Formulación y resolución de modelos de optimización binivel para problemas de determinación de precios de múltiples productos

Aitor Hernández González

Trabajo de fin de máster
Máster universitario en Modelización e Investigación matemática, Estadística y Computación

Directores del trabajo:
Carmen Galé Pola y José Ángel Iranzo Sanz

2 de diciembre de 2022

Resumen

El problema de determinación de precios de múltiples productos es un problema de optimización clásico muy estudiado en la literatura que surge de la necesidad de establecer el precio de los productos que ofrece una empresa con el objetivo de maximizar los beneficios obtenidos por la venta de los productos atendiendo a las preferencias de los clientes. Son muchas las variantes de este problema. La versión clásica asume que cada cliente quiere adquirir una única unidad de un producto y que para ello dispone de un presupuesto fijo. La formulación de un modelo binivel considera de forma natural la reacción de los clientes a los precios establecidos. Para cada cliente se dispone de una lista ordenada de los productos según sus preferencias. Cuando existe la posibilidad de que un cliente prefiera dos o más productos de la misma manera se produce un empate. El tratamiento del problema de determinación de precios con empates en las preferencias desde una perspectiva binivel es novedoso en la literatura. El modelo binivel se reformula como un problema binario puro lineal de un solo nivel para su resolución. Además, se propone por primera vez en la literatura un algoritmo genético para resolver el problema cuando el número de clientes y productos aumenta y la resolución exacta es muy costosa en tiempo computacional.

Abstract

The rank pricing problem of multiple products is a classical optimization problem widely present in the literature. This problem arises from the need to establish the prices of products offered by a company with the objective of maximize its revenues and considering the preferences of the customers. There are a lot of variants of this problem. The classical one is that in which customers only want to purchase one unit of a certain product and for that purpose they have a fix budget. The bilevel reformulation considers the natural reaction of the customers to the established prices. For each customer a sort list of the products according to its preferences is available. When the possibility in which a customer prefers several products in the same way exists, a tie appears. The study of the rank pricing problem with ties in the preferences from a bilevel approach is relatively new in the literature. The bilevel model is reformulated as a single level pure binary model for its resolution. Furthermore, it is also proposed for the first time in literature a genetic algorithm to solve the problem when the number of customers and the nunber of products grow and the exact resolution is very expensive in computational time.

Índice general

Resumen III

Abstract V

1. Introducción 1
- 1.1. El problema de determinación de precios de múltiples productos . . . 1
- 1.2. Optimización binivel lineal . . . 3
- 1.3. El problema de determinación de precios y la optimización binivel . . . 6
- 1.4. Contribución de este TFM . . . 8

2. Formulación binivel para el problema con empates 11
- 2.1. Descripción del problema . . . 11
- 2.2. Formulación de un solo nivel para el *RPPT* en la literatura . . . 12
- 2.3. Formulación binivel para el *RPPT* . . . 13
- 2.4. Reformulación y tratamiento del problema binivel . . . 15
  - 2.4.1. Reformulación como un problema binivel binario puro . . . 15
  - 2.4.2. Reformulación como un problema de un solo nivel . . . 17
  - 2.4.3. Búsqueda de cotas válidas M_1 y M_2 . . . 22
  - 2.4.4. Linealización del problema . . . 24

3. Un algoritmo evolutivo para la resolución del problema con empates 27
- 3.1. Los algoritmos evolutivos . . . 27
- 3.2. Descripción del algoritmo genético propuesto . . . 30
  - 3.2.1. Definición de los cromosomas . . . 30
  - 3.2.2. Definición de la función *fitness* . . . 30
  - 3.2.3. Selección de padres para el cruce . . . 31
  - 3.2.4. El operador de cruce . . . 31
  - 3.2.5. El operador de mutación . . . 32
  - 3.2.6. Selección de supervivientes . . . 33
  - 3.2.7. Algoritmo genético completo . . . 33

4. Experiencia computacional 35
- 4.1. Descripción del conjunto de problemas de prueba . . . 35
- 4.2. Comparación de la resolución exacta del modelo de un solo nivel y el algoritmo genético 36
- 4.3. Evaluación de varias configuraciones del algoritmo genético . . . 40
- 4.4. Comportamiento del algoritmo genético en problemas de mayor tamaño . . . 46

