Universidad de Valladolid

FACULTA DE CIENCIAS

GRADO EN ESTADISTICA

El problema bi-objetivo de cubrimiento máximo y cubrimiento reforzado

Alumno: Tomás Carretero Alarcón

Tutores: Jesús M. Sáez-Aguado y Jesús Alberto Tapia

Dedicatorias

A mi familia y amigos, por su apoyo y compresión incondicional en este camino.

Agradecimientos

Me gustaría comenzar agradeciendo a mis padres, mi hermano y mi abuela por su apoyo incondicional durante estos años de carrera universitaria, sin su apoyo y compresión, este camino no hubiera sido tan llevadero.

También me quiero acordar de mis amigos, que han sabido comprenderme y apoyarme en los momentos en los que yo necesitaba abstraerme de la vida social y centrarme en mis estudios. Pero sin embargo, me gustaría agradecerles especialmente, los momentos de diversión y locura que hemos compartido, sobre todo cuando yo necesitaba descononectar de la rutina universitaria. Así como también quiero hacer una mención especial para todos mis compañeros de carrera, que han hecho que la vida universitaria haya sido mucho más amena y divertida.

Por otro lado, no puedo dejar de acordarme de una persona muy especial para mi, que estuvo a mi lado, trabajando duro y dándome todo su apoyo, amor y comprensión durante mis inicios en el mundo academécio, me refiero a mi antigua vecina, amiga y para mi "tia postiza", Soledad, sin la que no se si hubiera podido llegar hasta aquí.

Por último, quiero agradecer, en especial a mis tutores: Jesús Sáez y Jesús Alberto Tapia, por su apoyo y ayuda durante la realización de este trabajo. Pero también sería descortés no acordarme de todos los profesores de los que he tenido el privivelegio de aprender durante estos años, y que han hecho que mi formación académica haya sido tan enriquecedora.

Resumen

La optimización es importante para la toma de decisiones en la vida diaria, pero en el mundo empresarial es un pilar fundamental a la hora de mejorar la eficiencia de los procesos, reducir el uso de recursos y aumentar la rentabilidad. Todo esto se puede traducir en una mejora competitiva de la empresa en el mercado, ya que no aplicar técnicas de optimización puede suponer quedarse por detrás de la competencia. Sin embargo, la mayoría de los problemas de optimización no son unidimensionales, sino que tienen múltiples objetivos que pueden entrar en conflicto entre sí, lo que supone que, al mejorar uno de los objetivos, se empeore otro.

El objetivo de estre Trabajo de Fin de Grado (TFG) es estudiar y compender los problemas de optimización multi-objetivo. Se ha realizado un estudio exhaustivo de todo el problema de optimización multi-objetivo, desde definir los conceptos básicos de este, tales como los espacios objetivo y decisión, hasta explicar conceptos más avanzados como las soluciones débilmente y estrictamente eficientes, la eficiencia prorpia de las soluciones o la dominancia propia de unas soluciones sobre otras.

No sólo se han estudiado las bases teóricas del problema multi-objetivo, sino que también se ha investigado acerca de los principales métodos para resolver estos problemas, analizando sus fundamentos matemáticos e implementándolos en XPRESS. Así mismo, se han puesto a prueba estas implementaciones, para resolver un problema real bi-objetivo de cubrimiento máximo y cubrirmiento máximo reforzado. Con el objetivo de comparar ambos métodos de obtención de la frontera de Pareto, y poder establecer diferencias, puntos fuertes y puntos más débiles de cada uno, se han ejecutado para varios conjuntos de datos.

La memoria del TFG se ha elaborado con LaTeX, pero esta no es la única herramienta utilizada, ya que también se ha utilizado XPRESS para implementar los métodos de optimización multi-objetivo, y Python para procesar los conjuntos de datos así como para generar los gráficos que se han incluido en esta memoria.

Abstract

Optimization is essential for decision-making in everyday life, but in the business world it serves as a fundamental pillar for improving process efficiency, reducing resource usage, and increasing profitability. All of this can be translate into a competitive advantage in business market, and not applying optimization techniques may result into falling behind the competition. However, most optimization problems are not one-objetive; rather, they involve multiple objectives that can conflict with each other, so improving one may lead to the deterioration of another.

The objective of this FYP is to study and understand multi-objective optimization problems. An exhaustive study of the entire multi-objective optimization problem has been carried out, from defining its basic concepts, such as the objective and decision spaces, to explaining more advanced concepts like weakly and strictly efficient solutions or self-efficiency and non-dominance.

As could be expected, not only the bases of the problem have been studied, rather a research has also been studied on the main methods for solving multi-objective problems, analyzing their mathematical foundations and implementing them in XPRESS, an optimization solver. In this way, two of these methods have been tested and compared.

The final proyect report has been prepared using LaTeX; however, this is not the only tool utilized, as XPRESS was also employed to implement the multi-objective optimization methods, and Python was used to process the original datasets and generate the graphs included in the thesis.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Al hablar de Programación Lineal (PL) es habitual pensar en un problema de optimización de recursos en el que queremos maximizar o minimizar una función objetivo, definida por unas variables de decisión, y verificando un conjunto de restricciones.

La Programacion Lineal Entera (PLE) es una extensión de la programación lineal en la que las variables de decisión son enteras, y por tanto, el conjunto de soluciones factibles es discreto. Cuando tenemos un problema de PLE es común resolverlo optimizando una úncia función objetivo, y aunque esto es un problema NP-complejo, existen algoritmos que pueden obtener soluciones optimas en un tiempo razonable. ¿Pero qué hacemos si nuestro problema está definido con varias funciones objetivo enfrentadas?

1.2. Motivación

Aunque el campo de los problemas de optimización uni-objetivo está ampliamente estudiado, el estudio los problemas de optimización multi-objetivo (POM), al menos en ámbito educativo, no está tan extendido. La motivación de este trabajo pasa por adquirir conocimientos en el campo de los POM, analizando los principales métodos que permiten resolver estos problemas, implementarlos y ponerlos a prueba con casos reales.

1.3. Objetivos

Los principales objetivos de este trabajo son:

- Aprender los conceptos básicos de los POM.
- Estudiar los conceptos teóricos de los principales métodos que permiten obtener la frontera de Pareto de un POM.
- Implementar algunos de los métodos anteriores para un caso bi-objetivo de Programacion Lineal Entera (PLE).
- Realizar un estudio comparativo de los resultados obtenidos con ambos métodos para conjuntos de datos reales.

1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 2 Principales definiciones: Describe los fundamentos de los POM.

Capítulo 3 Principales técnicas para la obtención de la frontera de Pareto: Se describen los principales métodos para obtener la frontera de Pareto de un POM, entrando en detalles con el método de sumas ponderas y el método ε-constraint.

Capítulo 4 Cubrimiento máximo: Se describen tanto el problema de cubrimiento máximo o parcial (MCLP) como el problema de cubrimiento máximo reforzado (MCLPR), así como también se introduce el modelo de cubrimiento máximo bi-objetivo (BACOP2).

Capítulo 5 Formulación y solución del problema BACOP2 en un caso real: Se describe como sería la codificacion de los métodos vistos en el capítulo anterior, así como se da un enlace para ver mi codificación de estos métodos.

Capítulo 6 Puesta a prueba de las soluciones al problema BACOP2 con varios conjuntos de datos: Se presentan los resultados obtenidos con ambos métodos para tres conjuntos de prueba distintos, así como una comparación entre ellos.

Capitulo 7 Conclusiones Se presentan las conclusiones obtenidas a partir de los resultados obtenidos en el capítulo anterior, así como una comparativa entre ambos métodos, una opinión personal y una propuesta de futuras líneas de trabajo.

Capítulo 2

Planteamiento de los problemas POM

En lugar de un único objetivo, en la optimización multi-objetivo se tienen dos o más objetivos que se deben optimizar simultáneamente. El problema es que no existe un único punto óptimo, sino un conjunto de soluciones, ya que en función de la importancia que le demos a cada función objetivo, se obtendrá un conjunto de soluciones diferentes. La primera referencia a este tipo de situación se atribuye a Wilfredo Pareto, [8]. Pareto introdujo el concepto de eficiencia económica, y propuso que una asignación de recursos es eficiente si no es posible mejorar la situación de un individuo sin empeorar la de otro. Este concepto se conoce como la eficiencia de Pareto, y es la base de la optimización multi-objetivo.

2.1. Notación

Al igual que en otras áreas de la ciencia, en los POM es habitual que cada autor utilice una notación diferente. Es por esto, que considero fundamental definir la notación que se va a emplear en este trabajo. Supongamos el problema multi-objetivo siguiente:

$$\text{Minimizar} \quad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})) \tag{2.1}$$

$$\text{sujeto a} \quad \mathbf{x} \in \mathcal{X} \tag{2.2}$$

donde:

- $\mathcal{X}$: conjunto factible de soluciones.
- $\mathcal{Y} := f(\mathcal{X})$: conjunto factible en el espacio objetivo.
- $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathcal{X}$: solución factible del problema de optimización.
- $\mathbf{f} = (f_1, f_2, ..., f_m)$ vector de funciones objetivo.
- $f_i(\mathbf{x})$: valor de la función objetivo i en la solución $\mathbf{x}$.
- $\mathbf{y} := \mathbf{f(x)} = (f_1(\mathbf{x}), , ..., f_n(\mathbf{x})) \in \mathcal{Y}$: solución factible en el espacio objetivo.
- y^I : punto ideal del espacio objetivo
- y^N : punto nadir del espacio objetivo

2.1.1. Formulación multiobjetivo en programación lineal

En el caso de la programación lineal, la formulación del problema general multi-objetivo 2.1 se traduce a la siguiente formulación:

$$\text{Minimizar} \quad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})) \tag{2.3}$$

$$\text{sujeto a} \quad A\mathbf{x} \leq b, \quad \mathbf{x} \geq \mathcal{X} \tag{2.4}$$

donde:

- $\mathbf{A} \in \mathbb{R}^{p \times n}$: es la matriz de coeficientes de las restricciones.
- $\mathbf{b} \in \mathbb{R}^{p}$: vector de términos independientes de las restricciones.
- $\mathcal{X} \in \mathbb{R}^{n}$: conjunto de soluciones factibles.

2.1.2. Formulación multiobjetivo en programación no lineal

Mientras que en el caso de la programación no lineal, la formulación del problema general multi-objetivo 2.1 se traduce a la siguiente formulación:

$$\begin{aligned} \text{Minimizar} \quad & \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})) && (2.5)\\ \text{sujeto a} \quad & \mathbf{g_j}(\mathbf{x}) \leq 0, j = 1, 2, ..., r && (2.6)\\ & \mathbf{h_l}(\mathbf{x}) = 0, l = 1, 2, ..., s && (2.7)\\ & \mathbf{x} \geq 0 && (2.8) \end{aligned}$$

donde:

- $\mathbf{g_j}(\mathbf{x}) \leq 0, j = 1, 2, ..., r$: son las restricciones de desigualdad.
- $\mathbf{h_l}(\mathbf{x}) = 0, l = 1, 2, ..., s$: son restricciones de igualdad.

2.1.3. Formulación multiobjetivo en programación entera

Finalmente, en el caso de la programación entera, la formulación del problema general multi-objetivo 2.1 se traduce a la siguiente formulación:

$$\begin{aligned} \text{Minimizar} \quad & \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})) && (2.9)\\ \text{sujeto a} \quad & A\mathbf{x} \leq b, \quad \mathbf{x} \geq \mathcal{X}, \quad \mathbf{x} \in \mathbb{Z}^n && (2.10) \end{aligned}$$

donde:

- $\mathbf{A} \in \mathbb{R}^{p \times n}$: es la matriz de coeficientes de las restricciones.
- $\mathbf{b} \in \mathbb{R}^{p}$: vector de términos independientes de las restricciones.
- $\mathcal{X} \in \mathbb{R}^{n}$: conjunto de soluciones factibles.

En las secciones y capítulos posteriores, explicaré con más detalle el significado de la mayoría de estos conceptos, introduciré los principales métodos para resolver los problemas multi-objetivo, e implementaré al menos un par de estos métodos, los cuales luego serán puestos a prueba con conjuntos de datos reales.

2.2. Solucion factible y conjunto factible

Una solución factible es aquella que cumple con todas las restricciones del problema. Por tanto $\mathbf{x} = (x_1, ..., x_n)$ será una solución factible si cumple tanto las restricciones explícitas, restricciones de igualdad y desigualdas impuestas sobre las variables de decisión, como las restricciones de dominio de las variables de decisión.

Si $\mathbf{x}$ es factible entonces $\mathbf{x} \in \mathcal{X}$, ya que $\mathcal{X}$ representa el conjunto de todas las soluciones factibles del problema de optimización.

2.3. Espacios objetivo y decision

El **espacio de decisión** es un conjunto n-dimensional, donde n es el número de variables de decisión. Se representa por $\mathbb{X}$ y está constituido por todas las soluciones posibles del problema, tanto factibles como no factibles. Por tanto $x \in \mathcal{X}$ si cumple las restricciones de dominio de las variables de decisión, pero puede o no cumplir las restricciones explícitas de igualdad y desigualdad.

Esto deja claro que $\mathcal{X} \subseteq \mathbb{X}$

El **espacio objetivo** es un conjunto m-dimensional, donde m es el número de funciones objetivo, formado por las evualuaciones en cada una de las posibles soluciones del espacio de decisión.

Esto es, el conjunto $(\mathbf{f(x)}|x \in \mathbb{X})$, donde $\mathbf{f(x)} = (f_1(x), f_2(x), ..., f_m(x))$.

2.4. Soluciones eficientes y dominancia

Para las siguientes definiciones, considerara estar ante un problema de minimización para todas las funciones objetivo. El caso de maximización o mezcla sería igual pero cambiando los signos de las desigualdades según corresponda.

Una solución $\hat{\mathbf{x}}$ es eficiente u óptimo de Pareto si no existe otra solución $\mathbf{x} \in \mathcal{X}$ tal que $\mathbf{f_i(x)} <= \mathbf{f_i}(\hat{\mathbf{x}})$ para todo $i \in \{1, ..., m\}$ y además $f_j(x) < f_j(\hat{x})$ para al menos un $j \in \{1, ..., m\}$.

Informalmente, podremos expresar que $\hat{\mathbf{x}}$ es eficiente si no es posible mejorar un objetivo sin empeorar otro.

Si $\hat{\mathbf{x}}$ es efiente entonces, entonces $\mathbf{f}(\hat{\mathbf{x}})$ es un punto no dominado.

Si $x^1, x^2 \in \mathcal{X}$, $f_i(x^1) <= f_i(x^2)$ para todo $i \in \{1, ..., m\}$, y además existe al menos un objetivo, j, tal que $f_j(x^1) < f_j(x^2)$, entonces diremos que $\mathbf{x^1}$ domina a $\mathbf{x^2}$ y de igual modo también podemos decir que $\mathbf{f}(\mathbf{x^1})$ domina a $\mathbf{f}(\mathbf{x^2})$.

El conjutno de todas las soluciones eficientes $\mathbf{x} \in \mathcal{X}$ se denota por $\mathcal{X}_E$ y se conoce como conjunto eficiente o de Pareto.

El conjunto de todos los puntos no dominados $\mathbf{f(x)} \in \mathcal{Y}$ se denota por $\mathcal{Y}_N$ y se conoce como conjunto no dominado.

2.4.1. Algunas propiedades del conjunto de soluciones no dominadas ($\mathcal{Y}_N$)

- De acuerdo a las definiciones anteriores, $\hat{\mathbf{y}} \in \mathcal{Y}$ será no dominado si no existe otro $\mathbf{y} \in \mathcal{Y}$ tal que $\mathbf{y} \leq \hat{\mathbf{y}}$.
- Para problemas de minimización, los puntos no dominados se localizan en la parte inferior izquierda de $\mathcal{Y}$. De igual modo que para problemas de maximización se localizan en la parte superior derecha de $\mathcal{Y}$.
- $\mathcal{Y}_N \subset bd(\mathcal{Y})$, esto es, los puntos eficientes deben pertececer a la frontera de $\mathcal{Y}$. Prueba: Proposición 2.4. del [4]

En 2.1 se puede ver los puntos ideal y madir así como los míminos de cada funcion objetivo

2.5. Limites del conjunto de soluciones no dominadas

Definimos los puntos ideal y nadir como límites inferior y superior respectivamente del conjunto de puntos no dominados. Estos puntos nos dan una idea del rango de valores que pueden tomar los puntos

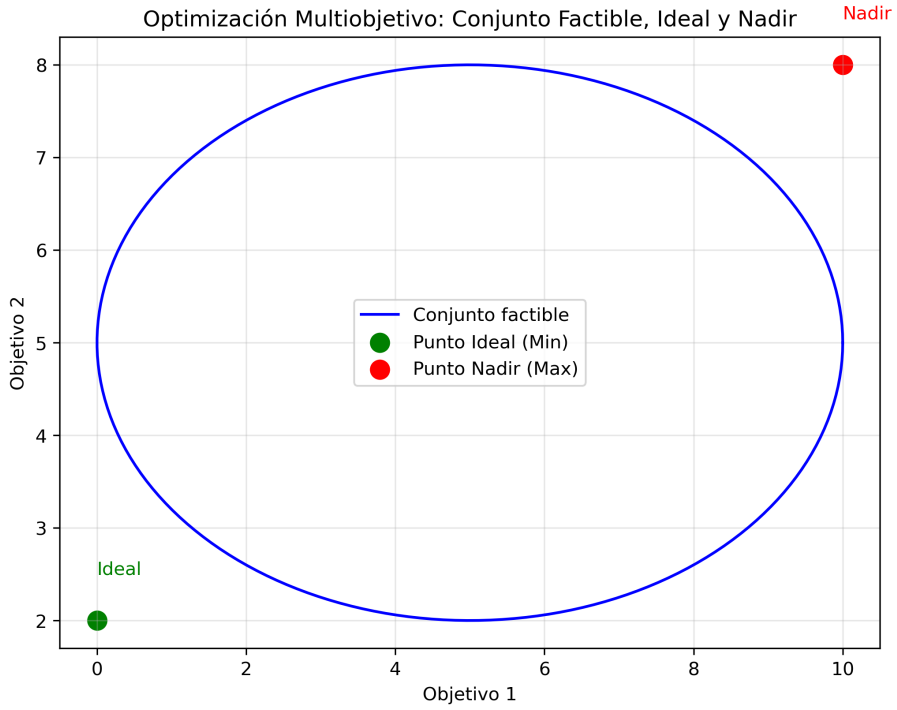

Figura 2.1: Conjutno factible y puntos ideal y Nadir para una función biobjetivo doble minimizacion.

no dominados. A menudo, son utilizados como como puntos referencia en la programación de compromiso pero también en métodos interactivos en los que el objetivo es encontrar la solución que mejor se ajuste a las preferencias del decisor.

