //Se quitan las columnas de las variables artificiales
   if (phase1needed) {
      A = A.extractMatrix(0, getM(), 0, getN() + numHolguras);
   }

   //Se calcula la matriz de costos básicos
   C_B = new SimpleMatrix(1, getM());
   for (int j = 0; j < getM(); j++) {
      C_B.set(0, j, C.get(0, base[j]));
   }

   //Se calcula la matriz de costos marginales
   C_M = C.minus(C_B.mult(A));
}

public int doPhase2() {
   int in;
   int out;
   int iteration = 1;
   while (!isPhase2Optimal()) {
      in = whichEntersPhase2();
      out = whichLeavesPhase1and2(in);
      if (out == -1) {
         //PROBLEMA NO ACOTADO
```

```
            return (problemaNoAcotado);
         }
         doPivotPhase2(in, out);
      }
      return problemaFactible;
   }

   boolean isPhase2Optimal() {
      int noBasica[] = new int[getN() + numHolguras];
      for (int i = 0; i < noBasica.length; i++) {
         noBasica[i] = i;
      }
      for (int i = 0; i < base.length; i++) {
         noBasica[base[i]] = -1;
      }
      for (int i = 0; i < getN() + numHolguras; i++) {
         if (noBasica[i] == -1) {
            continue;
         }
         if (C_M.get(0, i) < -epsilon) {
            return false;
         }
      }
      return true;
   }

   //-1 => solucion optima
   int whichEntersPhase2() {
      double minValue = Double.MAX_VALUE;
      int minPos = -1;

      int noBasica[] = new int[getN() + numHolguras];
      for (int i = 0; i < noBasica.length; i++) {
         noBasica[i] = i;
      }
      for (int i = 0; i < base.length; i++) {
         noBasica[base[i]] = -1;
      }
      for (int i = 0; i < getN() + numHolguras; i++) {
         if (noBasica[i] == -1) {
            continue;
         }
         if (C_M.get(0, i) < minValue) {
            minValue = C_M.get(0, i);
            minPos = i;
         }
      }
      return minPos;
   }

   void doPivotPhase2(int in, int out) {
      //Se actualiza la base, A, b y los costos marginales
      SimpleMatrix elem = SimpleMatrix.identity(getM());
      double Yst = 1.0 / A.get(out, in);
      for (int i = 0; i < getM(); i++) {
         if (i != out) {
            elem.set(i, out, -A.get(i, in) * Yst);
         } else {
```

```
            elem.set(i, out, Yst);
         }
      }
      //Se actualiza A
      A = elem.mult(A);
      //Se actualiza b
      b = elem.mult(b);
      //Se actualiza la base
      base[out] = in;
      //Se actualiza el vector de costos de variables básicas
      C_B.set(0, out, C.get(in));
      //Se actualizan los costos marginales
      C_M = C.minus(C_B.mult(A));
   }

   int solveSoloFase2(SimpleMatrix C) {
      SimpleMatrix objetivosHolguras = new SimpleMatrix(1, numHolguras);
      C = C.concatColumns(objetivosHolguras);
      this.C = C;

      int res = simplexUniObjective.problemaFactible;
      preparingPhase2();
      res = doPhase2();
      if (res != simplexUniObjective.problemaFactible) {
         return res;
      }
      solved = true;
      return res;
   }

   /**
   * ******************
   * GLOBAL ******************
   */
   int solve() {
      int res = simplexUniObjective.problemaFactible;
      if (phase1needed) {
         res = doPhase1();
      }
      if (res != simplexUniObjective.problemaFactible) {
         return res;
      }
      preparingPhase2();
      res = doPhase2();
      if (res != simplexUniObjective.problemaFactible) {
         return res;
      }
      solved = true;
      return res;
   }

   public simplexUniObjective clone() {
      simplexUniObjective nuevo = new simplexUniObjective();
      nuevo.m = this.m;
      nuevo.n = this.n;
      nuevo.numHolguras = this.numHolguras;
      nuevo.base = this.base;
      nuevo.A = this.A.copy();
```