5. Conclusiones y trabajo futuro 49

Bibliografía 51

Anexos **53**

A. Código R para el algoritmo genético **55**

B. Código R de algunas funciones auxiliares **61**

C. Código CPLEX para la resolución de los problemas **67**

D. Código R para la ejecución del algoritmo genético **71**

Capítulo 1

Introducción

1.1. El problema de determinación de precios de múltiples productos

El problema de determinación de precios (en inglés *Rank Pricing Problem* o *RPP* de forma abreviada), es un problema de optimización que surge de la necesidad de desarrollar estrategias por parte de empresas o entidades para determinar el precio de los productos o servicios que ofrecen a un conjunto de clientes teniendo en cuenta su comportamiento. El objetivo en un problema de determinación de precios consiste en maximizar los beneficios obtenidos por la venta de los productos atendiendo a que los clientes vean satisfechas sus necesidades, en la medida de lo posible, adquiriendo los productos que desean. En resumen, "vender el producto correcto al cliente correcto y por el precio correcto" [Calvete et al., 2019].

La versión clásica del problema de determinación de precios asume una situación en la que cada cliente está interesado en adquirir una única unidad de un producto determinado. Para ello, dispone de un presupuesto fijo con el que realizar la compra y sabe de antemano en qué productos está interesado y con qué orden de preferencia, sin la posibilidad de estar interesado por dos productos de igual manera. Además, la disponibilidad de los productos es ilimitada. Esta hipótesis tiene sentido si la empresa dispone de tantas unidades de cada producto como número de clientes o si los productos pueden ser producidos rápidamente de manera que no haya falta de existencias.

A la hora de determinar los precios de los productos ofertados se deben tener en cuenta las siguientes consideraciones. Por un lado, establecer precios muy bajos provoca que la empresa obtenga ganancias más bajas. Si los clientes están dispuestos a pagar más por esos mismos productos, la empresa estaría perdiendo beneficios. Sin embargo, si los productos son asequibles para una mayor cantidad de clientes, aumentan las ventas y además se cubren las necesidades de un mayor número de clientes. Por otro lado, establecer precios muy elevados produce mayores ganancias para las empresas si los clientes con alto poder adquisitivo adquieren los productos. También puede ocurrir que esas ganancias no se obtengan realmente si los clientes rechazan realizar alguna compra a causa de los precios altos, lo que supondría que ni las empresas tendrían beneficios ni los clientes verían cubiertas sus necesidades.

En la tabla 1.1 se muestra la matriz de preferencias y el vector de presupuestos de un caso concreto de problema de determinación de precios con 10 clientes y 5 productos extraído de [Calvete et al., 2019]. Si un producto i es el más preferido por un cliente k, este le asigna un valor de preferencia 5. Si otro producto j es el segundo producto más preferido por ese mismo cliente k le asigna un valor de preferencia 4 y así sucesivamente, contando con que existe la posibilidad de que algunos clientes no estén interesados en algunos productos y a los que no les asignan ningún valor de preferencia.

En verde y azul se muestran dos soluciones factibles del problema. En la solución marcada en verde, se apuesta por establecer precios altos para los productos. Con esta elección de precios solo los clientes con un presupuesto igual o mayor a 35, esto es, los cuatro primeros, pueden adquirir algún producto. Todos ellos adquieren su producto más preferido. Los clientes C_1 y C_2 el producto P_3 por 40 u.m.; el cliente C_3 el producto P_1 por 36 u.m.; y el C_4 el producto P_5 por 35 u.m., siendo el beneficio obtenido por las ventas de 151 u.m.

En la solución marcada en azul se apuesta por establecer precios bajos a los productos. Con esta

elección de precios todos los clientes se pueden permitir adquirir el producto más preferido. El rango de precios va desde el producto más barato, el producto P_2, de 13 u.m. que es adquirido solo por el cliente C_{10}, hasta el más caro, el producto P_3 de 25 u.m. adquirido por los clientes C_1, C_2 y C_6. El beneficio obtenido por todas las ventas es de 223 u.m.