Asumiendo: un problema de minimización y que los conjuntos $\mathcal{X}_E$ y $\mathcal{Y}_E$ no son vacios, entonces:

- El punto $y^I = (y_1^I, ..., y_p^I)$ dado por

$$y_k^I := min_{x \in \mathcal{X}}\{f_k(x)\} = min_{y \in \mathcal{Y}_E}\{y_k\}$$

  se conoce como **punto ideal del MOP**.

- El punto nadir $y_k^N = (y_1^N, ..., y_p^N)$ dado por

$$y_k^N := max_{x \in \mathcal{X}_E}\{f_k(x)\} = max_{y \in \mathcal{Y}_N}\{y_k\}$$

  se conoce como **punto nadir del MOP**.

Es evidente decir que $y_k^I \leq y_k$ así como que $y_k \leq y_k^N$ para todo $y_k \in \mathcal{Y}_N$. Es más, y^I y y^N son ampliamente los limites infererior y superior del conjunto de eficiencia. Ya que el punto ideal se obtinene resolviendo p problemas de optimización uniobjetivo, por lo que su cómputo puede considerarse sencillo (desde el punto de vista POM). Pero por otro lado, el cómputo de y^N implica ompitimizar sobre el conjunto eficiencia, un problema mucho más complejo. De hecho, no se conoce un método para determinar y^N de forma exacta, para un POM general.

Dada la complejidad del cómputo de y^N, es frecuente recurrir a heurísticas para su aproximación. Una estimación básica del punto nadir utiliza tablas de resultados de la optimización de problemas de optimización monobjetivo:

1 Primero, resolvemos p problemas de optimización uniobjetivo, $min_{\mathbf{x} \in \mathcal{X}} f_k(\mathbf{x})$.

  1.1 Denominamos a estas soluciones optimas, $\mathbf{x}_k^I$, $k = 1, \ldots, p$, esto es, $f_k(\mathbf{x}^k) = min_{\mathbf{x} \in \mathcal{X}} f_k(\mathbf{x})$.

2 Utilizando estas soluciones optimas, calcularemos una tabla de resultados como la tabla Como se muestra en la Tabla 2.1:

	x^1	x^2	...	x^{p-1}	x^p
f_1	y_1^I	$f_1(\mathbf{x}^2)$	...	$f_1(\mathbf{x}^{p-1})$	$f_1(\mathbf{x}^p)$
f_2	$f_2(\mathbf{x}^1)$	$\ddots$	...	...	$f_2(\mathbf{x}^p)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
f_{p-1}	$f_{p-1}(\mathbf{x}^1)$	...	...	$\ddots$	$f_{p-1}(\mathbf{x}^p)$
f_p	$f_p(\mathbf{x}^1)$	$f_p(\mathbf{x}^2)$	...	$f_p(\mathbf{x}^{p-1})$	y_p^I

Tabla 2.1: Tabla de resultados del algoritmo trade-off.

3 Finalmente, en la Tabla 2.1 vemos claremanete que $y_k^I = f_k(x^k), k = 1, ..., p$. Por otra parte también podemos extraer que

$$\tilde{y}_i^N := \max_{k=1,...,p} f_i(\mathbf{x}^k)$$

por tanto el elemento más grande de la fila i de la tabla como una estimción de y_i^N.

Aunque parezca atractivo a primera vista, el problema de las tablas de resultados viene dado porque $\tilde{y}_i^N$ puede sobre/infra estimacionar a y_i^N, cuando tengamos más de dos objetivos así como también en caso de que los problemas de optimización uniobjetivo presenten multiples soluciones optimas.

El único caso donde y^N puede calcularse de forma exacta es para $p = 2$. En este caso, el peor valor que puede tomar y_2 está ligado a que y_1 sea mínimo, y viceversa, por lo que mediante proceso de optimización bi paso, podemos eliminar las elecciones débilmene eficientes de la tabla de resultados. Desafortunadamente, este proceso no es aplicable para $p > 2$, ya que en el segundo paso, no sabríamos que objetivo ajustar.

2.6. Soluciones eficientes debíles y estríctas

Los puntos no dominados están definidos por el orden componente a componente en $\mathbb{R}^p$. Pero si, en su lugar, utilizamos el orden componenete a componente débil y estricto, obtenemos los conjuntos de soluciones eficientes débiles y estrictas, respectivamente.

- Una solución factible $\hat{\mathbf{x}} \in \mathcal{X}$ es **eficiente débil (óptimo de Pareto débil)** si no existe otra solución factible $\mathbf{x} \in \mathcal{X}$ tal que $\mathbf{f}(\mathbf{x}) < \mathbf{f}(\hat{\mathbf{x}})$, es decir, $f_k(\mathbf{x}) < f_k(\hat{\mathbf{x}}) \forall k = 1, ..., p$. El punto $\hat{\mathbf{y}} = \mathbf{f}(\hat{\mathbf{x}})$ es un punto no dominado débil.
- Una solución factible $\hat{\mathbf{x}} \in \mathcal{X}$ es **eficiente estricta (óptimo de Pareto estricto)** si no existe otra solución factible $\mathbf{x} \in \mathcal{X}, \mathbf{x} \neq \hat{\mathbf{x}}$ tal que $\mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\hat{\mathbf{x}})$, es decir, $f_k(\mathbf{x}) \leq f_k(\hat{\mathbf{x}}) \forall k = 1, ..., p$.
- Los conjuntos débil (estricto) eficientes y no dominados se denotan por $\mathcal{X}_{wE}$ ($\mathcal{X}_{sE}$) y $\mathcal{Y}_{wE}$, respectivamente.

Entendidas estas definiciones, resulta obvio resaltar que:

$$\mathcal{X}_{sE} \subset \mathcal{X}_E \subset \mathcal{X}_{wE}$$

así como que

$$\mathcal{Y}_N \subset \mathcal{Y}_{wN}$$

En 1997, Matthias Ehrgott publica un paper en el que demuestra que: Sea $\hat{\mathbf{x}} \in \mathcal{X}$ una solución factible, y si definimos $\hat{y}_k := f_k(\hat{x}), k = 1, ..., p$. Entonces:

- $\hat{\mathbf{x}}$ es estrictamente eficiente si y sólo sí no existe otra solución factible $\mathbf{x} \in \mathcal{X}, x \neq \hat{x}$ tal que $f_k(x) \leq f_k(\hat{x})$ para todo $k = 1, ..., p$.
- $\hat{\mathbf{x}}$ es eficiente si y sólo sí no existe otra solución factible $\mathbf{x} \in \mathcal{X}$ tal que se cumpla tanto que $f_k(x) < f_k(\hat{x})$ para todo $k = 1, ..., p$, como que $f_j(x) < f_j(\hat{x})$ para algún j.
- $\hat{\mathbf{x}}$ es débilmente eficiente si y sólo sí no existe otra solución factible $\mathbf{x} \in \mathcal{X}$ tal que $f_k(x) < f_k(\hat{x})$ para todo $k = 1, ..., p$.

2.7. Eficiencia propia y no dominancia propia

De acuerdo a la definición dada de solución eficiente, esta no permite la mejora de una función objetivo al mismo tiempo que el resto de funciones objetivo conserven sus valores. Por lo que la mejora de un objetivo sólo puede hacerse a costa de al menos otro de los objetivos. Estas compensaciones entre objetivos pueden medirse mediante el cálculo del incremento de un objetivo f_i por unidad de decremento de otro objetivo f_j. En algunas ocasiones, esta compensación entre objetivos puede ser ilimitada.

En 1968, Alfred W. Greoffrion [5] define las soluciones eficientes con compensaciones limitadas, lo que conocemos como soluciones eficientes propias:

- Una solución factible $\hat{\mathbf{x}} \in \mathcal{X}$ es **eficiente propia**, si esta es eficiente y existe un número real $M > 0$ tal que para todo i y $x \in \mathcal{X}$, que satisfaga que $f_i(\mathbf{x}) \leq f_i(\hat{\mathbf{x}})$, entonces existe un índice llamémoslo j tal que $f_j(\hat{\mathbf{x}}) < f_j(x)$ de modo que

$$\frac{f_i(\hat{\mathbf{x}}) - f_i(\mathbf{x})}{f_j(\mathbf{x}) - f_j(\hat{\mathbf{x}})} \leq M$$

  .

- El correspondiente punto $\hat{\mathbf{y}} = \mathbf{f}(\hat{\mathbf{x}})$ se conoce como solución **no dominada propia**.

Los resultados principales de Greoffrion acerca de la soluciones de eficiencia propia muestran que pueden obtenerse minimizando una suma de pesos de las funiones objetivo donde todos los pesos sean positivos.

Otras definiciones de eficiencia propia han sido propuestas, como la de Borwei, Benson, y Khun y Tucker.

Capítulo 3

Obtención de la frontera de Pareto

En este capítulo se van a mostrar las principales técnicas para encontrar la frontera de Pareto o el conjunto de puntos no dominados y eficientes, que cumplan con las restricciones del problema de optimización multi-objetivo en cuestión. En esta sección se presentan al menos dos métodos que permiten aproximar la frontera de Pareto. El primero es el método de la sumas ponderadas y el segundo el algoritmo de ε-constraint.

3.1. Método de sumas ponderadas

El método de sumas ponderadas trata de resolver un POM (encontrar sus soluciones eficientes) mediante la resolución de problemas de optimización uniobjetivo, donde cada uno de los objetivos es ponderado con un valor λ_i. Estos problema de optimización uniobjetivo (o escalares) se conocen como **problemas de escalarización por sumas ponderadas** del POM original.

La formulación del problema 2.1 por el método de sumas ponderadas sería la siguiente:

$$\min_{\mathbf{x}\in\mathcal{X}} \quad \mathbf{f(x)} = \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}) \tag{3.1}$$

sujeto a:

$$\mathbf{g}(\mathbf{x}) \leq 0 \tag{3.2}$$

$$\lambda_i \in [0,1], \quad \text{para } i = 1, 2, ..., m \tag{3.3}$$

$$\sum_{i=1}^{m} \lambda_i = 1 \tag{3.4}$$

Primero tendremos en cuenta el espacio objetivo $\mathcal{Y}$, en el que se provarán algunos resultados acerca de la relación entre los puntos no dominados (débiles y propios) y los valores de $\sum_{k=1}^{p} \lambda_k \cdot y_k$. De estos, se pueden derivar otros resultados acerca de la conexión entre $\mathcal{X}_{(w,p)E}$ y y las soluciones óptimas del problema de escalarización por sumas ponderadas. Además también serán útiles para demostrar las condiciónes de óptimalidad Fritz-John y Karush-Kuhn-Tucker para problemas con soluciones (débiles, propias) eficientes. Y finalmente, investigar las condiciónes que garantizan la conexión entre los conjunstos de eficiencia y no dominancia.

3.1.1. Espacio objetivo, puntos no dominados y sumas ponderadas

Suponiendo que $\mathcal{Y} \subset \mathbb{R}^p$, para un valor ajustado de $\lambda \in \mathbb{R}^p$, denominamos por

$$\mathcal{S}(\lambda, \mathcal{Y}) := \{\hat{y} \in \mathcal{Y} : \langle \lambda, \hat{\mathbf{y}} \rangle = \min_{\mathbf{y} \in \mathcal{Y}} \langle \lambda, \hat{\mathbf{y}} \rangle\}$$

al conjunto de puntos óptimos de $\mathcal{Y}$ con respecto a λ. Debido a la defínción de puntos no dominados tenemos que considerar sólo vectores ponderados no negativos, $\lambda \in \mathbb{R}^p_{\geq 0}$. Además, debemos asumir que que $\sum_{k=1}^p \lambda_k = 1$, la cual simplemente normaliza los pesos, pero no cambia $\mathcal{S}(\lambda, \mathcal{Y})$. Utilizaremos la siguientes notaciones por comodidad:

$$\mathcal{S}(\mathcal{Y}) := \bigcup_{\lambda \in \mathbb{R}^p_>} \mathcal{S}(\lambda, \mathcal{Y}) = \bigcup_{\{\lambda > 0 : \sum_{k=1}^p \lambda_k = 1\}} \mathcal{S}(\lambda, \mathcal{Y})$$

$$\mathcal{S}_0(\mathcal{Y}) := \bigcup_{\lambda \in \mathbb{R}^p_\geq} \mathcal{S}(\lambda, \mathcal{Y}) = \bigcup_{\{\lambda \geq 0 : \sum_{k=1}^p \lambda_k = 1\}} \mathcal{S}(\lambda, \mathcal{Y})$$

$$\Lambda := \{\lambda \in \mathbb{R}^p_> \ : \sum_{k=1}^p \lambda_k = 1\}$$

$$\Lambda^0 := ri\Lambda = \{\lambda \in \mathbb{R}^p_\geq : \sum_{k=1}^p \lambda_k = 1\}$$

En muchos de los resultados de las siguientes secciones, necesitaremos asunciones de convexidad. Sin embargo, requerir que $\mathcal{Y}$ sea convexo suele ser un requisito demasiado restrictivo. Después de todo, estamos buscando puntos no dominados, que, teniendo en cuenta la función de problema, se encuentran en uno de los bordes de $\mathcal{Y}$. ([4] páginas 67 y 68).

Para aquellos resultados válidos para cualquier conjunto de $\mathcal{Y}$, obtenemos los resultados análogos simplemente haciendo referencia al hecho de que las soluciones eficientes son preimágenes de los puntos no dominados. Para aquellos resultados que sólo son válidos bajo ciertas condiciónes de $\mathcal{Y}$, deben hacerse las asunciones pertinentes sobre $\mathcal{Y}$ y $\mathbf{f}$. Para asegurar la convexidad de $\mathcal{Y}$ debe hacerse la asunción de convexidad tanto de $\mathcal{X}$ como de todas las funciones objetivo f_k.

3.1.2. Escalarización de sumas ponderadas y eficiencia (débil)

Las soluciones óptimas del problema de sumas ponderadas con pesos positivos son siempre (débilmente) eficientes y bajo las asunciones de convexidad todas las soluciones (débilmente) eficientes son soluciones óptimas de problemas de escalarización con pesos positivos.

Teorema 3.1. Para todo conjunto $\mathcal{Y} \subset \mathbb{R}^p$ tenemos que $\mathcal{S} \subset \mathcal{Y}_{wN}$.

Prueba. Si $\lambda \in \mathbb{R}^p_\geq$ y $\hat{y} \in \mathcal{Y}(\lambda, \mathcal{Y})$.

$$\sum_{k=1}^p \lambda_k \cdot \hat{y}_k \geq \sum_{k=1}^p \lambda_k \cdot y_k \ para\ todo\ y \in \mathcal{Y}$$

Suponiendo que $\hat{y} \notin \mathcal{Y}_{wN}$, entonces existe algún $y' \in \mathcal{Y}$ con $y'_k < \hat{y_k},\ k = 1, \ldots, p$. Y por tanto,

$$\sum_{k=1}^p \lambda_k \cdot y'_k < \sum_{k=1}^p \lambda_k \cdot \hat{y}_k \ para\ todo\ y \in \mathcal{Y}$$

debido a que al menos uno de los pesos, λ_k debe ser positivo. Esta contradicción implica el resultado del teorema 3.1.

Teorema 3.2. Si $\mathcal{Y} \in \mathbb{R}^p$. Entonces, $\mathcal{S}(\mathcal{Y}) \subset \mathcal{Y}_N$.

Prueba. Suponiendo $\hat{y} \notin \mathcal{S}(\mathcal{Y})$. Entonces existe algún $\lambda \in \mathbb{R}^p_>$ que satisface que $\sum_{k=1}^p \lambda_k \cdot \hat{y}_k \geq \sum_{k=1}^p \lambda_k \cdot y_k$ *para todo* $y \in \mathcal{Y}$

Supongamos que $\hat{y} \notin \mathcal{Y}_N$. Por consiguiente, debe existir un $y' \in \mathcal{Y}$ con $y' \leq y$. El producto componente por los pesos nos da $\lambda_k \cdot \leq \lambda_k \cdot \hat{y}_k$ *para todo* $k = 1, \ldots, p$ y una desigualdad estricta para un k. Es desigualdad estricta junto con el hecho de que todos los pesos son positivos implican que $\sum_{k=1}^p \lambda_k \cdot y'_k < \sum_{k=1}^p \lambda_k \cdot \hat{y}_k$, contradiciendo que $\hat{y} \in \mathcal{S}(\mathcal{Y})$.

Proposicion 3.3. Si $\hat{y}$ es el único elemento de $\mathcal{S}(\lambda, \mathcal{Y})$ para algún $\lambda \in \mathbb{R}^p_\geq$ entonces $\hat{y} \in \mathcal{Y}_N$.

Resumamos ahora las analogías de los resultados anteriroes en términos del espacio de desión, quiero decir, de las soluciones (débilmente) eficientes de los POM.

Proposicion 3.4. Supongamos que $\hat{x}$ es una solución óptima de un problema de optimización de sumas ponderadas

$$\min_{x \in \mathcal{X}} \sum_{k=1}^{p} \lambda_k \cdot f_k(x) \qquad (3.1)$$

con $\lambda \in \mathbb{R}^p_\geq$. En esa caso las siguientes afirmaciones se cumplen:

1 Si $\lambda \in \mathbb{R}^p_\geq$ entonces $x \in \mathcal{X}_{wE}$.

2 Si $\lambda \in \mathbb{R}^p_>$ entonces $x \in \mathcal{X}_E$.

3 Si $\lambda \in \mathbb{R}^p_\geq$ y $\hat{x}$ es una solución óptima única del (3.17), entonces $x \in \mathcal{X}_{sE}$.