```
        nuevo.Aoriginal = this.Aoriginal.copy();
        nuevo.b = this.b.copy();
        nuevo.bOriginal = this.bOriginal.copy();
        nuevo.C = this.C.copy();
        nuevo.C_M = this.C_M.copy();

        return this;
    }

    public SimpleMatrix getbOriginal() {
        return bOriginal;
    }

    public SimpleMatrix getAoriginal() {
        return Aoriginal;
    }

    public int getNumArtificiales() {
        return numArtificiales;
    }

    public double getVFO() {
        return C.mult(this.getX_0().transpose()).get(0);
    }

    public SimpleMatrix getA() {
        return A;
    }

    public SimpleMatrix getb() {
        return b;
    }

    public SimpleMatrix getC_M() {
        return C_M;
    }

    public int getM() {
        return m;
    }

    public int getN() {
        return n;
    }

    public int getNumHolguras() {
        return numHolguras;
    }

    public SimpleMatrix getX_0() {
        SimpleMatrix x0 = new SimpleMatrix(1, getN() + this.numHolguras);
        for (int i = 0; i < getM(); i++) {
            x0.set(0, base[i], b.get(i, 0));
        }
        return x0;
    }

    public int[] getBase() {
```

```
      return base;
   }

   public boolean getPhase1needed() {
      return phase1needed;
   }

}
```

Clase Base.java:

```
package simplexmultiobjetivo;

import java.util.Arrays;
import org.ejml.simple.SimpleMatrix;

/*Cada objeto de esta clase representará una base eficiente del problema*/
public class Base implements Comparable<Base> {

   private int[] base;
   private double[] X;
   private double[] valorFuncionObjetivo;

   public Base(int[] base) {
      this.base = base;
   }

   @Override
   public int compareTo(Base b) {
      int contadorIguales = 0;

      for (int i = 0; i < base.length; i++) {
         if (b.base[i] == this.base[i]) {
            contadorIguales++;
         }
         if (b.base[i] < this.base[i]) {
            return 1;
         }
      }

      if (contadorIguales == base.length) {
         return 0;
      }

      return -1;
   }

   public int[] getBase() {
      return base;
   }

   public double[] getX() {
      return X;
   }

   public double[] getVFO() {
      return valorFuncionObjetivo;
   }

   public void setX(double x[]) {
      this.X = x;
   }

   public void setX(SimpleMatrix x) {
      double[] aux = new double[x.numRows()];
      for (int i = 0; i < x.numRows(); i++) {
         aux[i] = x.get(i, 0);
```