En rojo se muestra una solución óptima para el problema. En esta solución existe mayor variedad en los precios de los productos. El rango de precios va desde 16 u.m. el producto más barato a las 53 u.m. del producto más caro. Cada cliente adquiere el producto más preferido entre los productos accesibles dado su presupuesto. Así, en este caso solo la mitad de los clientes adquieren el producto más preferido, esto es, con una preferencia con valor 5. En el caso del cliente C_7, con un presupuesto de 25 u.m., solo puede acceder a la compra de los productos P_2 y P_4 y adquiere el producto P_2 con un valor de preferencia igual a 2. El beneficio obtenido es de 308 u.m., muy superior a los beneficios que proporcionan las soluciones marcadas en verde y azul.

Este ejemplo ilustrativo pone de manifiesto la importancia de que el modelo de optimización formulado para calcular la solución óptima del problema, tenga en cuenta tanto el objetivo de la empresa de obtener el máximo beneficio posible, como la reacción de los clientes que a la vista de los precios establecidos por la empresa adquirirán el producto que les satisfaga entre los accesibles dado su presupuesto.

Tabla 1.1: Matriz de preferencias, vector de presupuestos y algunas soluciones factibles de un problema de determinación de precios clásico con 10 clientes y 5 productos.

Clientes	Productos					Presupuestos
	P_1	P_2	P_3	P_4	P_5	
C_1	-	3	🟢🔵🔴 **5**	-	4	53
C_2	🔴 **4**	-	🟢🔵 5	-	-	40
C_3	🟢🔵🔴 **5**	-	4	2	3	40
C_4	2	3	1	4	🟢🔵🔴 **5**	38
C_5	🔵 5	-	3	-	🔴 **4**	32
C_6	2	3	🔵 5	1	🔴 **4**	31
C_7	🔵 5	🔴 **2**	4	-	3	25
C_8	🔵 5	-	3	🔴 **4**	-	25
C_9	-	-	4	🔵🔴 **5**	-	25
C_{10}	4	🔵🔴 **5**	-	2	3	16
	Precios					Beneficio
🟢	36	42	40	39	35	151
🔵	24	13	25	16	22	222
🔴	40	16	53	25	31	308

La necesidad de encontrar el precio justo de cada producto que atienda al compromiso anterior justifica la formulación de un problema de determinación de precios como un problema de optimización binivel. La optimización binivel permite modelizar sistemas de toma de decisiones en los que se ven involucrados dos niveles de decisión con una estructura jerárquica. Cada uno de los decisores controla un conjunto de variables de decisión y tiene su propia función objetivo a optimizar. En este caso, el nivel superior de decisión representa a la empresa que determina los precios de los productos que vende. Su objetivo consiste en maximizar las ganancias obtenidas por la venta de los productos. En el nivel inferior de decisión se sitúan los clientes, los cuales, una vez fijados los precios de los productos y teniendo en cuenta sus preferencias de compra y presupuesto disponible, deciden de manera autónoma sobre el producto que desean comprar. Así, la empresa ha de tomar en consideración la reacción de los clientes a los precios que proponga para calcular el precio óptimo de los productos.

La solución óptima marcada en rojo en la tabla 1.1 es, efectivamente, una solución óptima binivel para ese problema. Se observa que el tratamiento del problema desde un enfoque binivel proporciona

mejores resultados que el simple hecho de establecer el precio de los productos de manera intuitiva. Por ejemplo, en la solución factible marcada en azul, el cliente C_4 adquiere el producto P_5, que es el producto que más prefiere, a un precio de 22 u.m. Este cliente sin embargo tiene un presupuesto de 38 u.m. por lo que si se hubiera establecido el precio del producto P_5 en 38 u.m., la compra del resto de clientes no se vería afectada y el cliente C_4 lo habría comprado igualmente pagando 16 u.m. más de lo que ha pagado realmente. Lo mismo ocurre en casos como el del cliente C_3 en la solución factible marcada en verde, que compra el producto P_1 por 36 u.m cuando realmente está dispuesto a pagar 40 u.m por ese mismo producto. Este hecho sugiere que, quizá, los precios que se establecen para los productos deben tomar valores en el conjunto de presupuestos de los clientes.