Proposicion 3.5. Supongamos que $\mathcal{X}$ es un conjunto convexo, y que también son funciones convexas las f_k. Si $\hat{x} \in \mathcal{X}_{wE}$, entonces existe algún valor para $\lambda \in \mathcal{R}^p_\geq$ para el cual ese $\hat{x}$ es una solución óptima de (3.1).

3.1.3. Escalarización de sumas ponderadas y eficiencia propia

Estableceremos las relaciones entre los puntos no dominados propios (en sentido de Geoffrion) y los puntos óptimos de la escalarización por sumas ponderadas con pesos positivos. Los principales resultados muestran que estos puntos coinciden para conjuntos convexos.

Denotaremos al conjunto de puntos eficientes propios en sentido de Geoffrion por $\mathcal{Y}_{pE}$. Representaremos el conjunto de soluciones eficientes propias de un POM en sentido de Geoffrion como $\mathcal{X}_{pN}$. El siguiente teorema prueba que si $\hat{x}$ una solución óptima del problema de escalarización de sumas ponderadas (PESP) entonces también es una solución eficiente propia del POM si $\lambda > 0$.

Teorema 3.6. [5] Para $\lambda_k > 0, k = 1, \ldots, p$ *con* $\sum_{k=1}^p \lambda_k = 1$ pesos positivos, si $\hat{x}$ es una solución óptima del PESP entonces $\hat{x}$ es una solución propia eficiente del POM.

Prueba: [4]

Del teorema 3.6. se deduce que $\mathcal{S}(\mathcal{Y}) \subset \mathcal{Y}_{pE}$.

Como está probado con los teormeas 3.15. y 3.16. de [4], para conjuntos en general tenemos que,

$$\mathcal{S}(\mathcal{Y}) \subset \mathcal{Y}_{pE} \subset \mathcal{Y}_E \quad y \quad \mathcal{S}_(\mathcal{Y}) \subset \mathcal{Y}_{wE}$$

mientras que para conjuntos convexos tenemos que:

$$\mathcal{S}(\mathcal{Y}) = \mathcal{Y}_{pE} \subset \mathcal{Y}_E \subset \mathcal{Y}_{wE} = \mathcal{S}_(\mathcal{Y})$$

3.1.4. Conexión entre $\mathcal{S}(\mathcal{Y})$ y $\mathcal{X}_{(w,p)E}$

La conectividad es una propiedad topológica de los conjuntos de eficiencia y no dominancia, cuando $\mathcal{Y}_N$ o $\mathcal{X}_E$ poseen la conectividad, entonces los conjuntos completos de no dominancia o eficiencia pueden

explorarse, utilizando técnicas de busqueda local, comenzando por un único punto no dominado o eficiente. Además la conectividad también nos facilita la elección de una solución final x de compromiso al no haber diferencias con el conjunto de eficiencia.

Definición 3.7. Un conjunto $\mathcal{S} \subset \mathbb{R}^p$ se dice no concectado si puede ser escrito como $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, *con* $\mathcal{S}_1, \mathcal{S}_2 \neq \emptyset$, $cl(\mathcal{S}_1 \cap \mathcal{S}_2) = \mathcal{S}_1 \cap cl(\mathcal{S}_2) = \emptyset$. Equivalentemente, $\mathcal{S}$ es no conectado si existen conjunto abiertos $\mathcal{O}_1, \mathcal{O}_2$ tales que $\mathcal{S} \subset \mathcal{O}_1 cup \mathcal{O}_2$, $\mathcal{S} \cap \mathcal{O}_1 \neq \emptyset$, $\mathcal{S} \cap \mathcal{O}_2 \neq \emptyset$, $\mathcal{S} \cap \mathcal{O}_1 \cap \mathcal{O}_2 \neq \emptyset$. De cualquier otro modo, $\mathcal{S}$ se dice conectado.

Lema 3.8.

- Si $\mathcal{S}$ está conectado y $\mathcal{S} \subset \mathcal{U} \subset cl(\mathcal{S})$, entonces $\mathcal{U}$ es conectado.
- Si $\{\mathcal{S}_i : i \in \mathcal{I}\}$ es una familia de conjuntos conectados con $\cap_{i \in \mathcal{I}} \mathcal{S}_i = \emptyset$ entonces $\cup_{i \in \mathcal{I}} \mathcal{S}_i$ es conectado.

Utilizando el teorema 3.17 de [4] sabemos que $\mathcal{S}(\mathcal{Y}) \subset \mathcal{Y}_N \subset cl\mathcal{S}(\mathcal{Y})$ para conjuntos convexos. Se puede demostrar que las conectividad de $\mathcal{S}(\mathcal{Y})$ en el caso en que $\mathcal{Y}$ sea compacto, lo que implica la conectividad de $\mathcal{Y}_N$ por el lema 3.32.

Teorema 3.9. Si $\mathcal{Y}$ es un conjunto convexo y compacto, entonces $\mathcal{S}(\mathcal{Y})$ es conectado. Prueba en [4] teorema 3.33.

Teorema 3.10. Si $\mathcal{X}$ es un conjunto convexo y compacto, y además asumimos que las funciones objetivo $f_k : \mathbb{R}^p -> \mathbb{R}$ son continuas, entonces $\mathcal{X}_{wE}$ es conectado. Prueba en [4] teorema 3.34.

Teorema 3.11. Si $\mathcal{X} \subset \mathbb{R}^n$ es un conjunto convexo y compacto, y además asumimos que las funciones objetivo son convexas, entonces $\mathcal{X}_E$ es conectado. Prueba en [4] teorema 3.35.

3.1.5. Procedimiento para elegir el valor de lambda en el caso bi-objetivo

La elección de los pesos es fundamental para este método, ya que puede influir significativamente en las soluciones obtenidas. Diferentes combinaciones de pesos pueden generar la misma solucion, y por lo tanto, puede que variar los pesos no sea suficiente para encontrar todas las soluciones eficientes. Lo que nos puede llevar a no encontrar toda la frontera de pareto.

En el caso bi-objetivo, un procedimiento común es el algoritmo de NISE [3]:

1) Comenzamos maximizando ambos objetivos de manera individual, lo que nos deja dos puntos del espacio objetivo, así la imagen del óptimo para el objetivo Z_1 es P_1 y para el objetivo Z_2 es P_2. (En caso de haber varias soluciones óptimas alternativas, nos quedaremos con la maximice el otro objetivo). Denominemos P_2 y P_1 como S_1 y S_2 respectivamente, y además pongamos que $n = 2$. Además calcularemos la tasa de error actual $\Psi_{1,2} = \frac{area*2}{dist_e(P_1,P_2)}$ donde $area$ es la superficie del triángulo que forman P_1 y P_2 con el punto $(P_{1,y}, P_{2,x})$ y $dist_e(P_1, P_2)$ es la distancia euclídea entre P_1 y P_2. Además, aquí también podemos obtener el valor del error máximo tolerable que denotaremos como $T = \Psi_{1,2} * fraccionImpuesta$

2) Si $\Psi_{i,i+1} \leq T$ para $i = 1, 2, ..., n-1$ entonces termina el algoritmo, de modo que la aproximación formada por los puntos $S_1, S_2, ..., S_n$ así como las líneas que unen los puntos adyacentes es la solución. En caso contrario, se procede al paso 3.

3) Buscar el $\Psi_{i,i+1}, i = 1, 2, ..., n-1$, con el mayor valor. De modo que para la tupla $(imax, imax+1)$ que tenga el mayor de error se resuelve el problema de sumas ponderadas, utilizando la siguiente función:

$$maximize\ Z(x_1, ..., x_n; i, i+1) =$$
$$[Z_2(S_i) - Z_2(S_{i+1})] \times Z_1(x_1, ..., x_n) + [Z_1(S_{i+1}) - Z_1(S_i)] \times Z_2(x_1, ..., x_n)$$

Calculamos el $B_{imax,imax+1}$ sobre la función de pesos ponderados, evaluándola sobre S_{imax} o S_{imax+1}, obtendremos en ambos casos el mismo resultado. Si el resultado $Z(x_1, ..., x_n; i, i+1) = B_{i,i+1}$, entonces ponemos $\Psi_{i,i+1} = 0$ y retornamos al paso 2). En caso de que $Z(x_1, ..., x_n; i, i+1) > B_{i,i+1}$, entonces asignamos la solución óptima como pertenciente a la frontera de pareto P_{n+1}, y vamos al paso 4).

4) Reordenamos los puntos $P_t, t = 1, 2, ..., n+1$. Destacar que i es el valor de t tal que $Z_2(S_t) \geq Z_2(P_{n+1})$, es decir, la nueva solución tiene un valor de Z_2 que es el segundo más alto después de $Z_2(S_i)$. Lo que nos permite usar el siguiente esquema de reordenación:

$$S'_t = S_t, \quad t = 1, 2, ..., i$$

$$S'_{i+1} = P_{n+1}$$

$$S'_{t+1} = S_t, \quad t = i+1, ..., n$$

De la misma manera hay que reetiquear los términos Ψ'

$$\Psi'_{t,t+1} = \Psi_{t,t+1}, \quad t = 1, 2, ..., i-1 \quad (si \quad i \geq 2)$$

$$\Psi'_{t+1,t+2} = \Psi_{t,t+1}, \quad t = 1, 2, ..., n-1 \quad (si \quad i \leq n-2)$$

Calcular tanto $\Psi'_{i,i+1}$ como $\Psi'_{i+1,i+2}$ Y por último incrementar el valor de n en 1, y volver al paso 2).

Otra forma sería elegir los pesos decrementado el valor de λ en cada iteración, de modo que se obtenga una mayor cantidad de soluciones, sin embargo, estas soluciones no tienen por qué ser las más óptimas, a lo que hay que sumar el coste computacional sería significativamente mayor.

3.1.6. Limitaciones del método de sumas ponderadas

Como ya hemos dicho con anterioridad, para que el método de sumas ponderadas te permita obtener toda la frontera eficiente, se requiere que el conjunto de soluciones factibles sea convexo, ya que para ciertas regiones de la frontera de Pareto, este método no será capaz de encontrar soluciones eficientes, especialmente en las áreas no convexas. Una demostracion de esto se puede ver en [7].

En la programación entera, como el conjunto de soluciones eficientes no es convexo, las soluciones de la frontera de pareto que no pueden ser encontradas por el método de sumas ponderadas, se conocen como soluciones no soportadas. Aunque el método de sumas ponderadas no puede encontrar las soluciones no soportadas, puede proporcionar una aproximación de la frontera de Pareto, pero no garantiza encontrar identificar todas las soluciones no soportadas.

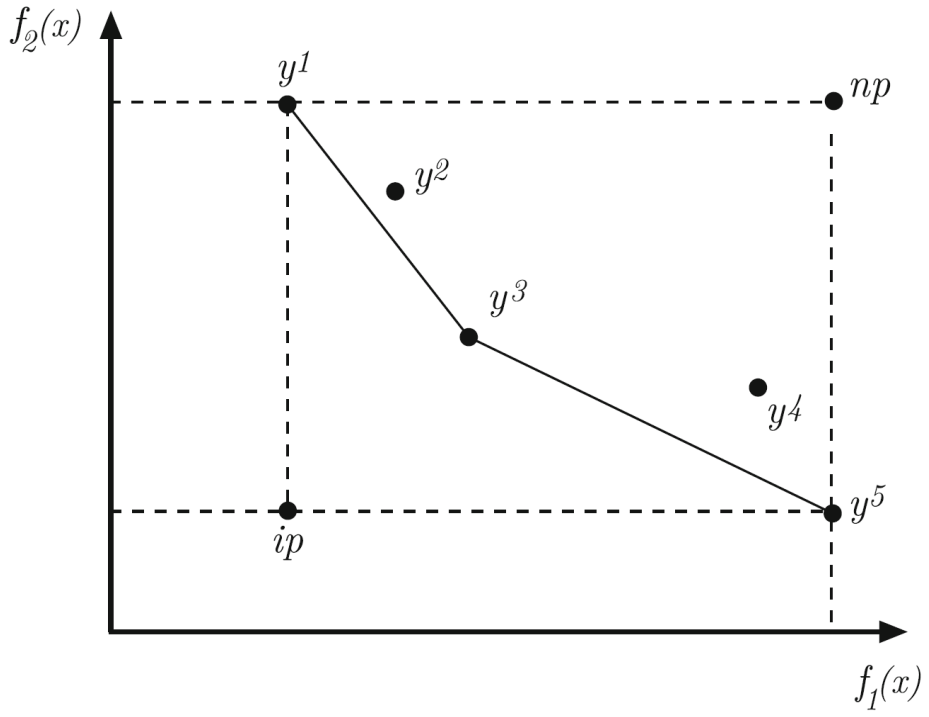

Figura 3.1: Soluciones no soportadas con el método de sumas ponderadas y puntos ideal y nadir [9].

En la figura 3.1 se puede ver como con método de sumas ponderadas hay algunas soluciones, que llamamos soluciones no soportadas, que no se puede llegar a alcanzar. Así por ejemplo, en la figura vemos que la envolvente inferior estaría formadas por las rectas que unen los puntos y1, y3 e y5 y sin embargo las soluciones y2 y y4 no están en esta recta y son parte de la frontera de Pareto, estas son un claro ejemplo de soluciones no soportadas. Por tanto, con el método de sumas ponderadas no vamos a poder obtener solo toda la frontera de Pareto, aún así el método nos da una aproximación de esta frontera.

3.2. Método ε-constraint

Este método de escalarización también es aplicable a problemas en los que $\mathcal{Y}$ no es convexo, a diferencia del método de las sumas ponderadas. Más allá de agregar objetivos adicionales, el método de ε-constraint trata de resolver el POM por medio de minimizar/maximizar uno de los objetivos originales, mientras que el resto se transforman en restricciones. Fue introducido por Yacov Y. Haimes, y una extensa revisión puede encontrarse en Chankong y Haimes [1].

Sustituimos el POM por el siguiente problema ε-constraint:

$$\begin{aligned} \text{Minimizar} \quad & f_j(x) && (3.5)\\ s.a. \quad & f_k(x) \leq \varepsilon_k, \ \textit{para todo } k \neq j && (3.6)\\ & x \in \mathcal{X} && (3.7) \end{aligned}$$

donde:

- $\varepsilon_{\mathbf{k}} \quad \in \mathbb{R}^p$ *para todo* $k \neq j$

Es habibitual que, al estar optimizando la función uni-objetivo, obtengamos para un mismo valor ε varias soluciones óptimas pero siempre deberemos quedarnos con la mejor para cada ε. Esto en el caso biojetivo, que es el que nos ocupa, es muy sencillo, puesto que siempre nos quedaremos con la solución que minimice/maximice, según corresponda, el otro objetivo.

Para justificar este enfoque, basta con mostrar que las soluciones óptimas del problema (4.1) son como poco débilmente eficientes. Un condición necesaria y suficiente para la eficiencia muestra que este método funciona para problemas en general, por lo que no es necesaria la asunción de convexidad.

Proposición 4.1. Si $\hat{x}$ es una solución óptima de (4.1) para algún j, entonces $\hat{x}$ es débilmente eficiente.

Prueba. Supongamos que $\hat{x} \neq \mathcal{X}_{wE}$. Entonces existiría algún $x \in \mathcal{X}$ con $f_k(x) \leq f_k(\hat{x})$ para todo $k = 1, \ldots, p$. En particular, $f_j(x) \leq f_j(\hat{x})$. Dado que $f_k(x) < f_k(\hat{x}) \leq \varepsilon_k$ para $k \neq j$, la solución x sería factible para (4.1). Esto contradice la suposición de que $\hat{x}$ es una solución óptima de (4.3).

Para fortalecer la Proposición 4.1 y obtener eficiencia, necesitamos que la solución óptima de (4.1) sea única:

Proposición 4.2. Si $\hat{x}$ es una solución óptima única de (4.3) para algún j, entonces $\hat{x} \in \mathcal{X}_{sE}$ (y por tanto $\hat{x} \in \mathcal{X}_E$).

Prueba. Supongamos que exite algún $x \in \mathcal{X}$ con $f_k(x) \leq f_k(\hat{x}) \leq \varepsilon_k$ *para todo* $k \neq j$. Si además, $f_j(x) < f_j(\hat{x})$, deberíamos tener que $f_j(x) = f_j(\hat{x})$ ya que $\hat{x}$ es una solución óptima única de (4.3). Por tanto, la unicidad de $\hat{x}$ implica que $x = hatx$, y poi tanto $\hat{x} \in \mathcal{X}_{sE}$.

En general, la eficiencia de $\hat{x}$ está relacionada con que $\hat{x}$ sea una solución óptima única de (4.3) para todo $j = 1, \ldots, p$ con el mismo valor de ε utilizado para todos estos problemas.

Teorema 4.3. Una solución factible $\hat{x}$ es eficiente si y solo sí existe un vector $\varepsilon \in \mathbb{R}^p$ tal que $\hat{x}$ es una solución óptima única de (4.1) para todo $j = 1, \ldots, p$.

Prueba en el teorema 4.3. de [4].

Por tanto, el teormea 4.3. da a entender que con las elecciones apropiadas de ε pueden encontrarse todas las soluciones eficientes. Sin embargo, estos valores de ε_j son iguales que el valor del objetivo de

la solución eficiente correspondiente que uno querría encontrar. Una confirmación o comprobación de la eficiencia se obtiene más allá de descubirir las soluciones eficientes.

Denotaremos por

$$\mathcal{E}_j := \{\varepsilon \in \mathbb{R}^p \{x \in \mathcal{X} : f_k(x) \leq \varepsilon_k,\ k \neq j\} \neq \emptyset\}$$

al conjunto de valores de ε para los cuales el problema (4.1) es factible, y por

$$\mathcal{X}_j(\varepsilon) := \{x \in \mathcal{X} : x \textit{ es una solucin ptima de } (4{,}3)\}$$

para $\varepsilon \in \mathcal{E}_j$ al conjunto de soluciones óptimas de (4.1). Gracias al teorema 4.3 y a la Proposicion 4.1. tenemos que para cada $\varepsilon \in \cap_{k=1}^{p} \mathcal{E}_j$

$$\bigcap_{j=1}^{p} \mathcal{X}_j(\varepsilon) \subset \mathcal{X}_E \subset \mathcal{X}_j(\varepsilon) \subset \mathcal{X}_{wE}$$

para todo $j = 1, \ldots, p$.