```
      }
      this.X = Arrays.copyOf(aux, aux.length);
   }

   public void setVFO(double[] valorFuncionObjetivo) {
      this.valorFuncionObjetivo = valorFuncionObjetivo;
   }
}
```

Anexo D

Presentamos en primer lugar un resumen de los resultados obtenidos:

Problema	Resuelto con	Nº de bases eficientes	Tiempo de computación
1 (Ref. [2])	Simplex biobjetivo	4	85 milisegundos
2 (Ref. [4])	Simplex biobjetivo	7	81 milisegundos
3 (Ref. [4])	Simplex general	11	124 milisegundos
4 (Ref. [7])	Simplex general	18	143 milisegundos
5 (Ref. [8])	Simplex general	51	251 milisegundos

Todos los resultados a continuación se muestran redondeados a 2 cifras decimales. Se resuelve, en primer lugar y con el primer programa (simplex biobjetivo), el problema 1 (de mínimo, con restricciones de tipo "$\geq$").

Los datos del mismo son: $A = \begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}$, $b = \begin{pmatrix} 4 \\ 3 \\ 4 \\ -5 \\ -5 \end{pmatrix}$, $C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Antes de introducirlos en el programa, cambiamos de signo las dos últimas restricciones para tener $b \geq 0$. Se obtiene la salida:

Bases eficientes, soluciones respectivas y valor de la función objetivo:
[3, 4, 1, 5, 6] : [0.0, 4.0, 0.0, 1.0, 4.0, 5.0, 1.0] : [0.0, 4.0], para todo lambda en [0.67,1.0]
[0, 4, 1, 5, 6] : [1.0, 2.0, 0.0, 0.0, 1.0, 4.0, 3.0] : [1.0, 2.0], para todo lambda en [0.5,0.67]
[0, 2, 1, 5, 6] : [2.0, 1.0, 1.0, 0.0, 0.0, 3.0, 4.0] : [2.0, 1.0], para todo lambda en [0.33,0.5]
[0, 2, 3, 5, 6] : [4.0, 0.0, 4.0, 1.0, 0.0, 1.0, 5.0] : [4.0, 0.0], para todo lambda en [0.0,0.33]
Tiempo de ejecución: 85 milisegundos

Resolvemos seguidamente el problema 2 (de máximo, con restricciones de tipo "$\geq$"), de nuevo con el primer programa.

En este caso, los datos son : $A = \begin{pmatrix} 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 60 \\ 60 \\ 50 \\ 50 \end{pmatrix}$, $C = \begin{pmatrix} 3 & 1 & 2 & 1 \\ -1 & 5 & 1 & 2 \end{pmatrix}$.

Salida del programa:

Bases eficientes, soluciones respectivas y valor de la función objetivo:
[2, 5, 6, 0] : [6.67, 0.0, 11.67, 0.0, 0.0, 28.33, 8.33, 0.0] : [43.33, 5.0], para todo lambda en [0.86,1.0]
[2, 5, 3, 0] : [7.5, 0.0, 7.5, 5.0, 0.0, 20.0, 0.0, 0.0] : [42.5, 10.0], para todo lambda en [0.83,0.86]
[2, 5, 1, 0] : [2.5, 5.0, 12.5, 0.0, 0.0, 20.0, 0.0, 0.0] : [37.5, 35.0], para todo lambda en [0.77,0.83]

[2, 5, 1, 4] : [0.0, 10.0, 10.0, 0.0, 10.0, 10.0, 0.0, 0.0] : [30.0, 60.0], para todo lambda en [0.71,0.77]
[2, 3, 1, 4] : [0.0, 12.5, 5.0, 2.5, 20.0, 0.0, 0.0, 0.0] : [25.0, 72.5], para todo lambda en [0.33,0.71]
[2, 6, 1, 4] : [0.0, 14.0, 4.0, 0.0, 30.0, 0.0, 10.0, 0.0] : [22.0, 74.0], para todo lambda en [0.12,0.33]
[7, 6, 1, 4] : [0.0, 15.0, 0.0, 0.0, 45.0, 0.0, 20.0, 5.0] : [15.0, 75.0], para todo lambda en [0.0,0.12]
Tiempo de ejecución: 81 milisegundos

A continuación se muestran los otros tres problemas resueltos, esta vez con el programa que implementa el algoritmo simplex general.