Para el planteamiento del problema de determinación de precios de múltiples productos es clave el desarrollo tecnológico experimentado durante los últimos años, ya que permite disponer de gran cantidad de información sobre el comportamiento y las preferencias de los clientes.

Por ejemplo, [Rusmevichientong et al., 2006] formulan un modelo no paramétrico de establecimiento de precios motivados por la cantidad de datos disponibles sobre preferencias de clientes recogidos por la empresa de automoción *General Motors* a través de un sitio web que recomienda automóviles a los clientes en base a sus necesidades y a su presupuesto.

1.2. Optimización binivel lineal

En esta sección se hace una revisión de los conceptos básicos de la optimización binivel lineal y de algunas aproximaciones recogidas en la literatura a la hora de tratar modelos de este tipo. Las referencias seguidas para esta revisión son [Calvete and Galé, 2020] y [Labbé and Violin, 2016].

Anteriormente, se ha introducido la optimización binivel como aquella que permite modelizar sistemas de toma de decisiones en los que se ven involucrados dos niveles de decisión con una estructura jerárquica y donde cada uno de los decisores controla un conjunto de variables de decisión y tiene su propia función objetivo a optimizar.

Sea $x \in \mathbb{R}^{n_1}$ el vector de variables de decisión del nivel superior e $y \in \mathbb{R}^{n_2}$ el vector de variables de decisión del nivel inferior. El decisor del nivel inferior (o segundo nivel), también llamado *seguidor* optimiza su función objetivo, f_2, una vez conocido el valor de las variables de decisión x controladas por el decisor del nivel superior. A este decisor del nivel superior (o primer nivel) se le denomina *líder*. Este, en cambio, una vez que tiene toda la información acerca de la reacción del nivel inferior, elige el valor de sus variables de decisión x para optimizar su función objetivo, f_1. Debido a esto, los problemas de optimización binivel se formulan como problemas de optimización en los que otro problema de optimización está incluido en el conjunto de restricciones.

Un problema de optimización binivel de máximo se formula, sin pérdida de generalidad, como

$$\max_{x} \quad f_1(x,y) \tag{1.1a}$$

sujeto a

$$(x,y) \in S_1 \tag{1.1b}$$

donde dado x, y es una solución óptima de

$$\max_{y} \quad f_2(x,y) \tag{1.1c}$$

sujeto a

$$(x,y) \in S_2 \tag{1.1d}$$

donde S_1 y S_2 son las regiones de factibilidad de los niveles superior e inferior, respectivamente.

En un problema de optimización binivel lineal, tanto las funciones objetivo como las restricciones de ambos niveles de decisión son funciones lineales. Un problema de optimización binivel lineal de máximo

se formula, sin pérdida de generalidad, como

$$\max_{x} \quad c_1 x + d_1 y \tag{1.2a}$$

sujeto a

$$A_1 x + B_1 y \leqslant b_1 \tag{1.2b}$$

$$x \geqslant 0 \tag{1.2c}$$

donde dado x, y es una solución óptima de

$$\max_{y} \quad d_2 y \tag{1.2d}$$

sujeto a

$$A_2 x + B_2 y \leqslant b_2 \tag{1.2e}$$

$$y \geqslant 0 \tag{1.2f}$$

donde para $i = 1,2$, c_i es un vector de dimensión $1 \times n_1$, d_i es un vector de dimensión $1 \times n_2$, b_i es un vector de dimensión $m_i \times 1$, A_i es una matriz $m_i \times n_1$, B_i es una matriz $m_i \times n_2$ y x e y tienen todas sus componentes son no negativas. Nótese que, en la función objetivo del seguidor no se incluyen las variables del primer nivel del decisión, x, porque proporcionan un valor constante y no afecta al cálculo del valor óptimo de las variables y dado dicho valor x.