3.2.1. Más acerca del método de ε-constraint

Un análisis más que detallado de este método puede encontrarse en el artículo [9], donde se estudia más en profundidad este método de resolver problemas de programación entera biobjetivo. En ese artículo se identifican y estudian hasta seis variantes conocidas del método de ε-constraint, y además se introducen tres nuevas variantes, dos basadas en parámetros de aumento dependientes de los datos y una basada en restricciones elásticas.

3.2.2. Diferencia principal con el método de sumas ponderadas

A diferencia del método de sumas ponderadas, visto en la sección anterior, el método de ε-constraint si que obtiene toda la frontera de Pareto, es decir, todas las soluciones eficientes.

3.3. Otros métodos:

- **Método de la Restricción Eslástica** [4].
  - Al igual que el método de ε-constraint, el método de la restricción elástica también se basa en la idea de seleccionar un objetivo principal y tatar el resto de objetivos como restricciones, en este caso restricciones elásticas, ya que ahora no tenemos un valor fijo para ε sino que este valor puede variar.
  - Las restricciones elásticas son de la forma $f_i(x) \leq z_i + s_i, i \neq objPrincial\_idx$, donde z_i es representa el límite permitido para el objetivo i y s_i es la variable de holgura positiva que permite relajar la restricción, en caso de que fuera necesario.
  - Además, esta variable de holgura s_i se añade a la función objetivo principal, de modo que esta queda de la siguiente forma:

    $$\min_{x \in \mathcal{X}} f_{objPrincipal}(x) + \lambda \times \sum_{i \neq objPrincipal\_idx} s_i$$

    y por tanto el problema se resuelve variando el valor de λ.
- **Método de la Programación por Metas** [4].
  - Consiste en transformar el problema multi-objetivo en una única función objetivo, que se obtiene medinate la suma de las desviaciones de los objetivos originales a las metas establecidas g_i. De modo que la función objetivo a minimizar sea de la forma:

    $$\min_{x \in \mathcal{X}} \sum_{i=1}^{m} \omega_i \times (d_i^+ + d_i^-)$$

- d_i^+ y d_i^- son respectivamente las desviaciones positivas y negativas, esto es cuando el valor de la función objetivo es mayor o menor que la meta establecida ($f_i(x) > g_i$ y $f_i(x) < g_i$).
- ω_i es un peso que se le da a cada objetivo, variando el cual se puede dar más importancia a un objetivo que a otro y por tanto se puede obtener la aproximación a la frotera de parteto.

- **Método de Normas de Chebychev** [4].
  - Trata de miminzar la máxima desviación respecto a un punto de referencia, como puede ser el punto ideal.
  - Las funciones objetivo se transforman en una única función objetivo de la forma:

$$\text{mín} \max_{i=1}^{m} \omega_i \times |f_i(x) - z_i|$$

    donde z_i es el valor de referencia para el objetivo i, mientras que ω_i es el peso que se le da a cada objetivo.
  - Por tanto, lo que se busca es minimizar la máxima devición de cada objetivo respecto a su valor de referencia, lo que nos permite obtener una aproximación de la frontera de Pareto a través de la variación de los pesos ω_i.
  - Este método también permite obtener la frontera de Pareto completa, al igual que el método de ε-constraint.

Capítulo 4

Cubrimiento máximo

En este capítulo se explicarán de los problemas uniobjetivo que se van a resolver en el problema biobjetivo de ejemplo.

4.1. Problema de cubrimiento máximo o parcial (MCLP)

Se parte de que el número de puntos de servicio que pueden abrirse generalmente está prefijado, por algún tipo de limitación del problema, y por tanto lo que se busca es maximizar la demanda cubierta. Este problema, MCLP (Maximal Covering Location Problem), tiene multitud de dirversas aplicaciones en la vida real, como puede verse en el artículo de Chen-Hua Chung. [2]

La notación que emplearé tanto para este modelo como para el MCLPR (Problema de cubrimiento maximo reforzado) es la siguiente:

- $M = 1, 2, ..., m$ es el conjunto de los puntos de demanda
- $N = 1, 2, ..., n$ es el conjunto de los puntos de servicio, donde se pueden abrir las instalaciones
- Para cada $i \in M$ se tiene el conjunto $N_i \subseteq N$ de los puntos de servicio que pueden cubrir la demanda del punto i. El caso habitual, es aquel en el que viene dado a través de las distancias: o bien se conocen; o bien se calculan, la distancias d_{ij} entre los puntos de demanda i y los puntos de servicio j.
- Además se fija la distancia de cubrimiento dc que es la distancia máxima a la que los puntos de servicio puede cubrir la demanda de los puntos de demanda. Por tanto:

$$N_i = j \in N : d_{ij} \leq d_C$$

- En el MCLP es habitual considerar la demanda h_i de cada punto demanda $i \in M$, así como que el número de instalaciones a situar sea fijo, p. De modo que el objetivo del MCLP sea maximizar la demanda total que va a quedar cubierta, sujeto al número de instalaciones que se pueden abrir, p.
- De este modo, definimos para cada posible punto de servicio $j \in N$ la variable binaria x_j que toma el valor 1 si se abre la instalación en el punto j, y 0 en caso contrario. De igual modo, definimos la variable binaria y_i que toma el valor 1 si la demanda del punto demanda i queda cubierto, y 0 en otro caso.

De este modo el modelo para el MCLP es el siguiente:

$$\begin{aligned} \text{máx} \quad & \sum_{i \in M} h_i y_i \\ \text{sujeto a:} \quad & \sum_{j \in N_i} x_j \geq y_i \quad \forall i \in M \\ & \sum_{j \in N} x_j = p \\ & x_j \in \{0,1\} \quad \forall j \in N \\ & y_i \in \{0,1\} \quad \forall i \in M \\ & N_i = \{j \,|\, d_{ij} \leq d_C\} \quad \forall i \in M \end{aligned}$$

Es común encontrarse el MCLP formulado con una matriz binaria A de dimensiones $m \times n$ donde $a_{ij} = 1$ si el punto de servicio $j \in N$ puede cubrir al punto demnanda $i \in M$, esto es, si $j \in N_i$; y $a_{ij} = 0$ en otro caso. De este modo, las restricciones de cubrimiento se pueden reescribir en función de los coeficientes de la matriz A:

$$\sum_{j=1}^{n} a_{ij} \times x_j \geq y_i, \;\; i = 1, ..., m$$

4.1.1. Implementación del MCLP en Xpress

```
declarations
    pdemanda= 1..m
    pservicio = 1..n
    cx, cy:array(pservicio)of integer
    dist:array(pdemanda,pservicio)of real
    dem:array(pdemanda)of real
    a:array(pdemanda,pservicio)of integer
    dc=15
    y: array(pdemanda) of mpvar
    x: array(pservicio) of mpvar
    p=3
end-declarations

!!!! Encontrar la solucion optima
obj := sum(j in pdemanda) dem(j)*y(j)

forall(i in pdemanda)
    res1(i):= sum(j in pservicio)x(j)*a(i,j)>= y(i)


res2 := sum(j in pservicio) x(j) = p

forall(i in pdemanda) y(i) is_binary
forall(j in pservicio) x(j) is_binary

maximize(obj)

writeln("")
writeln("La solucion optima es\t", getobjval)

writeln("")
forall( j in pservicio| x(j).sol>0.9999 )
    writeln("Se habre el punto de servicio\t", j)
```

4.2. Modelo de cubrimiento máximo reforzado (MCLPR)

La idea detrás del MCLPR (Maximal Covering Location Problem with Reinforcement) es la de reforzar el cubrimiento de la demanda de los puntos de servicio, de manera que si el mejor servicio que se puede prestar servicio a un punto de demanda está ocupado o bien temporalmente no disponible por el motivo que fuera, exista otro punto de servicio que pueda cubrir la demanda de ese punto de demanda.

Por tanto estaríamos asignando a cada punto de demanda varios puntos de servicio, de manera que si el mejor no puede proporcionar el servicio, sea otro el que sea haga cargo de satisfacer esa demanda demanda.

Basándonos en la notación y esquema del MCLP, para el modelo MCLPR bastaría con añadir una nueva variable binaria u_i, $i \in M$ que toma el valor 1 si el punto de demanda i es cubierto por al menos dos puntos de servicio j, y 0 en otro caso.

De este modo, la formulación del probelma MCLPR sería la siguiente:

$$
\begin{aligned}
& \text{máx} \sum_{i \in M} h_i u_i \\
\text{sujeto a:} \quad & \sum_{j \in N_i} x_j \geq 2 * u_i \quad \forall i \in M \\
& \sum_{j \in N} x_j = p \\
& x_j \in \{0,1\} \quad \forall j \in N \\
& u_i \in \{0,1\} \quad \forall i \in M \\
& N_i = \{j \,|\, d_{ij} \leq d_C\} \quad \forall i \in M
\end{aligned}
$$

```
declarations
    pdemanda= 1..m
    pservicio = 1..n
    cx, cy:array(pservicio)of integer
    dist:array(pdemanda,pservicio)of real
    dem:array(pdemanda)of real
    a:array(pdemanda,pservicio)of integer
    dc=15
    y: array(pdemanda) of mpvar
    x: array(pservicio) of mpvar
    p=4
end-declarations

!!!! Encontrar la solucion optima
obj := sum(j in pdemanda) dem(j)*u(j)

forall(i in pdemanda)
    res1(i):= sum(j in pservicio)x(j)*a(i,j) >= 2*u(i)

res2 := sum(j in pservicio) x(j) = p

forall(i in pdemanda) u(i) is_binary
forall(j in pservicio) x(j) is_binary

maximize(obj)

writeln("")
writeln("La solucion optima es\t", getobjval)

writeln("")
forall( j in pservicio| x(j).sol>0.9999 )
    writeln("Se habre el punto de servicio\t", j)
```

4.3. Problema BACOP2

El problema BACOP2 (Bi-objective Area Covering Problem) es un problema biobjetivo que combina los problemas MCLP y MCLPR, en el cual se trata de maximizar la demanda doblemente (o más veces) cubierta, con un número de puntos de servicios fijo, p. Cabe destacar que el problema BACOP2 fue desarrollado por HOGAN, K. y REVELLE, C. en [6].

La formulación del problema BACOP2 es la siguiente:

$$
\begin{aligned}
& \text{máx} \sum_{i \in M} h_i y_i \\
& \text{máx} \sum_{i \in M} h_i u_i \\
\text{sujeto a:} \quad & \sum_{j \in N_i} x_j \geq y_i \quad \forall i \in M \\
& \sum_{j \in N_i} x_j \geq 1 + u_i \quad \forall i \in M \\
& \sum_{j \in N} x_j = p \\
& x_j \in \{0,1\} \quad \forall j \in N \\
& y_i \in \{0,1\} \quad \forall i \in M \\
& u_i \in \{0,1\} \quad \forall i \in M \\
& N_i = \{j \,|\, d_{ij} \leq d_C\} \quad \forall i \in M
\end{aligned}
$$

4.3.1. Implementaciones para aproximar el problema BACOP2

En el capítulo 5 5.

Capítulo 5

Formulación y solución del problema BACOP2 en un caso real

En este capítulo se implementará un caso real de problema biobjetivo, y para la implementación se van a utilizar los métodos anteriormente descritos.

5.1. Implementción del método de sumas ponderadas con NISE

Peudo-código explicando como se ha implementado el método de sumas ponderadas con NISE.

1. Declaraciones:
   - `(cx, cy, cSx, cSy)`: vectores para las posiciones de los puntos de servicios y demandas.
   - `(dist)`: matriz para las distancias entre los puntos de servicios y demandas.
   - `(dc)`: variable con la distancia máxima de cobertura para los puntos de servicios.
   - `(a)`: una matriz binaria que toma el valor 1 cuando el punto de servicio i puede cubrir el punto de demanda j, y 0 en otro caso.
   - `(Pn)`: matriz bidimensional para guardar las soluciones de la frontera de Pareto.
   - `(Sn)`: matriz bidimensional para guardar las soluciones ordenadas según el método de NISE.
   - `(Phis)`: vector de valores para la diferencia de criterio NISE entre las soluciones ordenadas consecutivas.
   - `(x, y, u)`: vectores vistos para resolver los modelos anteriores uniobjetivo.
   - `(psAbiertos)`: matriz de enteros que contiene para cada solución de Pn el identificador de los puntos de servicios abiertos.
2. Comenzamos maximizando ambos objetivos de manera individual, lo que nos deja dos puntos en el espacio objetivo. Así, la imagen del óptimo para el objetivo Z_1 es $Pn[1,1]$ y para el objetivo Z_2 es $Pn[2,2]$. Como puede haber varias soluciones óptimas alternativas para cada uno, nos quedaremos con aquellas que maximicen el criterio opuesto, respectivamente, poniendo como restricción que el primer objetivo quede cubierto con los valores previamente obtenidos, también respectivamente.
3. De este modo ya tenemos $Pn[1]$ y $Pn[2]$ que denominamos respectivamente como $Sn[2]$ y $Sn[1]$ ya que hemos dicho que Sn contiene las soluciones ordenadas.
4. Establecemos el número de puntos ($n = 2$) ya que tenemos dos soluciones en la frontera de Pareto.
5. Calculamos el valor del criterio NISE para las soluciones que tenemos como sigue:

$$\Psi_{1,2} = \frac{area * 2}{dist_e(P_1, P_2)}$$

donde:

- *area* es la superficie del triángulo que forman P_1 y P_2 con el punto $(P_{1,y}, P_{2,x})$
- $dist_e(P_1, P_2)$ es la distancia euclídea entre P_1 y P_2

6. Obtenemos el valor del error máximo tolerable que denotaremos como $T = \Phi[1,2] * fraccionImpuesta$.

7. Si $\Psi_{i,i+1} \leq T$ para $i = 1, 2, ..., n-1$ entonces termina el algoritmo, de modo que aproximación precisa formada por los puntos $S_1, S_2, ..., S_n$ así como las líneas que unen los puntos adyacentes es la solución. En caso contrario, se procede al paso siguiente (8).

8. Buscar el $\Psi_{i,i+1}, i = 1, 2, ..., n-1$, con el mayor valor. De modo que para la tupla $(imax, imax+1)$ que tenga el mayor de error más grande se resuelve el problema de sumas ponderadas, utilizando la siguiente función:

$$maximize\ Z(x_1, ..., x_n; i, i+1) =$$

$$[Z_2(S_i) - Z_2(S_{i+1})] \times Z_1(x_1, ..., x_n) + [Z_1(S_{i+1}) - Z_1(S_i)] \times Z_2(x_1, ..., x_n)$$

9. Calculamos el $B_{imax,imax+1}$ sobre la función de pesos ponderados, evaluándola sobre S_{imax} o S_{imax+1}, obtendremos en ambos casos el mismo resultado. Si el resultado $Z(x_1, ..., x_n; imax, imax+1) = B_{imax,imax+1}$, entonces ponemos $\Psi_{i,i+1} = 0$ y retornamos al paso (7). En caso de que $Z(x_1, ..., x_n; imax, imax+1) > B_{imax,imax+1}$, entonces designamos asignamos la solución óptima como pertenciente a la frontera de pareto P_{n+1}, y vamos al paso (7).

10. Reordenamos los puntos $P_t, t = 1, 2, ..., n+1$. Destacar que i es el valor de t tal que $Z_2(S_t) \geq Z_2(P_{n+1})$, es decir, la nueva solución tiene un valor de Z_2 que es el segundo más alto después de $Z_2(S_i)$. Lo que nos permite usar el siguiente esquema de reordenación:

$$S'_t = S_t, \quad t = 1, 2, ..., i$$

$$S'_{i+1} = P_{n+1}$$

$$S'_{t+1} = S_t, \quad t = i+1, ..., n$$

De la misma manera hay que reetiquear los términos Ψ'

$$\Psi'_{t,t+1} = \Psi_{t,t+1}, \quad t = 1, 2, ..., i-1 \quad (si \quad i \geq 2)$$

$$\Psi'_{t+1,t+2} = \Psi_{t,t+1}, \quad t = 1, 2, ..., n-1 \quad (si \quad i \leq n-2)$$

Calcular tanto $\Psi'_{i,i+1}$ como $\Psi'_{i+1,i+2}$ Y por último incrementar el valor de n en 1, y volver al paso (7).

El código correspondiente a mi implementacion de este método se encuentra en el anexo del TFG A.

5.2. Implementción del método ϵ-constraint:

Peudo-código explicando como se ha implementado el método ϵ-constraint:

1. Declaraciones:

- `(cx, cy, cSx, cSy)`: vectores para las posiciones de los puntos de servicios y demandas.
- `(dist)`: matriz para las distancias entre los puntos de servicios y demandas.
- `(dc)`: variable con la distancia máxima de cobertura para los puntos de servicios.
- `(a)`: una matriz binaria que toma el valor 1 cuando el punto de servicio i puede cubrir el punto de demanda j, y 0 en otro caso.
- `(Pn)`: matriz bidimensional para guardar las soluciones de la frontera de Pareto.
- `(x, y, u)`: vectores vistos para resolver los modelos anteriores uniobjetivo.
- `(psAbiertos)`: matriz de enteros que contiene para cada solución de Pn el identificador de los puntos de servicios abiertos.

2. Buscamos la solución ópitma uniobjetivo para uno de los dos objetivos, por ejemplo, para el objetivo Z_1. Cabe destacar que de nuevo puede haber varias soluciones óptimas alternativas, por lo que nos quedaremos con aquellas que maximicen el otro objetivo, Z_2, poniendo como restricción que Z_1 tome el valor óptimo previamente obtenido. Este primer valor optimo servirá para saber hasta que valor de ε iterar, ya que es el maximo valor que puede tomar el objetivo en cuestion, asignamos este valor a una varaible llamada "varepMax".

3. Buscamos ahora la solución óptima para el otro objetivo, Z_2, y de nuevo nos quedaremos con aquellas que maximicen el otro objetivo, Z_1, poniendo como restricción que Z_2 tome el valor óptimo previamente obtenido. Este segundo valor no servirá para saber desde que valor de ε comenzar a iterar y de este modo nos ahorramos cómputo innecesario, puesto que esas soluciones son son atractivas. Por tanto, creamos una variable "varepIter"que tome este valor.