El primero de ellos (Problema 3) será el que resulta de añadir una función objetivo más al anterior, $\max x_1 - x_2 + 2x_3 + 4x_4$, de modo que la matriz C tendrá una fila más: $(1,-1,2,4)$.
Manteniendo las mismas restricciones, la salida del programa es la siguiente:

Bases eficientes, soluciones respectivas y valor de la función objetivo:
[4, 1, 3, 0] : [5.0, 7.5, 0.0, 7.5, 20.0, 0.0, 0.0, 0.0] : [30.0, 47.5, 27.5]
[4, 1, 3, 2] : [0.0, 12.5, 5.0, 2.5, 20.0, 0.0, 0.0, 0.0] : [25.0, 72.5, 7.5]
[4, 1, 3, 7] : [0.0, 11.67, 0.0, 6.67, 28.33, 0.0, 0.0, 8.33] : [18.33, 71.67, 15.0]
[2, 5, 3, 0] : [7.5, 0.0, 7.5, 5.0, 0.0, 20.0, 0.0, 0.0] : [42.5, 10.0, 42.5]
[4, 5, 3, 0] : [10.0, 0.0, 0.0, 10.0, 10.0, 10.0, 0.0, 0.0] : [40.0, 10.0, 50.0]
[4, 5, 3, 7] : [0.0, 0.0, 0.0, 12.5, 22.5, 35.0, 0.0, 37.5] : [12.5, 25.0, 50.0]
[0, 1, 5, 2] : [2.5, 5.0, 12.5, 0.0, 0.0, 20.0, 0.0, 0.0] : [37.5, 35.0, 22.5]
[4, 1, 5, 2] : [0.0, 10.0, 10.0, 0.0, 10.0, 10.0, 0.0, 0.0] : [30.0, 60.0, 10.0]
[4, 1, 6, 2] : [0.0, 14.0, 4.0, 0.0, 30.0, 0.0, 10.0, 0.0] : [22.0, 74.0, -6.0]
[4, 1, 6, 7] : [0.0, 15.0, 0.0, 0.0, 45.0, 0.0, 20.0, 5.0] : [15.0, 75.0, -15.0]
[2, 5, 6, 0] : [6.67, 0.0, 11.67, 0.0, 0.0, 28.33, 8.33, 0.0] : [43.33, 5.0, 30.0]
Tiempo de ejecución: 124 milisegundos

Para el problema 4 (de máximo, con restricciones de tipo "$\leq$"), con datos:

$$A = \begin{pmatrix} 1 & 2 & 1 & 1 & 2 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 & 0 & 1 \\ -1 & 0 & 1 & 0 & 2 & 0 & -2 \\ 0 & 1 & 2 & -1 & 1 & -2 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} 16 \\ 16 \\ 16 \\ 16 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 2 & -1 & 3 & 2 & 0 & 1 \\ 0 & 1 & 1 & 2 & 3 & 1 & 0 \\ 1 & 0 & 1 & -1 & 0 & -1 & -1 \end{pmatrix},$$

se tiene que las soluciones eficientes son degeneradas (distintas bases dan la misma solución), por lo que a pesar de haber 6 soluciones eficientes, se obtienen 18 bases eficientes.

Bases eficientes, soluciones respectivas y valor de la función objetivo:
[0, 8, 9, 2] : [8.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 32.0, 16.0, 0.0] : [0.0, 8.0, 16.0]
[0, 2, 4, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[0, 3, 4, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[0, 3, 9, 10] : [0.0, 0.0, 0.0, 16.0, 0.0, 0.0, 0.0, 0.0, 0.0, 16.0, 32.0] : [48.0, 32.0, -16.0]
[0, 4, 9, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[0, 8, 4, 2] : [0.0, 0.0, 5.33, 0.0, 5.33, 0.0, 0.0, 0.0, 5.33, 0.0, 0.0] : [5.33, 21.33, 5.33]
[0, 8, 4, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[0, 8, 9, 10] : [16.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 48.0, 32.0, 16.0] : [16.0, 0.0, 16.0]
[3, 8, 9, 2] : [0.0, 0.0, 10.67, 5.33, 0.0, 0.0, 0.0, 0.0, 10.67, 5.33, 0.0] : [5.33, 21.33, 5.33]
[2, 3, 4, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[1, 3, 9, 10] : [0.0, 0.0, 0.0, 16.0, 0.0, 0.0, 0.0, 0.0, 0.0, 16.0, 32.0] : [48.0, 32.0, -16.0]
[2, 3, 9, 10] : [0.0, 0.0, 0.0, 16.0, 0.0, 0.0, 0.0, 0.0, 0.0, 16.0, 32.0] : [48.0, 32.0, -16.0]
[3, 8, 4, 2] : [0.0, 0.0, 5.33, 0.0, 5.33, 0.0, 0.0, 0.0, 5.33, 0.0, 0.0] : [5.33, 21.33, 5.33]