Sea R el poliedro que definen las restricciones del nivel superior (1.2b) y (1.2c), sea S el poliedro que definen las restricciones del nivel inferior (1.2e) y (1.2f) y sea $T = R \cap S$. Dado un valor x, sea $S(x) = \{y \in \mathbb{R}^{n_2} : \ (x,y) \in S\}$ la región factible del problema del nivel inferior (1.2d)-(1.2f) y $M(x)$ el conjunto de soluciones óptimas de dicho problema, también llamado *conjunto de reacciones racionales del nivel inferior*. Se define la región factible del problema de optimización binivel (1.2), también llamada *región inducida*, como el conjunto

$$IR = \{(x,y) : \ (x,y) \in T, \ y \in M(x)\} \tag{1.3}$$

Cualquier punto de IR se denomina *solución factible binivel* del problema (1.2). El problema binivel queda formulado como:

$$\max_{x,y} \quad c_1 x + d_1 y \tag{1.4a}$$

$$\text{sujeto a} \quad (x,y) \in IR \tag{1.4b}$$

Cuando las restricciones (1.2b) involucran a las variables de decisión de ambos niveles se denominan *restricciones "coupling"*. En la literatura se ha estudiado que la existencia de este tipo de restricciones pueden provocar que el problema binivel sea no factible aún siendo el poliedro $T \neq \emptyset$. Un vector $x \in \mathbb{R}^{n_1}$ se denomina *admisible* si existe un vector $y \in \mathbb{R}^{n_2}$ tal que $(x,y) \in IR$. Cuando no existen valores admisibles, el problema binivel no es factible.

A diferencia de los problemas de optimización lineal, los problemas de optimización binivel lineal pueden no tener solución incluso cuando las funciones objetivo y las restricciones de ambos niveles sean funciones continuas sobre los conjuntos compactos que definen las regiones factibles de ambos niveles. En particular, surgen dificultades cuando el conjunto de reacciones racionales del nivel inferior, $M(x)$, no tiene un único elemento. El hecho de que existan varias soluciones óptimas del nivel inferior dado un valor x no afecta al valor de la función objetivo de este nivel, pero puede ser que con cada una de ellas se obtengan diferentes valores de la función objetivo del nivel superior.

En la literatura se han explorado varias aproximaciones para tratar este problema y asegurar que los problemas de optimización binivel están bien planteados. Una primera aproximación consiste en asumir que para cada valor de las variables del nivel superior, x, existe una única solución para el problema del nivel inferior, es decir, que el conjunto $M(x)$ tiene un único elemento.

Como la hipótesis anterior en algunos problemas no resulta razonable o es difícil de verificar, otras aproximaciones buscan elegir una única solución $y \in M(x)$ para evaluar la función objetivo del nivel superior cuando el conjunto $M(x)$ tiene más de un elemento. Dos enfoques a la hora de elegir soluciones de $M(x)$ son los siguientes.

- El enfoque optimista es el más extendido y asume que el nivel superior es capaz de influenciar al nivel inferior para que siempre elija la solución óptima que más beneficie al nivel superior, de manera que éste siempre obtiene el mejor valor posible para su función objetivo de entre los valores que le proporcionan las soluciones óptimas del nivel inferior. Esta aproximación permite demostrar importantes resultados de caracterización de la solución óptima del problema binivel lineal. En la formulación del problema binivel dado en 1.4 se ha asumido implícitamente esta aproximación optimista.

- El enfoque pesimista asume que el nivel inferior siempre elige la solución óptima que más perjudica al nivel superior, de manera que éste siempre obtiene el peor valor posible para su función objetivo de entre los valores que le proporcionan las soluciones óptimas del nivel inferior. Los problemas que se plantean con este enfoque suelen ser más complicados en cuanto a tratamiento y han sido menos estudiados en la literatura.

Existen diferentes métodos para tratar de resolver un problema de optimización binivel lineal. Uno de estos métodos consiste en reformular el problema como un problema lineal de un solo nivel sustituyendo el problema del nivel inferior por sus condiciones de Karush-Kuhn-Tucker cuando éstas son necesarias y suficientes. Esta transformación queda justificada por el hecho de que para un problema lineal las condiciones de Karush-Kuhn-Tucker caracterizan el conjunto de soluciones óptimas del problema.

El problema dual del problema del nivel inferior (1.2d)-(1.2f) es

$$\min_{y} \quad u(b_2 - A_2 x) \tag{1.5a}$$

sujeto a:

$$uB_2 \geqslant d_2 \tag{1.5b}$$

$$u \geqslant 0 \tag{1.5c}$$

Nótese que la región factible del problema dual (1.5) definida por las restricciones (1.5b) y (1.5c) no depende de x. Por tanto, los problemas duales asociados a los diferentes valores admisibles de x tienen la misma región factible.