4. Encontramos la solución óptima para el segundo de los objetivos, Z_2, incluyendo como restricción extra que el primer objetivo tome al menos el valor de varepIter, y guardamos el valor de Z_2 en la variable para después volver a optimizar, pero por el otro de los objetivos, Z_1, poniendo como restricción ahora que el valor de Z_2 sea mayor o igual que el valor de esta última variable mencionada. El valor de Z_1 lo guardamos en la variable "varepIterz así nos hemos evitado un computo inncesario de soluciones dominadas. La solución que obtenemos pertence al conjunto de Pareto, y la guardamos en la matriz Pn.

5. Incrementamos en uno la variable varepIter, si este es mayor que "varepMax", entonces hemos terminado, sino volvemos al paso (4).

El código correspondiente a mi implementación de este método se encuentra en el anexo del TFG A.

Capítulo 6

Puesta a prueba de las soluciones al problema BACOP2 con varios conjuntos de datos

Ejecutaré los códigos anteriores para resolver el problema BACOP2 4.3 para varios conjuntos de datos y realizaré al final del capítulo una pequeña comparación de ambos métodos.

6.1. Conjunto de pruebas simulado

El primer conjunto de pruebas que se ha utilizado es un conjunto de datos simulado, el cual consta únicamente 200 puntos de servicio y 400 puntos de demanda. El conjunto de datos fue generado de manera aleatoria, por lo que no se corresponde con un caso real, sin embargo ha sido muy útil para poder probar el correcto funcionamiento de los algoritmos implementados. Puede encontrarse en el repositorio de codigo y conjuntos de datos A. Cabe destacar que el problema bi-objetivo que se resuelve en este caso es el BACOP2 4.3, siendo como ya se explicó en la sección 4.3 los objetivos a maximizar el MCLP 4.1 y el MCLPR 4.2, teniendo siempre en cuenta que todos los puntos demanda tienen demanda igual a 1. El número máximo de puntos de servicio que se pueden abrir es 15 y además se fija una distancia de cubrimimiento máxima de 10 unidades métricas. Por tanto en este primer problema lo que se busca es maximizar el número de puntos de demanda que quedan cubiertos.

6.1.1. Resultados obtenidos con el método de ε-constraint

Lo primero que vemos es una gráfica 6.1 correspondiente a las soluciones de la frontera de Pareto obtenidas con el método de ε-constraint.

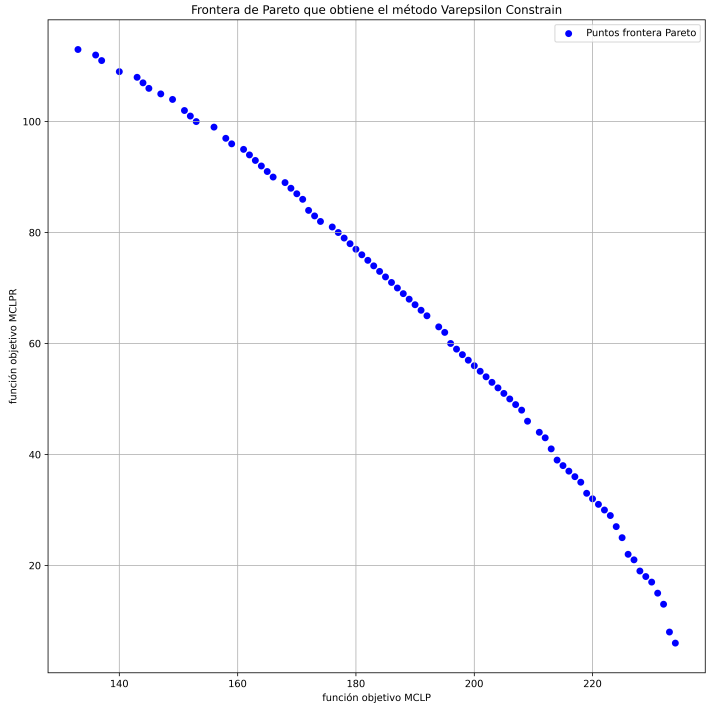

Figura 6.1: Frontera de Pareto obtenida con el método de ε-constraint

A continuación se muestra una tabla 6.1 con los resultados obtenidos para el conjunto de pruebas simulado con el método de ε-constraint.

Id	**MCLP**	**MCLPR**
1	234	6
2	233	8
3	232	13
4	231	15
5	230	17
6	229	18
...	...	...
79	145	106
80	144	107
81	143	108
82	140	109
83	137	111
84	136	112

Tabla 6.1: Resultados de MCLP y MCLPR

Finalmente tenemos una gráfica 6.2 que muestra la mejor solución obtenida para el primero de los objetivos (MCLP 4.1), otra gráfica 6.3 que muestra la mejor solución obtenida para el segundo de los objetivos (MCLPR 4.2) y una última gráfica 6.4 que muestra una solución intermedia de compromiso entre ambos objetivos.

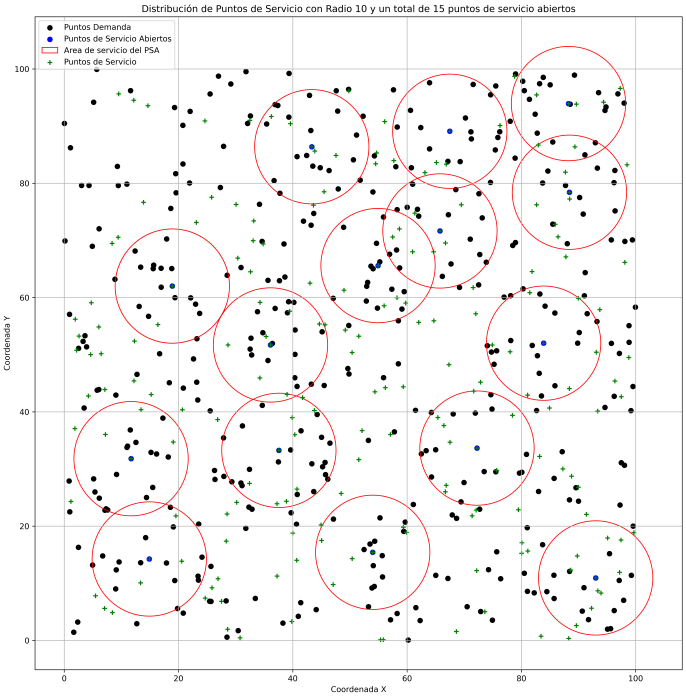

Figura 6.2: Mejor solución para el MCLP con el método de ε

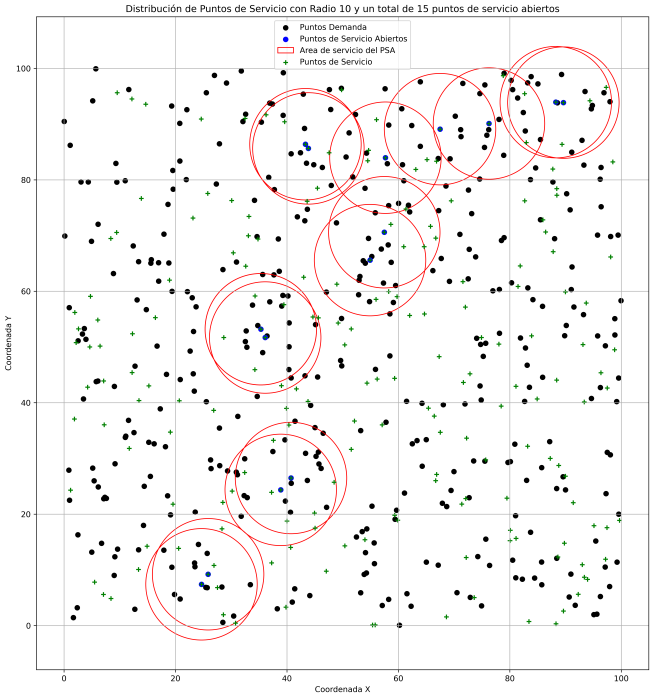

Figura 6.3: Mejor solución para el MCLPR con el método de ε

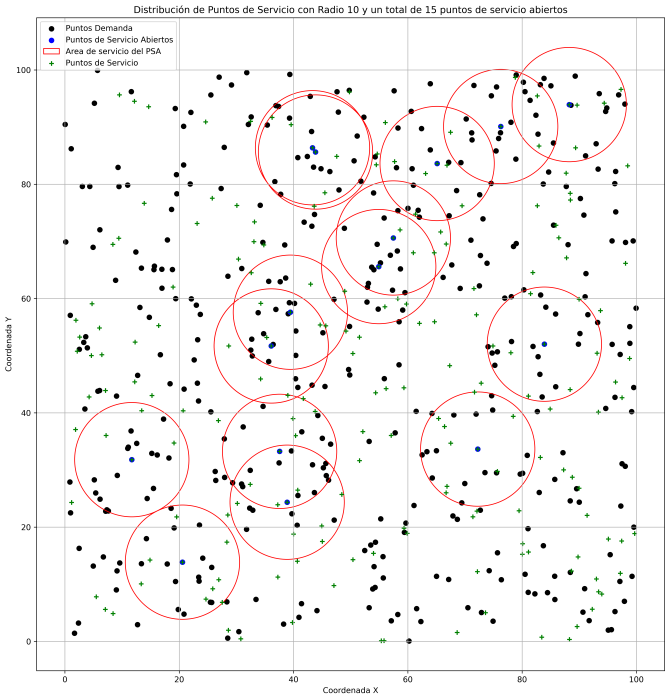

Figura 6.4: Solucion de compromiso BACOP2 con el método de ε

6.1.2. Resultados obtenidos con el método de NISE

Lo primero que vemos es una gráfica correspondiente a las soluciones de la frontera de Pareto obtenidas con el método de NISE.

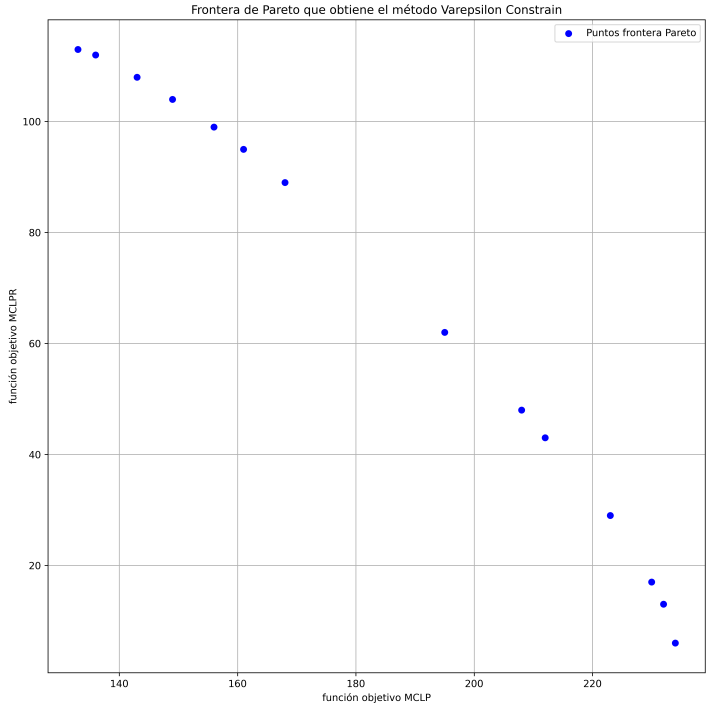

Figura 6.5: Frontera de Pareto obtenida con el método de NISE

A continuación se muestra una tabla 6.2 con los resultados obtenidos para el conjunto de pruebas simulado con el método NISE.

Id	**MCLP**	**MCLPR**
1	234	6
2	133	113
3	195	62
4	161	95
5	223	29
6	168	89
7	208	48
8	143	108
9	212	43
10	156	99
11	232	13
12	230	17
13	149	104

Tabla 6.2: Resultados de MCLP y MCLPR

Finalmente tenemos una gráfica 6.6 que muestra la mejor solución obtenida para el primero de los objetivos (MCLP 4.1), otra gráfica 6.7 que muestra la grafica de la mejor solución obtenida para el segundo de los objetivos (MCLPR 4.2) y una última gráfica 6.8 que muestra una solución intermedia de compromiso entre ambos objetivos.

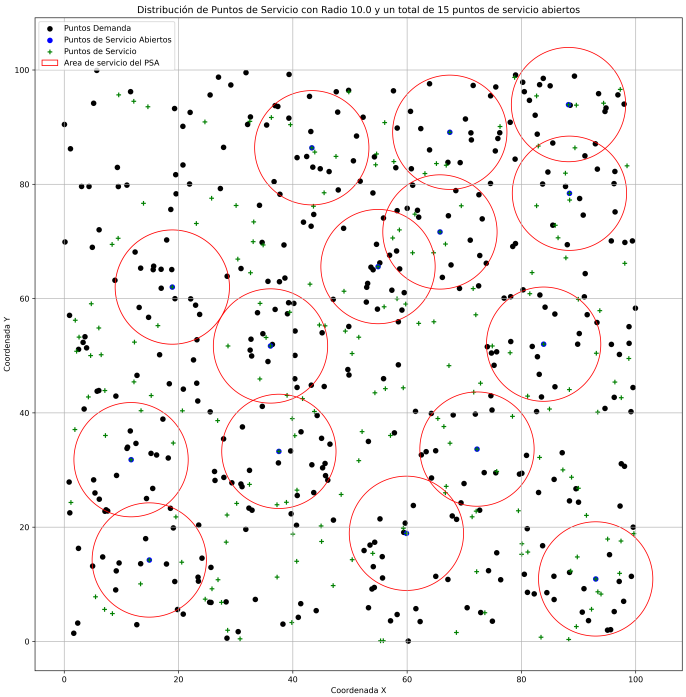

Figura 6.6: Mejor solución para el MCLP con el método de NISE

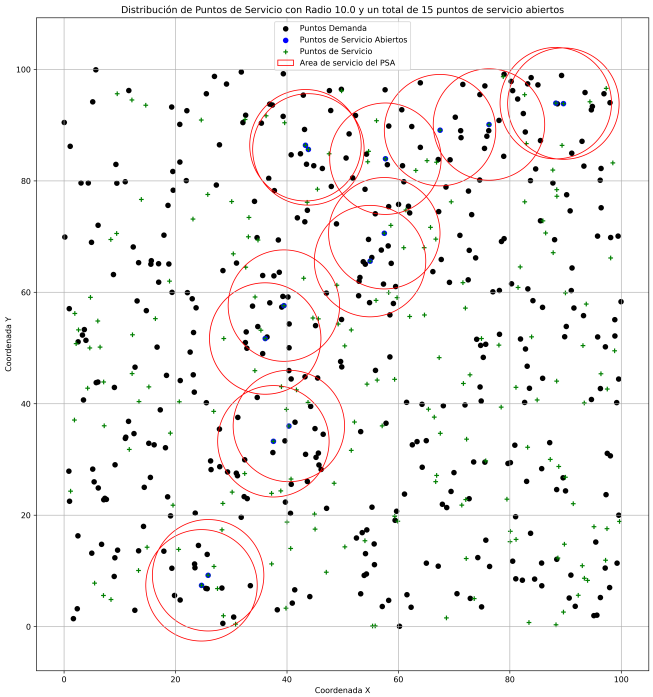

Figura 6.7: Mejor solución para el MCLPR con el método de NISE

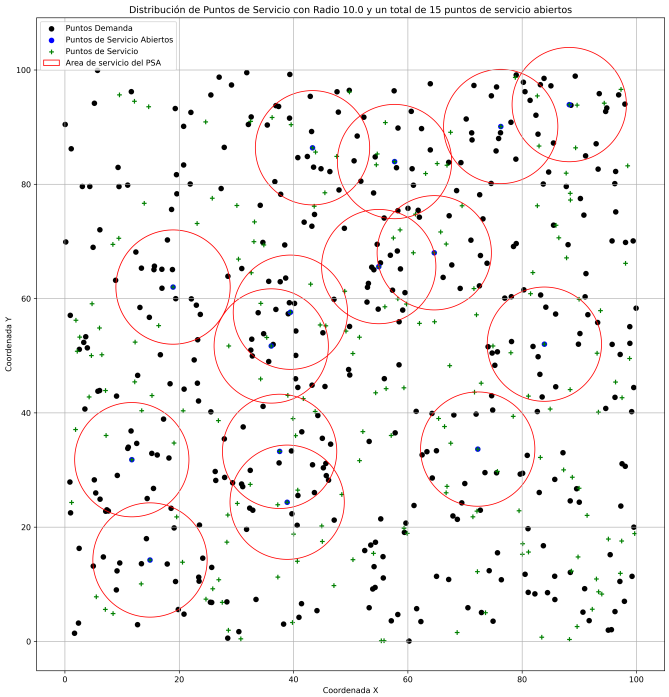

Figura 6.8: Solucion de compromiso BACOP2 con el método de NISE

6.2. Conjunto de pruebas con datos reales de Castilla y León

Este conjunto de pruebas representa de manera realista cada uno de los municipios de Castilla y León, por tanto contamos con un total de 2248 puntos de demanda, para los puntos de servicio, consideraremos solo aquellos municipios con una población superior a los 1000 habitantes, lo que nos deja con unos 239 puntos de servicio. El conjunto puede encontrarse en la siguiente dirección: A. Cabe destacar que el problema bi-objetivo que se resuelve en este caso es el BACOP2 4.3, siendo como ya se explicó en la sección 4.3 los objetivos a maximizar el MCLP 4.1 y el MCLPR 4.2. El número máximo de puntos de servicio que se pueden abrir es 150 y además se fija una distancia de cubrimimiento máxima de 10000 unidades métricas. Cabe decir que en este problema lo que se busca maximizar por los dos objetivos el la población cubierta, es decir, el número de habitantes que quedan cubiertos por al menos un punto de servicio (MCLP) y al menos dos veces (MCLPR).

6.2.1. Resultados obtenidos con el método de ε-constraint

Lo primero que vemos es una gráfica 6.9 correspondiente a las soluciones de la frontera de Pareto obtenidas con el método de ε-constraint.