[3, 8, 9, 10] : [0.0, 0.0, 0.0, 16.0, 0.0, 0.0, 0.0, 0.0, 0.0, 16.0, 32.0] : [48.0, 32.0, -16.0]

[4, 8, 9, 2] : [0.0, 0.0, 5.33, 0.0, 5.33, 0.0, 0.0, 0.0, 5.33, 0.0, 0.0] : [5.33, 21.33, 5.33]
[4, 8, 9, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[8, 3, 4, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
[9, 2, 4, 10] : [0.0, 0.0, 0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.0] : [16.0, 24.0, 0.0]
Tiempo de ejecución: 143 milisegundos

Se resuelve por último el problema 5 (de máximo, con restricciones de tipo "$\leq$"), con datos:

$$A = \begin{pmatrix} 1 & 3 & -4 & 1 & -1 & 1 & 1 & 1 \\ 5 & 2 & 4 & -1 & 3 & 7 & 2 & 7 \\ 0 & 4 & -1 & -1 & -3 & 0 & 0 & 1 \\ -3 & -4 & 8 & 2 & 3 & -4 & 5 & -1 \\ 12 & 8 & -1 & 4 & 0 & 1 & 1 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 8 & -12 & -3 & 4 & -1 & 0 & 0 & 0 \\ -5 & -6 & 12 & 1 & 0 & 0 & -1 & 1 \end{pmatrix},$$

$$b^T = \left(40, 84, 18, 100, 40, -12, 30, 100\right),$$

$$C = \begin{pmatrix} 3 & -7 & 4 & 1 & 0 & -1 & -1 & 8 \\ 2 & 5 & 1 & -1 & 6 & 8 & 3 & -2 \\ 5 & -2 & 5 & 0 & 6 & 7 & 2 & 6 \\ 0 & 4 & -1 & -1 & -3 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

y tras cambiar de signo la restricción número 6 para partir de $b \geq 0$. Se obtienen 51 bases eficientes, mostramos las 20 primeras.

Bases eficientes, soluciones respectivas y valor de la función objetivo:
[8, 4, 10, 3, 0, 2, 14, 13] :
[3.52, 0.0, 9.64, 1.86, 9.90, 0.0, 0.0, 0.0, 83.10, 0.0, 59.19, 0.0, 0.0, 12.91, 33.26, 0.0] :
[50.99, 74.20, 125.19, -41.19, 24.91]

[8, 4, 10, 3, 1, 2, 14, 13] :
[0.0, 5.50, 10.94, 1.74, 10.34, 0.0, 0.0, 0.0, 75.85, 0.0, 39.71, 0.0, 0.0, 16.51, 132.12, 0.0] :
[7.02, 98.69, 105.71, -21.71, 28.51]

[8, 5, 10, 4, 3, 2, 14, 13] :
[0.0, 0.0, 7.43, 10.81, 11.90, 4.20, 0.0, 0.0, 66.63, 0.0, 71.95, 0.0, 0.0, 22.34, 20.96, 0.0] :
[36.34, 101.60, 137.95, -53.95, 34.34]

[8, 7, 10, 4, 3, 2, 14, 13] :
[0.0, 0.0, 6.87, 11.72, 9.14, 0.0, 0.0, 5.83, 59.08, 0.0, 58.19, 0.0, 0.0, 21.56, 12.89, 0.0] :
[85.84, 38.35, 124.19, -40.19, 33.56]

[8, 7, 10, 6, 3, 2, 14, 13] :
[0.0, 0.0, 7.3, 10.40, 0.0, 0.0, 5.70, 7.69, 45.42, 0.0, 28.02, 0.0, 0.0, 19.08, 10.30, 0.0] :
[95.40, -1.39, 94.02, -10.02, 31.08]

[8, 7, 10, 11, 3, 2, 14, 13] :
[0.0, 0.0, 6.54, 11.63, 0.0, 0.0, 0.0, 9.93, 44.59, 0.0, 26.24, 34.37, 0.0, 16.10, 3.07, 0.0] :
[117.20, -24.96, 92.24, -8.24, 28.10]

[8, 0, 10, 4, 3, 15, 14, 13] :

[0.82, 0.0, 0.0, 7.53, 29.14, 0.0, 0.0, 0.0, 60.78, 0.0, 112.94, 0.0, 0.0, 25.49, 22.43, 96.59] :
[10.0, 168.94, 178.94, -94.94, 37.49]

[8, 1, 10, 4, 3, 15, 14, 13] :
[0.0, 1.17, 0.0, 7.67, 29.78, 0.0, 0.0, 0.0, 58.61, 0.0, 110.33, 0.0, 0.0, 26.61, 43.11, 99.33] :
[-0.5, 176.83, 176.33, -92.33, 38.61]

[8, 5, 10, 4, 3, 15, 14, 13] :
[0.0, 0.0, 0.0, 9.70, 28.45, 1.19, 0.0, 0.0, 57.56, 0.0, 113.06, 0.0, 0.0, 27.35, 19.65, 90.30] :
[8.51, 170.55, 179.06, -95.06, 39.35]

[8, 7, 10, 4, 3, 15, 14, 13] :
[0.0, 0.0, 0.0, 10.0, 27.25, 0.0, 0.0, 1.75, 55.5, 0.0, 108.0, 0.0, 0.0, 27.0, 17.25, 88.25] :
[24.0, 150.0, 174.0, -90.0, 39.0]
...
Tiempo de ejecución: 251 milisegundos