Las condiciones de Karush-Kuhn-Tucker del problema del nivel inferior (1.5) son

$$\begin{aligned}
&B_2 y \leqslant b_2 - A_2 x \\
&uB_2 \geqslant d_2 \\
&(uB_2 - d_2)\, y = 0 \\
&u(b_2 - A_2 x - B_2 y) = 0 \\
&u, y \geqslant 0
\end{aligned}$$

Por tanto, el problema binivel lineal (1.2) considerando la aproximación optimista, en caso de óptimo

múltiple del problema del nivel inferior, se puede reformular como:

$$\max_{x,y} \quad c_1 x + d_1 y \tag{1.6a}$$

sujeto a

$$A_1 x + B_1 y \leqslant b_1 \tag{1.6b}$$

$$A_2 x + B_2 y \leqslant b_2 \tag{1.6c}$$

$$u B_2 \geqslant d_2 \tag{1.6d}$$

$$(u B_2 - d_2)\, y = 0 \tag{1.6e}$$

$$u\,(b_2 - A_2 x - B_2 y) = 0 \tag{1.6f}$$

$$x, y, u \geqslant 0 \tag{1.6g}$$

El problema (1.6) es un problema no lineal debido al uso de las condiciones de holgura complementaria (1.6e) y (1.6f). En la literatura se han propuesto diferentes aproximaciones para tratar esta no linealidad. Una aproximación propone sustituir las condiciones de holgura complementaria introduciendo nuevas restricciones que involucran variables binarias. De esta manera se obtiene un problema de optimización lineal entera que puede ser resuelto con técnicas clásicas ya conocidas para este tipo de problemas.

En la literatura también se han propuesto algoritmos similares al algoritmo de ramificación y acotación para los problemas de optimización lineal entera que eliminan del conjunto de restricciones las restricciones no lineales y comprueban en cada iteración si la solución incumbente satisface o no las restricciones no lineales dando lugar a un árbol de decisión con diferentes problemas.

Otra aproximación tiene en cuenta el hecho de que como el problema del nivel inferior es un problema lineal, este se puede sustituir bien por sus condiciones de Karush-Kuhn-Tucker o bien por las restricciones de factibilidad primal y de factibilidad dual junto con la condición de *strong duality*.

Todas las aproximaciones comentadas anteriormente utilizan algoritmos exactos para su resolución. Estos algoritmos suelen requerir tiempos de computación muy elevados y no son capaces de resolver problemas grandes en un tiempo razonable. Como alternativa, se han propuesto en la literatura algunos algoritmos metaheurísticos que intentan evitar el problema del tiempo de cómputo y son capaces de enfrentarse a problemas de mayor tamaño. Al igual que los algoritmos exactos, los algoritmos metaheurísticos, o bien se aprovechan de las propiedades geométricas de un problema determinado, o bien reformulan el problema binivel como un problema de un solo nivel. Los algoritmos metaheurísticos que se han propuesto para la resolución problemas de optimización binivel son en su mayoría algoritmos evolutivos, en concreto algoritmos genéticos. Este tipo de algoritmos han resultado eficaces para la resolución de este tipo de problemas.

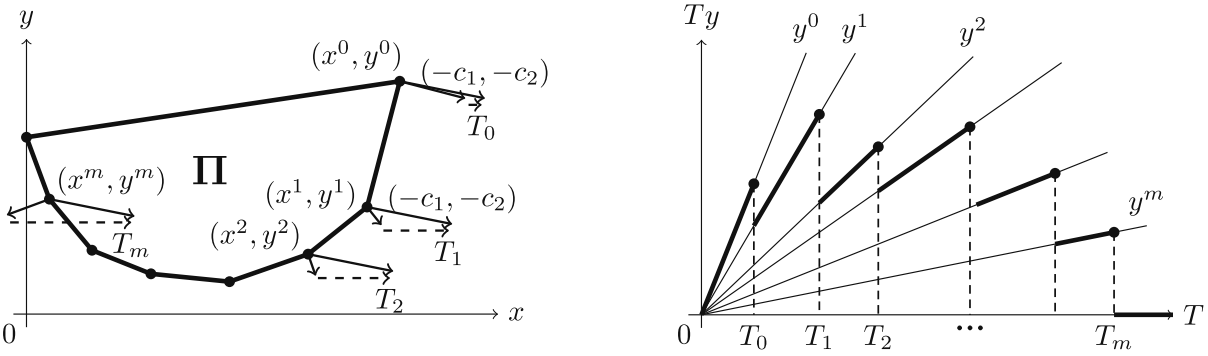
1.3. El problema de determinación de precios y la optimización binivel