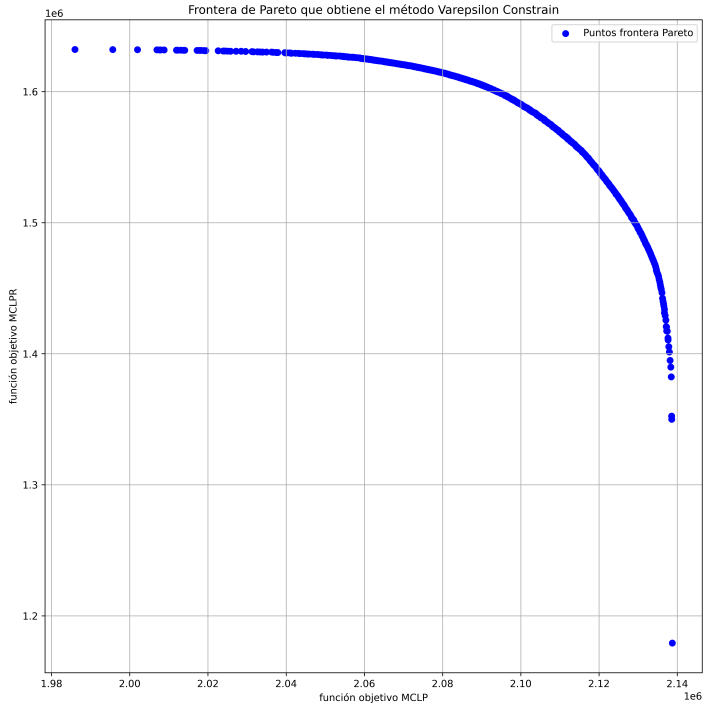

Figura 6.9: Frontera de Pareto obtenida con el método de ε-constraint

A continuación se muestra una tabla con los resultados obtenidos para el conjunto de Castilla y León con el método de ε-constraint, debido a la gran cantidad de puntos se opta por mostrar la matriz transpuesta de la tabla para que sea más legible.

Id	MCLP	MCLPR
1	2138741	1179194
2	2138585	1349998
3	2138540	1352380
4	2138462	1382367
5	2138334	1389743
6	2138155	1394931
...	...	...
1169	2008797	1631789
1170	2007929	1631805
1171	2007617	1631821
1172	2006947	1631883
1173	2001969	1632020
1174	1995658	1632107

Tabla 6.3: Resultados de MCLP y MCLPR para el conjunto de Castilla y León con el método varepsilon constraint

Finalmente tenemos una gráfica 6.10 que muestra la mejor solución obtenida para el primero de los objetivos (MCLP 4.1), otra gráfica 6.11 que muestra la mejor solución obtenida para el segundo de los objetivos (MCLPR 4.2) y una última gráfica 6.12 que muestra una solución intermedia de compromiso entre ambos objetivos.

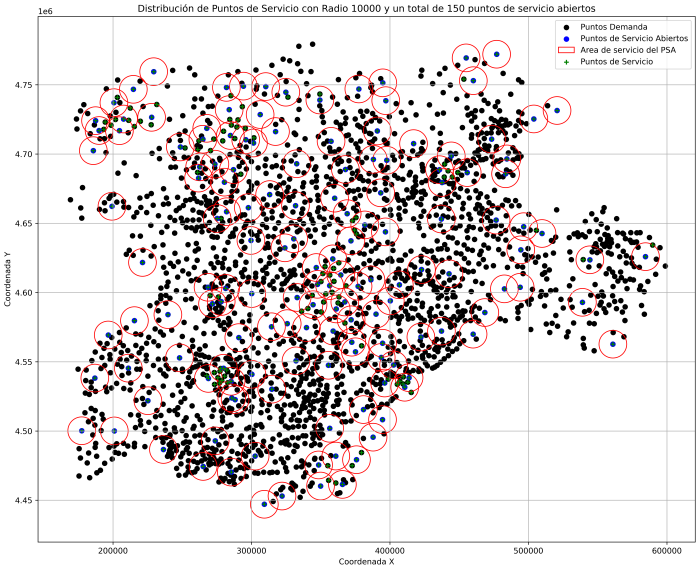

Figura 6.10: Mejor solución para el MCLP con el método de ε

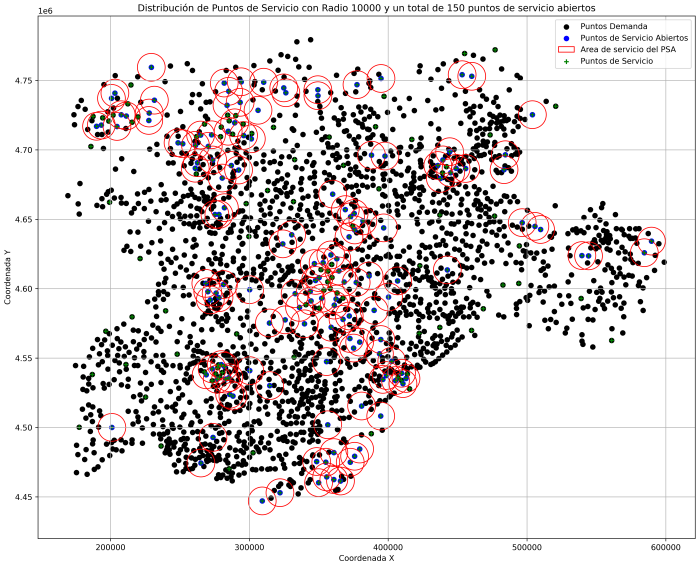

Figura 6.11: Mejor solución para el MCLPR con el método de ε

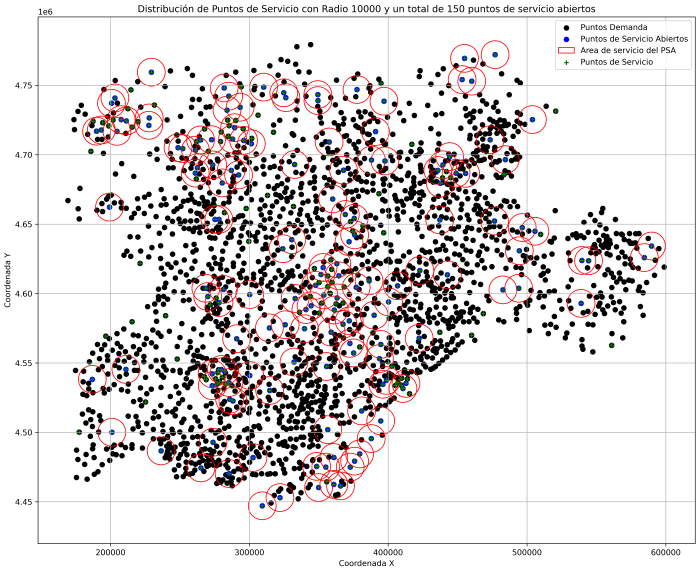

Figura 6.12: Solucion de compromiso BACOP2 con el método de ε

6.2.2. Resultados obtenidos con el método de NISE

Lo primero que vemos es una gráfica correspondiente a las soluciones de la frontera de Pareto obtenidas con el método de NISE.

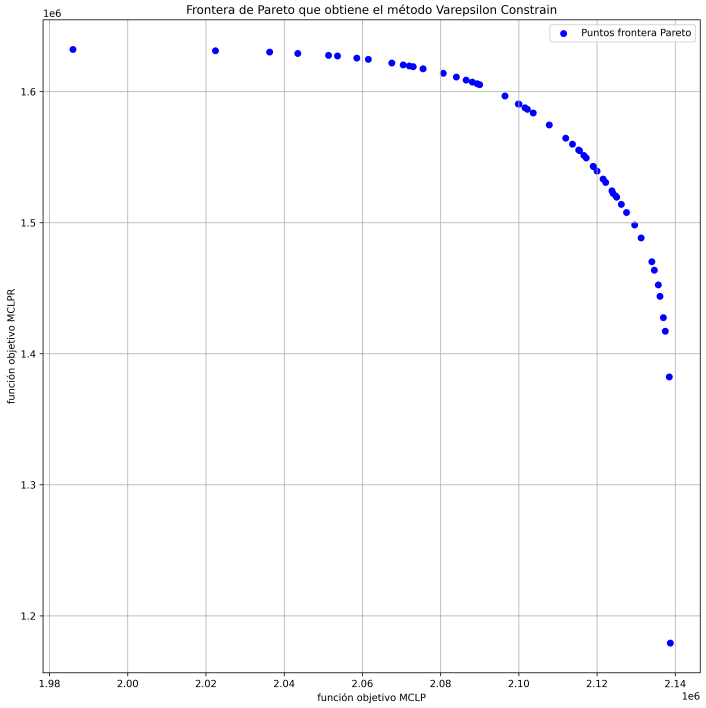

Figura 6.13: Frontera de Pareto obtenida con el método de NISE

A continuación se muestra una tabla con los resultados obtenidos para el conjunto de Castilla y León con el método NISE.

Id	MCLP	MCLPR
1	2138741	1179194
2	2138462	1382367
3	2137448	1417167
4	2136961	1427537
5	2136074	1443821
6	2135664	1452485
...	...	...
49	2058554	1625650
50	2053622	1627212
51	2051346	1627707
52	2043478	1629164
53	2036272	1630172
54	2022438	1631238

Tabla 6.4: Resultados de MCLP y MCLPR para el conjunto de Castilla y León con el método nise

Finalmente tenemos una gráfica 6.14 que muestra la mejor solución obtenida para el primero de los objetivos (MCLP 4.1), otra gráfica 6.15 que muestra la grafica de la mejor solución obtenida para el segundo de los objetivos (MCLPR 4.2) y una última gráfica 6.16 que muestra una solución intermedia de compromiso entre ambos objetivos.

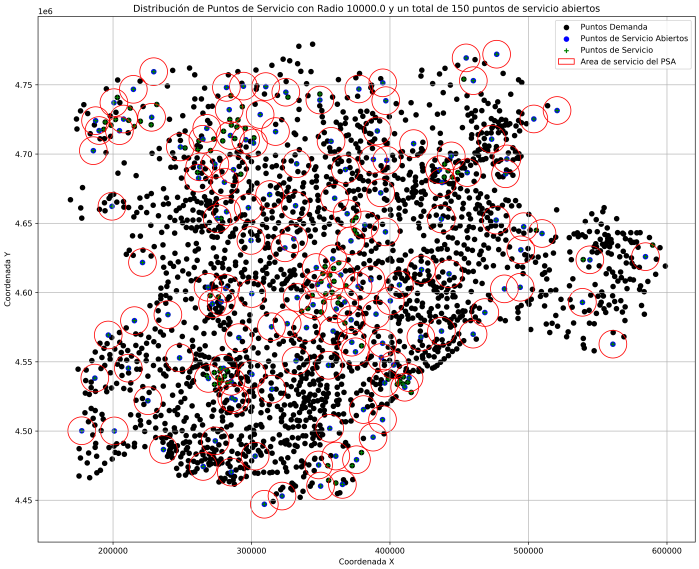

Figura 6.14: Mejor solución para el MCLP con el método de NISE

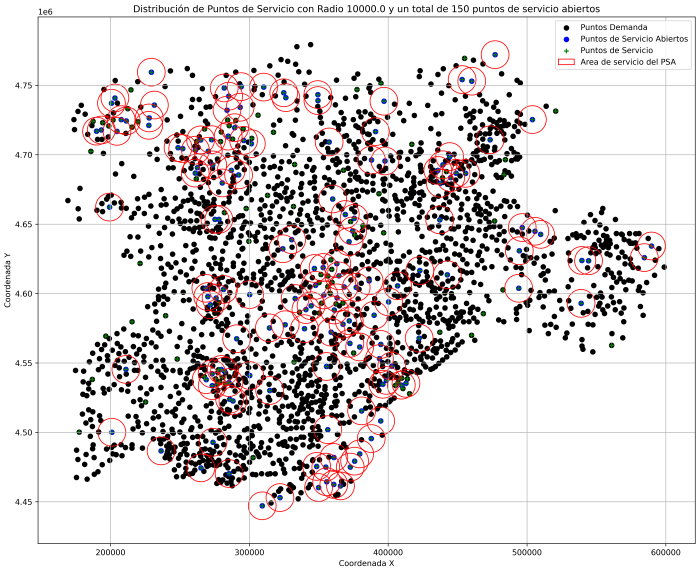

Figura 6.15: Mejor solución para el MCLPR con el método de NISE

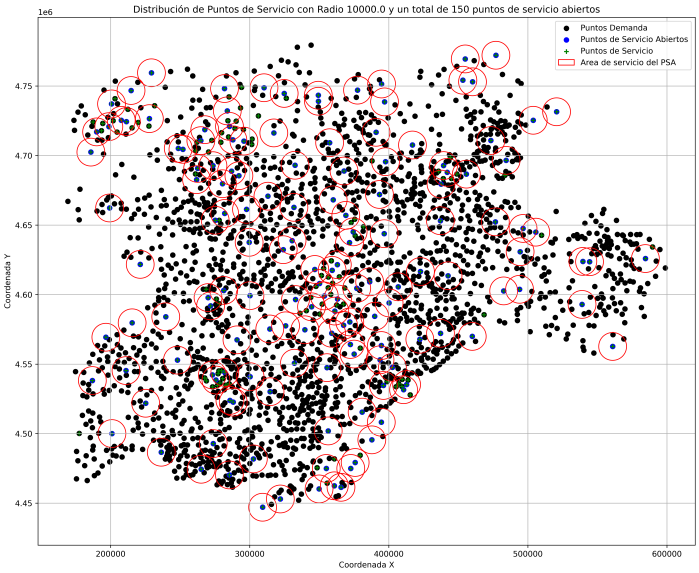

Figura 6.16: Solucion de compromiso BACOP2 con el método de NISE

6.3. Conjunto de pruebas con datos reales de toda España

Este conjunto de pruebas representa de manera realista cada uno de los municipios de España, por tanto contamos con un total de 7974 puntos de demanda, para los puntos de servicio, consideraremos solo aquellos municipios con una población superior a los 8000 habitantes, lo que nos deja con unos 840 posibles puntos de servicio. El conjunto puede encontrarse en la siguiente dirección: A. Cabe destacar que el problema bi-objetivo que se resuelve en este caso es el BACOP2 4.3, siendo como ya se explicó en la sección 4.3 los objetivos a maximizar el MCLP 4.1 y el MCLPR 4.2. El número máximo de puntos de servicio que se pueden abrir es 150 y además se fija una distancia de cubrimimiento máxima de 200 unidades métricas. En este problema, para ambos objetivos la demanda de cada punto de demannda es 1, y por tanto lo que se busca es maximizar el número de puntos de demanda que quedan cubiertos por al menos un punto de servicio (MCLP) y al menos dos veces (MCLPR).

6.3.1. Resultados obtenidos con el método de ε-constraint

Lo primero que vemos es una gráfica 6.17 correspondiente a las soluciones de la frontera de Pareto obtenidas con el método de ε-constraint.

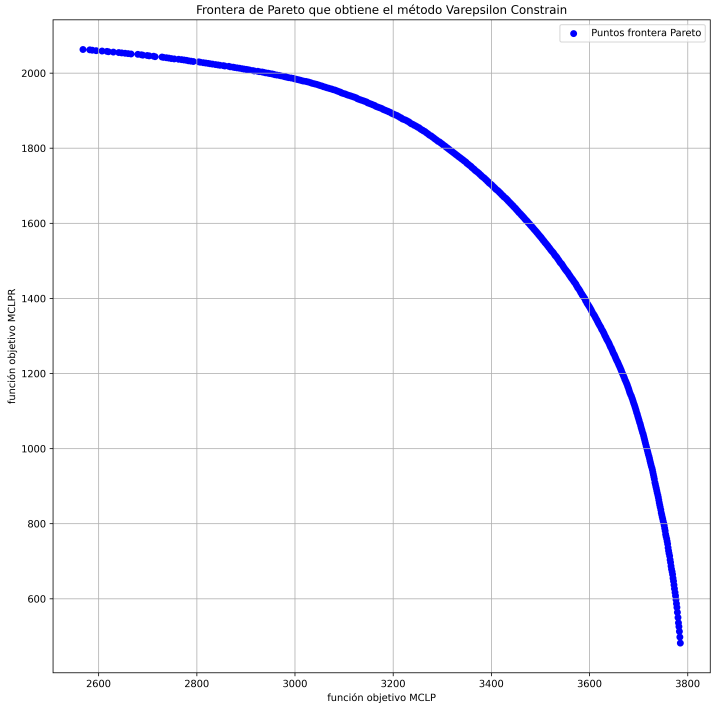

Figura 6.17: Frontera de Pareto obtenida con el método de ε-constraint

A continuación se muestra una tabla con los resultados obtenidos para el conjunto de España con el método de ε-constraint.

Id	**MCLP**	**MCLPR**
1	3785	482
2	3784	498
3	3783	513
4	3782	526
5	3781	536
6	3780	550
...	...	...
732	2620	2057
733	2617	2058
734	2607	2059
735	2595	2060
736	2587	2061
737	2582	2062

Tabla 6.5: Resultados de MCLP y MCLPR con el método varepsilon constraint para conjunto de España

Finalmente tenemos una gráfica 6.18 que muestra la mejor solución obtenida para el primero de los objetivos (MCLP 4.1), otra gráfica 6.19 que muestra la mejor solución obtenida para el segundo de los objetivos (MCLPR 4.2) y una última gráfica 6.20 que muestra una solución intermedia de compromiso entre ambos objetivos.

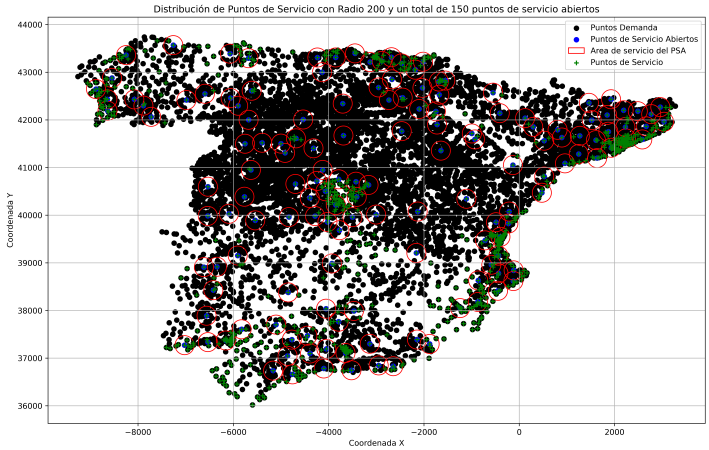

Figura 6.18: Mejor solución para el MCLP con el método de ε

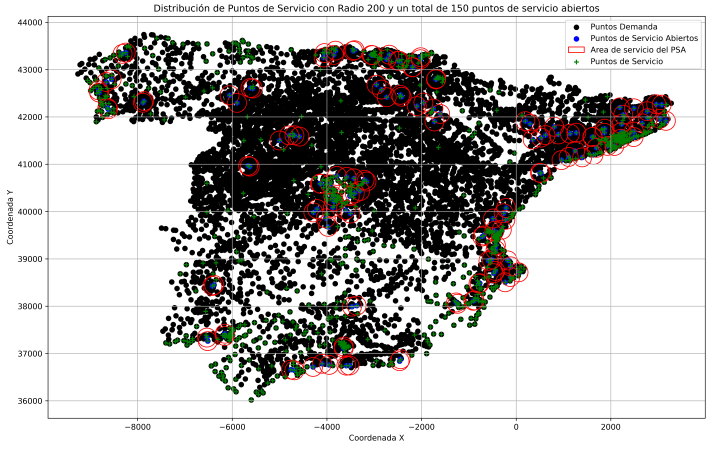

Figura 6.19: Mejor solución para el MCLPR con el método de ε

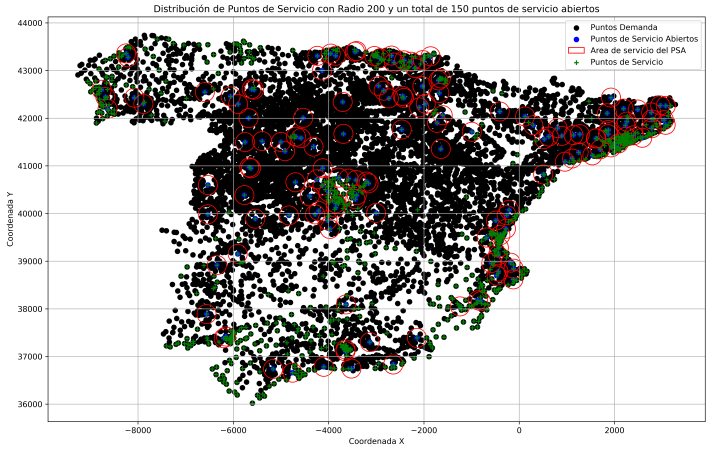

Figura 6.20: Solucion de compromiso BACOP2 con el método de ε

6.3.2. Resultados obtenidos con el método de NISE

Lo primero que vemos es una gráfica correspondiente a las soluciones de la frontera de Pareto obtenidas con el método de NISE.

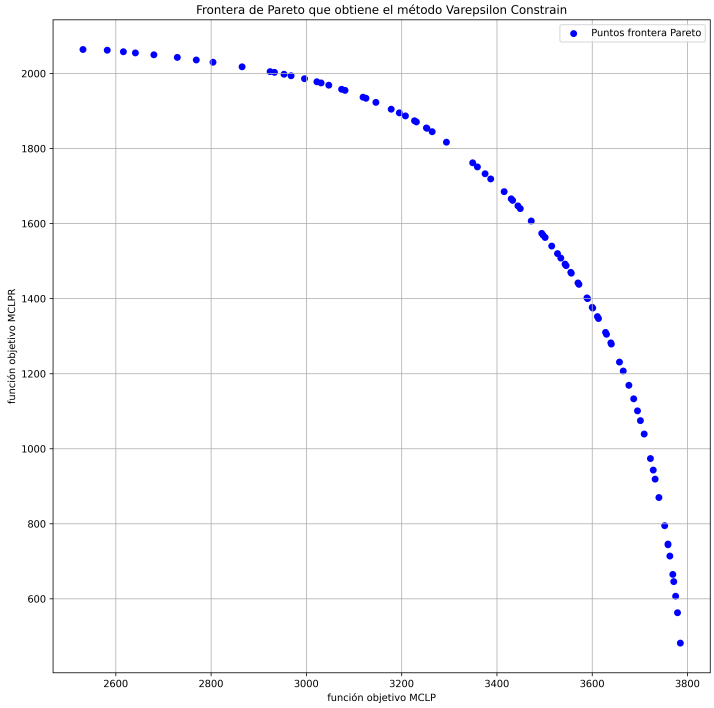

Figura 6.21: Frontera de Pareto obtenida con el método de NISE

A continuación se muestra una tabla con los resultados obtenidos para el conjunto de municipios de España con el método NISE.

Id	MCLP	MCLPR
1	3785	482
2	3779	563
3	3775	607
4	3771	646
5	3769	665
6	3763	714
...	...	...
81	2769	2036
82	2729	2043
83	2680	2050
84	2641	2055
85	2616	2058
86	2582	2062

Tabla 6.6: Resultados de MCLP y MCLPR con el método varepsilon constraint para conjunto de municipios de España

Finalmente tenemos una gráfica 6.22 que muestra la mejor solución obtenida para el primero de los objetivos (MCLP 4.1), otra gráfica 6.23 que muestra la grafica de la mejor solución obtenida para el segundo de los objetivos (MCLPR 4.2) y una última gráfica 6.24 que muestra una solución intermedia de compromiso entre ambos objetivos.

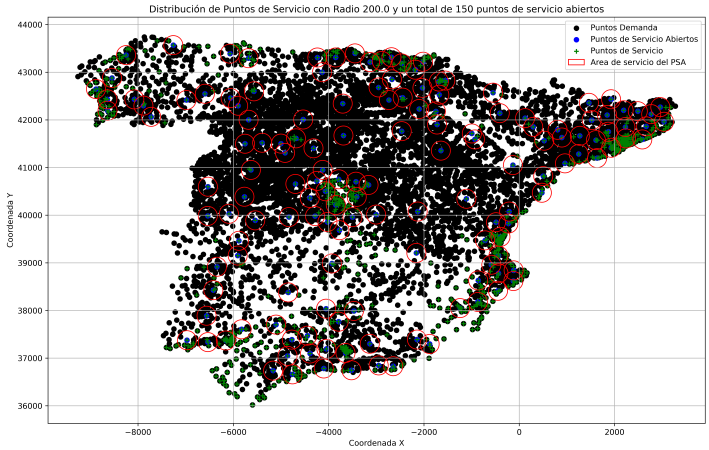

Figura 6.22: Mejor solución para el MCLP con el método de NISE

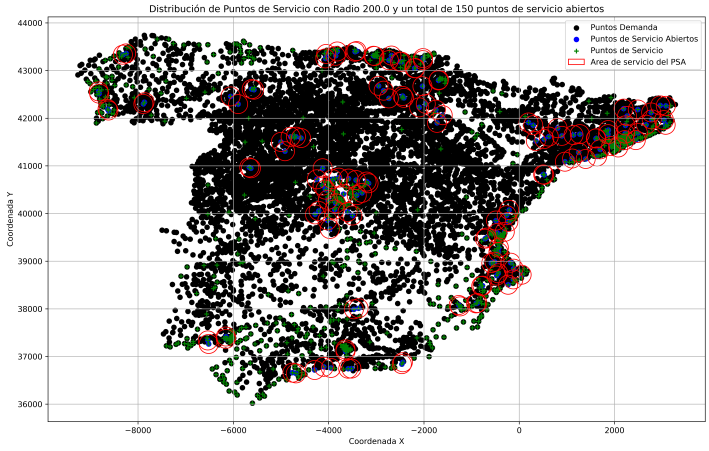

Figura 6.23: Mejor solución para el MCLPR con el método de NISE

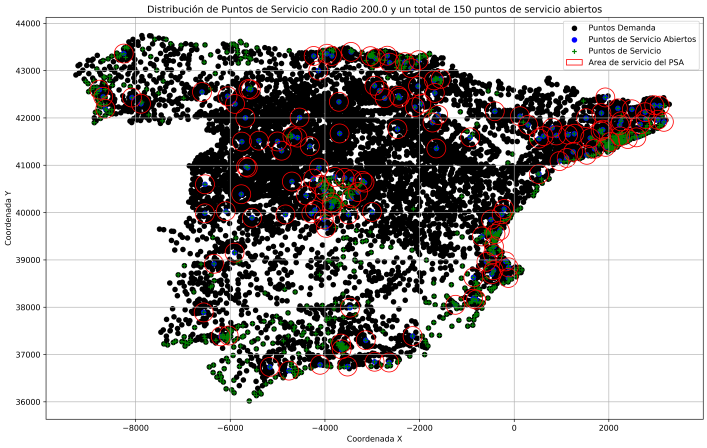

Figura 6.24: Solucion de compromiso BACOP2 con el método de NISE

Capítulo 7

Conclusiones

7.1. Comparación de los métodos:

En cuanto a los tiempos de ejecución, el método NISE es claramente más rápido que el método ε-constraint, y estas diferencia se aprecia ya, auque en menor medida, incluso para conjuntos de datos pequeños (pocos puntos demanda y / o servicio). Aunque esta diferencia sea muy evidente, cabe añadir que simplemente por construcción, como vimos en la seccion 3.1.5, era de esperar, ya que goza de una ventaja algorítmica al no buscar las soluciones óptimas en función de un criterio más logrado que simplemente recorrer todo el espacio de soluciones de un criterio optimizando el otro.

Sin embargo, esta ventaja en el tiempo de ejecución no deja de ser un arma de doble filo, puesto que tienes que controlar el porcentaje de confinaza que impones para desechar la búsqueda de soluciones entre las soluciones de cada iteración. Poner muy alto este valor haría al algoritmo muy rápido pero al mimso tiempo, la precisión de las soluciones sería más baja, lo que con seguridad en la mayoría de los casos nos podría alejar de buena solución. Sin embargo, cabe destacer, tras las pruebas realizadas, que por muy alto que pongas este porcentaje de confianza, el método NISE siempre será significativamente más rápido que el método ε-constraint.

Por otra parte, el método NISE no es más que una mejora sobre el método de Sumas Ponderadas 3.1, y al pertenecer a esta familia, tiene sus mismas limitaciones vistas en la sección 3.1.6, lo que nos hace no poder encontrar toda la frontera de Pareto por mucho que disminuyamos el porcentaje anteriomente comentado. Por tanto con el método NISE no podremos tampoco encontrar las conocidas como soluciones no soportadas, algo que, cabe destacar que no ocurre con el método varepsilon constraint, el cual si que encuentra todas las soluciones eficientes de la frontera de Pareto.

Me gustaría remarcar, que para ambos métodos sería posible aplicar programación concurrete, es decir paralelizar el flujo de ejecución, ya que en ambos métodos se puede dividir por secciones y ejecutar en varios ordenadores a la vez o bien en un mismo ordenador o incluso un servidor. Aunque sea válido para ambo métodos, el método considero que el ε- constraint se vería significativamente más beneficiado de esta técnica, no solo porque sea más fácil de dividir el espectro de búsqueda para cada dispositivo en paralelo al ser indepdiente los propios espectros. Sino que también porque, con NISE puede que sólo uno de los dispositivos acabara haciendo todo el trabajo mientras que el resto terminar de ejecutar mucho antes, y aunque esto se puede programar para volver a dividir la carga, necesitaria constantemente comunicación entre los dispositivos.

7.1.1. Opinión personal

No considero que haya un método mejor que otro, y como todo en la vida, depende de la situación en la que te encuentres. Si lo único que quieres es saber en que valores ronda las soluciones, utiliza el método NISE, pero si necesitas mayor precisión y la diferencia entre una solución aceptable y una muy buena puede ser significativa o trascendental entonces te propongo dos opciones:

- En caso de no necesitar la solución de manera extremadamente rápida o bien que tu conjunto de datos no sea excesivamente grande, entonces sin duda utiliza el método ε-constraint.
- Para caso en los que el tiempo sea clave, simplemente miremos que hacen ambos métodos, ninguno rechaza un objetivo y se queda con el otro. sino que buscan un compromiso de ambos. Por tanto, haz lo propio y utiliza el método NISE para buscar entre que soluciones te gustaría explorar más a fondo y establece los parámetros de búsqueda del método ε-constraint para que únicamente busque en un rango pequeño. De esta manera, enconrarás todas las solucioens eficinete para esa parte que te interesa de la frontera de Pareto.

7.2. Líneas de trabajo futuras

- Como ya se ha comentado en la comparación de métodos, sería interesante implementar la programación concurrente o paralela.
- De igual forma, también pudiera ser interesante pensar en metaherísticas que no utilizaran la solución óptima de un solver, si bien esto tendría que tenerse en cuenta con una reformulación de ambos métodos, ya que al no estar seguros de llegar una solución optima.
- Sin duda alguna, otra posiblidad sería en pensar en metahurísticas para el problema en su conjunto y no solo para calcular una aproximación de la solución óptima uni-objetivo.
- Por último, sería interesante implementar los métodos restantes de la sección 3.3.

Bibliografía

[1] Vira Chankong and Yacov Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. North-Holland, New York, 1983.

[2] Chen-Hua Chung. Recent applications of the maximal covering location planning (m.c.l.p.) model. *Journal of the Operational Research Society*, 37(8):735–746, 1986.

[3] Jared L. Cohon. *Multiobjective Programming and Planning*, volume 140 of *Mathematics in Science and Engineering*. Academic Press, Orlando, FL, USA, 1985.

[4] Matthias Ehrgott. *Multicriteria Optimization*. Springer, Berlin, Heidelberg, 2nd edition, 2005.

[5] Alfred W. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22:618–630, 1968.

[6] Kathleen Hogan and Charles ReVelle. Concepts and applications of backup coverage. *Management Science*, 32(11):1434–1444, 1986.

[7] Kaisa Miettinen. *Nonlinear Multiobjective Optimization*, volume 12 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Boston, 1999.

[8] Wilfredo Pareto. *Manual of Political Economy*. Macmillan, London, 1906.

[9] Jesús Sáez-Aguado and Paula Camelia Trandafir. Variants of the varepsilon-constraint method for biobjective integer programming problems: application to p-median-cover problems. *Mathematical Methods of Operations Research*, 2017.

Apéndice A

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código y datos: `https://gitlab.inf.uva.es/tomcarr/codigo-tfg-estadistica.git`.

No se si dejar solo el enlace al Repositorio o si añadir también aquí el propio código, rollo:

A.1. Código del algoritmo NISE

```
model NISE
uses "mmxprs";
uses "mmsystem"
uses "mmive"


declarations
    pdemanda= 1..m
    pservicio = 1..n
    cxS, cyS:array(pservicio)of real
    cx, cy: array(pdemanda) of real
    dist:array(pdemanda,pservicio)of real
    !nc:array(pdemanda)of integer
    dem:array(pdemanda)of real
    a:array(pdemanda,pservicio)of integer
    dc=r
    tam = 1000
    Pn : array(1..tam, 1..2) of real
    Sn : array(1..tam, 1..2) of real
    Snn : array(1..tam, 1..2) of real
    Phis : array(1..tam) of real
    Phisn : array(1..tam) of real
    y: array(pdemanda) of mpvar
    u: array(pdemanda) of mpvar
    x: array(pservicio) of mpvar
    psAbiertos : array(1..tam, 1..p) of integer
end-declarations


(!
    Funcion que dada la longitud de 3 lados de un triangulo, devuelve su
        area
!)
function area(a: real, b: real, c: real): real
```

```
declarations
s: real
resultado: real
end-declarations
s := (a + b + c) / 2
resultado := sqrt(s * (s - a) * (s - b) * (s - c))
returned := abs(resultado)
end-function


(!
    Funcion que devuvelve la distancia entre dos puntos bidimensionales
        dados como sigue P1x, P1y, P2x, P2y
!)
function distEucl(x1: real, yp1: real, x2: real, y2: real): real
declarations
    dist : real
end-declarations
dist :=  sqrt((x2 - x1)^2 + (y2 - yp1)^2)
returned := abs(dist)
end-function



(!
        Comienzo maximizando el primer criterio (MCLP)
!)
!setparam("XPRS_MAXTIME",-100)
!obj := sum(j in pdemanda) y(j)
obj := sum(j in pdemanda) dem(j)*y(j)

forall(i in pdemanda)
    res1(i):= sum(j in pservicio)x(j)*a(i,j)>= y(i)

forall(i in pdemanda)
    res2(i):= sum(j in pservicio)x(j)*a(i,j) >= 2*u(i)

res3 := sum(j in pservicio) x(j) = p

forall(i in pdemanda) y(i) is_binary
forall(j in pservicio) x(j) is_binary
forall(i in pdemanda) u(i) is_binary

maximize(obj)
Pn(1,1) := getobjval

! Busco para el otro criterio (MCLPR) la solucion optima, si pongo como
    restriccion que el MCLP sea el maximo anteiror
!obj2 := sum(i in pdemanda) u(i)
obj2 := sum(j in pdemanda) dem(j)*u(j)

resEliminar := sum(i in pdemanda) y(i)*dem(i) >= Pn(1,1)

maximize(obj2)
Pn(1,2) := getobjval
writeln("")
writeln("Max␣MCLP␣=␣", Pn(1,1), "\t␣con␣MCLPR␣=␣", Pn(1,2))
! guardo los puntos abiertos para esta solcuion de la frontera de pareto
indR := 1
forall(indiceRellenar in pservicio | x(indiceRellenar).sol = 1) do
    psAbiertos(1, indR) := indiceRellenar
    indR := indR +1
end-do
(!
    Fin maximizar el primer criterio (MCLP)
!)
```

```
(!
    Paso a maximizar el segudno criterio (MCLPR)
!)

!obj := sum(j in pdemanda) u(j)
obj := sum(j in pdemanda) dem(j)*u(j)

resEliminar := sum(i in pdemanda) u(i) >= 0

maximize(obj)

Pn(2,2) := getobjval
! Busco para el otro criterio (MCLP) la solucion optima, si pongo como
   restriccion que el MCLPR sea el maximo anteiror
!obj2 := sum(i in pdemanda) y(i)
obj2 := sum(j in pdemanda) dem(j)*y(j)
resEliminar := sum(i in pdemanda) dem(i)*u(i) >= Pn(2,2)

maximize(obj2)
Pn(2,1) := getobjval
writeln("")
writeln("MCLP␣=␣", Pn(2,1), "\t␣correspondiente␣a␣Max␣MCLPR␣=␣", Pn(2,2))
indR := 1
forall(indiceRellenar in pservicio | x(indiceRellenar).sol = 1) do
    psAbiertos(2, indR) := indiceRellenar
    indR := indR +1
end-do
(!
    Fin de maximizar el segundo criterio
!)



! Limpio las restricciones de tipo Eliminar
resEliminar := 0



(!
    Guardo las dos primeras solcuiones de pareto previamente obtenidas de
       manera ordenada en S,
    asi como calculo el punto intermedio del criterio NISE y pongo n = 2 (2
       soluciones)
!)
imax := 2
Sn(1,1) := Pn(2,1)
Sn(1,2) := Pn(2,2)
Sn(2,1) := Pn(1,1)
Sn(2,2) := Pn(1,2)
interX := Pn(1,1)
interY := Pn(2,2)
n := 2

! Calculo la distiancia entre las dos soluciones que tenemos
distP1P2 := distEucl(       Pn(1,1),
                            Pn(1,2),
                            Pn(2,1),
                            Pn(2,2)     )

! Calculo el valor del criterio NISE para las dos soluciones
Phis(1) := (area(   (interX-Pn(2,1)),
                    (interY-Pn(1,2)),
                    distP1P2                )*2)/distP1P2
```

```
Phi := Phis(1)

! Establezco el error maximo permitido, criterio de finalizacion del bucle.
T := Phi *0.008
!T := Phi *0.01
writeln(T)


(!
    Se itera hasta que todas las soluciones ordenadas consecutivas tienen un
        criterio menor que T,
    por lo que en cada iteracion se podra annadir una nueva solucion de la
        frontera de pareto de Pareto.
!)
while(Phi <> 0) do
    ! Obtego el indice el Phi maximo porque de esas dos soluciones sale la
        proxima intermedia.
    maximo := 0.0
    forall(iden in 1..tam)do
        if(Phis(iden) >= maximo)then
            imax := iden
            maximo := Phis(iden)
        end-if
    end-do
    ! Solucion optima NISE para el imax anterior
    !obj := abs(Sn(imax,2) - Sn((imax+1),2))*(sum(j in pdemanda)y(j)) + abs(
        Sn((imax+1),1)-Sn(imax,1))*(sum(j in pdemanda)u(j))
    obj := abs(Sn(imax,2) - Sn((imax+1),2))*(sum(j in pdemanda)dem(j)*y(j))
        + abs(Sn((imax+1),1)-Sn(imax,1))*(sum(j in pdemanda)dem(j)*u(j))
    forall(i in pdemanda)
        res1(i):= sum(j in pservicio)x(j)*a(i,j) >= 2*u(i)

    forall(i in pdemanda)
        res2(i):= sum(j in pservicio)x(j)*a(i,j) >= y(i)

    res3 := sum(j in pservicio) x(j) = p

    forall(i in pdemanda) u(i) is_binary
    forall(i in pdemanda) y(i) is_binary
    forall(j in pservicio) x(j) is_binary

    maximize(obj)

    ! Obtengo el valor de la solucion optima para MCLP y MCLPR
    Syi := 0
    forall(i in pdemanda) do
        iter :=0
        forall(j in pservicio | a(i,j) = 1 and x(j).sol > 0.999)do
            iter := iter + 1
        end-do
        if(iter >= 2)then
            Syi := Syi + floor(dem(i))
        end-if
    end-do

    Sxi := 0
    forall(ii in pdemanda) do
        iter :=0
        forall(j in pservicio | a(ii,j) = 1 and x(j).sol > 0.999)do
            iter := iter + 1
        end-do
        if(iter >= 1)then
            Sxi := Sxi + floor(dem(ii))
        end-if
    end-do
```

```
! Calculo B, si mayor que la sol optima anteiror entonces entre estos
    puntos que nos da imax, no se pueden encontrar mas soluciones del
    conjutno de pareto, de otro modo se annade la solucion de esta
    iteracion
B := abs(Sn(imax,2) - Sn((imax+1),2))*Sn(imax,1)
B := B + abs(Sn((imax+1),1)-Sn(imax,1))*Sn(imax,2) + 0.1 ! el mas una
    decima, hay que ponerlo porque si es igual se mete aunque no debiera
    , eso es culpa de XPRESS no mia, en algunos PC va bien otros no.
!writeln("\t", Sn((imax),1),"\t", Sn((imax),2),"\t", Sn((imax+1),1),"\t
    ", Sn((imax+1),2))
writeln(getobjval,  "\t", Sxi,  "\t", Syi,  "\t B =",B)

if( ( Sn((imax),1) = Sn((imax+1),1) and Sn((imax),2) = Sn((imax+1),2) )
    or ( Sn((imax),1) = Sxi and Sn((imax),2) = Syi ) or (  Sxi = Sn((
    imax+1),1) and Syi = Sn((imax+1),2)  )   )then
    !writeln("Es por esto")
    Phis(imax) := 0
else if( getobjval > B ) then
    ! incrementamos el numero de soluciones en la frontera de pareto de
       pareto
    n := n + 1

    Pn(n,1) := Sxi
    Pn(n, 2) := Syi


    ! para la solucion nueva guardamos que puntos de servicio abre
    indR := 1
    forall(indiceRellenar in pservicio | x(indiceRellenar).sol = 1) do
        psAbiertos(n, indR) := indiceRellenar
        indR := indR +1
    end-do

    ! Actualizamos segun corresponde los vectores Phi y matriz S
    iterador := 1
    while(iterador <= n)do
        if(iterador <= imax)then
            Snn(iterador, 1) :=          Sn(iterador, 1)
            Snn(iterador, 2) :=          Sn(iterador, 2)
        elif(iterador = (imax+1)) then
            Snn(iterador, 1) :=          Sxi
            Snn(iterador, 2) :=          Syi
        else
            Snn(iterador, 1) :=          Sn((iterador-1), 1)
            Snn(iterador, 2) :=          Sn((iterador-1), 2)
        end-if
        iterador := iterador + 1
    end-do

    Sn := Snn

    iterador := 1
    while(iterador < (n))do
        if(iterador < imax)then
            Phisn(iterador) :=  Phis(iterador)

        elif(iterador = imax) then
            interX := Sn((imax+1),1)
            interY := Sn(imax,2)

            distSiSimas1 := distEucl(   Sn((imax),1),
                                        Sn((imax),2),
                                        Sn((imax+1),1),
                                        Sn((imax+1),2))
```

```
                Phisn(iterador) := area((interX-Sn((imax),1)),
                                        (interY-Sn((imax+1),2)),
                                        distSiSimas1)*2/distSiSimas1

            elif(iterador = (imax+1)) then
                interX := Sn((imax+2),1)
                interY := Sn((imax+1),2)

                distSiSimas2 := distEucl(   Sn((imax+1),1),
                                            Sn((imax+1),2),
                                            Sn((imax+2),1),
                                            Sn((imax+2),2))

                Phisn(iterador) := area((interX-Sn((imax+1),1)),
                                        (interY-Sn((imax+2),2)),
                                        distSiSimas2 )*2/distSiSimas2

            else
                Phisn(iterador) :=  Phis(iterador-1)
            end-if
            iterador := iterador + 1
        end-do

        Phis := Phisn


    else
        Phis(imax) := 0
    end-if
    end-if

    ! Se presupone que se termina en todas las iteraciones, pero con que
    !       un phi sea mayor que T, se sigue buscando soluciones
    Phi := 0
    forall(idenTerminar in 1..(n-1)) do
        if(Phis(idenTerminar) > T)then
            Phi := 1
        end-if
    end-do
    !writeln("la lista de phis queda: \t", Phis)
    !forall( ind2 in 1..n)do
    !       writeln(Sn(ind2, 1),", ", Sn(ind2, 2))
    !end-do
end-do


! Dibujar los pintos de servicio y las asignaciones
id1:=IVEaddplot("puntos", IVE_RED)
id2:=IVEaddplot("lineas", IVE_BLACK)
id3:= IVEaddplot("mejores", IVE_BLUE)
!forall(i in pdemanda) IVEdrawpoint(id1, cx(i), cy(i))! Dibujamos los puntos
!forall(i in pdemanda, j in pservicio | y(i).sol > 0.999 and x(j).sol >
    0.999 and a(i,j) = 1 ) IVEdrawline(id2, cx(i), cy(i), cx(j), cy(j)) !
    Dibujamos las lineas


forall(i in 1..n)do
    IVEdrawpoint(id3, Pn(i,1), Pn(i,2))
end-do
!IVEdrawpoint(id3, Pn(1,1), Pn(1,2))
!IVEdrawpoint(id3, Pn(2,1), Pn(2,2))
IVEdrawpoint(id1, 0, 0)

writeln("\n\n")
```

```
    writeln("El␣conjunto␣de␣soluciones␣factibles␣es␣\n")

    forall( ind2 in 1..n)do
        writeln(Pn(ind2, 1),",␣", Pn(ind2, 2))
        !fprintln(f, Pn(ind2, 1),", ", Pn(ind2, 2))
    end-do

    writeln("\n\nLos␣puntoes␣de␣servicio␣que␣se␣habren␣son:␣\n")


    forall( indImprimir in 1..n)do
        forall( ind2 in 1..p) do
        write( psAbiertos(indImprimir,ind2), ",␣")
        !fprint(f, psAbiertos(indImprimir,ind2), ", ")
        end-do
        writeln("␣")
        !fprintln(" ")
    end-do


    !fichero de escritura de las soluciones:
    !fopen("solucionNISE_large1.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    !fopen("solucionNISE_cyl.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    !fopen("solucionNISE_esp.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    fopen("solucionNISE_esp_Borrar.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    writeln(cx)
    writeln(cy)
    writeln(cxS)
    writeln(cyS)
    writeln(n)
    writeln(r)
    writeln(p)

    forall( ind2 in 1..n)do
        writeln(Pn(ind2, 1),",␣", Pn(ind2, 2))
    end-do


    forall( indImprimir in 1..n)do
        forall( ind2 in 1..p) do
        write( psAbiertos(indImprimir,ind2), "␣")
        end-do
        writeln("␣")
    end-do

    fclose(F_OUTPUT)  ! Cierra el archivo para guardar los cambios

    end-model
```

A.2. Código del algortimo ε-constraint

```
    model VCMaxMCLP
    uses "mmxprs";
    uses "mmsystem"
    uses "mmive"


    declarations
        pdemanda= 1..m
```

```
    pservicio = 1..n
    cxS, cyS:array(pservicio)of real
    cx, cy: array(pdemanda) of real
    dist:array(pdemanda,pservicio)of real
    !nc:array(pdemanda)of integer
    dem:array(pdemanda)of real
    a:array(pdemanda,pservicio)of integer
    dc=r
    tam = 1000
    Pn : array(1..tam, 1..2) of real

    y: array(pdemanda) of mpvar
    u: array(pdemanda) of mpvar
    x: array(pservicio) of mpvar
    psAbiertos : array(1..tam, 1..p) of integer
    !p=3
end-declarations


(!
    Comienzo maximizando el primer criterio (MCLPR)
!)
!setparam("XPRS_MAXTIME",-100)
!obj := sum(j in pdemanda) u(j)
obj := sum(j in pdemanda) dem(j)*u(j)

forall(i in pdemanda) res1(i):= sum(j in pservicio)x(j)*a(i,j) >= 2*u(i)
forall(i in pdemanda) res2(i):= sum(j in pservicio)x(j)*a(i,j)>= y(i)
res3 := sum(j in pservicio) x(j) = p

forall(i in pdemanda) u(i) is_binary
forall(j in pservicio) x(j) is_binary
forall(i in pdemanda) y(i) is_binary
maximize(obj)

varep := getobjval
writeln("La␣solucion␣optima␣es\t", varep)


n := 1

! Busco para el otro criterio (MCLP) la solucion optima, si pongo como
    restriccion que el MCLPR sea el maximo anteiror
!obj2 := sum(i in pdemanda) y(i)
obj2 := sum(j in pdemanda) dem(j)*y(j)
res4 := sum(i in pdemanda) u(i)*dem(i) >= varep

maximize(obj2)

Pn(n,2) := getobjval
Pn(n,1) := varep

writeln("La␣solucion␣del␣MCLP␣=␣", Pn(n,1), "\t␣corresponde␣a␣maximizar␣
    MCLPR␣a␣", Pn(n,2))
! guardo los puntos abiertos para esta solcuion de la frontera de pareto
indR := 1
forall(indiceRellenar in pservicio | x(indiceRellenar).sol = 1) do
    psAbiertos(1, indR) := indiceRellenar
    indR := indR +1
end-do
(!
    Fin maximizar el primer criterio (MCLPR)
!)


(!
```

```
        Busquemos el comienzo de varepIter para no hacer calculo redundante:
            Esto lo hacemos sacando el maximo para MCLP y su correspondiente
                optimo de Pareto en MCLPR
    !)

    res4 := 0 ! nos aseguramos de no arrastrar una restriccion previa
    maximize(obj2)

    res4:= sum(i in pdemanda) y(i)*dem(i)  >= getobjval

    maximize(obj)

    varepIter := getobjval
    (!
        Fin Busquemos el comienzo de varepIter
    !)


    (!
        Recorremos todo el espectro de entre varepIter y varep para obtener
            todas las soluciones de la frontera de pareto
    !)
    while(varepIter <= varep)do
        n := n+1

        ! Encontrar la solucion optima para MCLP imponiendo como restriccion que
             el MCLPR sea varepIter
        !obj := sum(j in pdemanda) y(j)
        obj := sum(j in pdemanda) dem(j)*y(j)

        res4 := sum(i in pdemanda) u(i)*dem(i)  >= varepIter

        forall(i in pdemanda) y(i) is_binary
        forall(j in pservicio) x(j) is_binary
        forall(i in pdemanda) u(i) is_binary
        maximize(obj)


        ! no queremos quedarnos con los puntos dominados
        if(  getobjval >= Pn(n-1,1)-0.1  ) then ! el menos una decima es
            necesario por culpa de que getobjval puede que sea un numero mal
            menos 10 a la menos el limite de XPRESS
            n := n - 1
            writeln("los␣datos␣del␣de␣anterior␣corresponden␣a␣un␣dominado")
        end-if
        !writeln("")
        !writeln("MCLP = ", getobjval, "\tMCLPR = ", varepIter, "-")

        ! COMIENZO de una cosa nueva para quitar iteraciones de dominados (que
            no se hagan)
        maxIterMCLP := getobjval
        res4 := sum(i in pdemanda) y(i)*dem(i)  >= maxIterMCLP
        !obj := sum(j in pdemanda) u(j)
        obj := sum(j in pdemanda) dem(j)*u(j)
        maximize(obj)

        ! FIN de una cosa nueva para quitar iteraciones de dominados (que no se
            hagan)

        if(getobjval > varepIter) then
            varepIter := getobjval
        end-if
        Pn(n,2) := varepIter
        Pn(n,1) := maxIterMCLP
        writeln("MCLP␣=␣", maxIterMCLP, "\tMCLPR␣=␣", varepIter,"␣getobjval␣",
            getobjval)
```

```
        indR := 1
        forall(indiceRellenar in pservicio | x(indiceRellenar).sol = 1) do
            psAbiertos(n, indR) := indiceRellenar
            indR := indR +1
        end-do
        ! incrementamos el valor de varepIter
        varepIter := varepIter + 1

    end-do


    ! Dibujar los pintos de servicio y las asignaciones
    id1:=IVEaddplot("puntos", IVE_RED)
    id2:=IVEaddplot("lineas", IVE_BLACK)
    id3:= IVEaddplot("mejores", IVE_BLUE)
    !forall(i in pdemanda) IVEdrawpoint(id1, cx(i), cy(i))! Dibujamos los puntos
    !forall(i in pdemanda, j in pservicio | y(i).sol > 0.999 and x(j).sol >
        0.999 and a(i,j) = 1 ) IVEdrawline(id2, cx(i), cy(i), cx(j), cy(j)) !
        Dibujamos las lineas


    forall(i in 1..n)do
        IVEdrawpoint(id3, Pn(i,1), Pn(i,2))
    end-do
    !IVEdrawpoint(id3, Pn(1,1), Pn(1,2))
    !IVEdrawpoint(id3, Pn(2,1), Pn(2,2))
    IVEdrawpoint(id1, 0, 0)

    writeln("\n\n")

    writeln("El␣conjunto␣de␣soluciones␣factibles␣es␣\n")

    forall( ind2 in 1..n) writeln(Pn(ind2, 1),",␣", Pn(ind2, 2))


    writeln("\n\nLos␣puntoes␣de␣servicio␣que␣se␣habren␣son:␣\n")

    forall( indImprimir in 1..n)do
        forall( ind2 in 1..p) do
        write( psAbiertos(indImprimir,ind2), ",␣")
        end-do
        writeln("␣")
    end-do




    !fichero de escritura de las soluciones:
    !fopen("solucionVarepMCLP_large1.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    !fopen("solucionVarepMCLP_cyl.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    !fopen("solucionVarepMCLP_esp4.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    !fopen("solucionVarepMCLP_cyl_v2.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    fopen("solucionVarepMCLP_Borrar.txt", F_OUTPUT)  ! Abre el archivo en modo
        escritura
    writeln(cx)
    writeln(cy)
    writeln(cxS)
    writeln(cyS)
    writeln(n)
    writeln(r)
```

```
    writeln(p)

    forall( ind2 in 1..n)do
        writeln(Pn(ind2, 1),",␣", Pn(ind2, 2))
    end-do


    forall( indImprimir in 1..n)do
        forall( ind2 in 1..p) do
        write( psAbiertos(indImprimir,ind2), "␣")
        end-do
        writeln("␣")
    end-do

    fclose(F_OUTPUT)  ! Cierra el archivo para guardar los cambios



    end-model
```