La optimización binivel es útil para modelizar gran cantidad de situaciones de la vida real de establecimientos de precios como se expone en [Labbé and Violin, 2016]. Por ejemplo en problemas de ubicación de instalaciones, concesión de préstamos, problemas de producción energética, problemas de optimización en ambientes agrarios, producción de combustibles, problemas de transporte y un largo etcétera. En el campo de la economía, los problemas de determinación de precios son un clásico que aparece de forma recurrente y presentan una estructura fácilmente modelizable a través de un enfoque binivel.

Algunas situaciones que se modelizan de forma adecuada en el marco de un problema de determinación de precios son, por ejemplo, determinar el precio por la entrega de correo de manera urgente, fijar los precios de compañías aéreas o de ferrocarril, alquiler de vehículos, reserva de habitaciones de hotel... Pero el ejemplo clásico por excelencia y que ha sido muy tratado en la literatura es el de establecer el

precio de los peajes de las autopistas. En este problema existe un propietario de una red de carreteras que debe establecer el precio de los peajes por utilizar dicha red y una serie de viajeros debe decidir o bien viajar por la red de carreteras de pago o bien viajar por una red de carreteras gratuita alternativa [Labbé and Violin, 2016].

Anteriormente se ha comentado la idea de que los precios establecidos para los productos deben tomar valores en el conjunto de presupuestos de los clientes. Para ilustrar este hecho se muestra a continuación un gráfico extraído de [Labbé and Violin, 2016] de un caso concreto de problema binivel lineal en el que el nivel inferior solo controla dos variables de decisión, x e y, que pueden significar elegir un servicio de pago o uno gratuito, como en el ejemplo de las autopistas, y el nivel superior solo tiene que establecer el precio T del servicio.



(a) Región de factibilidad del seguidor y soluciones óptimas según el precio.

(b) Región de factibilidad del líder y valor de la función objetivo según la reacción del seguidor.

Figura 1.1: Funciones objetivo y región de factibilidad de un problema de establecimiento de precios extraído de [Labbé and Violin, 2016].

En la figura 1.1a se muestra la región factible para el nivel inferior o seguidor junto con su función objetivo. Cada vértice de la región factible es una solución óptima potencial para el nivel inferior de acuerdo con los resultados conocidos de optimización lineal. Esto permite determinar para cada vértice el valor de T para el que dicho vértice es una solución óptima. Por ejemplo, el vértice $\left(x^0, y^0\right)$ es óptimo para $T \in [0, T_0]$, el vértice $\left(x^1, y^1\right)$ es óptimo para $T \in (T_0, T_1]$ y así sucesivamente.

En la figura 1.1b se representa la función objetivo del primer nivel evaluada para cada valor de T atendiendo a la solución óptima del segundo nivel fijado ese valor de T. Se observa que esta función objetivo es lineal a trozos. Notemos que, cada punto extremo de la región factible del segundo nivel es óptimo para un intervalo de valores de T y la función objetivo del nivel superior es lineal dado el valor de ese punto extremo fijado. El salto en la función objetivo del primer nivel se produce en los valores de T en los que se pasa de un punto extremo a otro, esto es, en $T_0, T_1, \cdots, T_m$.

En el ejemplo de la figura 1.1b, la solución óptima consiste en establecer $T = T_1$. Si se estableciera un precio T entre T_0 y T_1, el nivel inferior pagaría menos de lo que estaría dispuesto a pagar por ese mismo servicio. En este caso, se puede demostrar que, el precio óptimo T será un valor en el conjunto $\{T_0, T_1, \cdots, T_m\}$. En el problema de establecimiento de precios de múltiples productos, también se puede demostrar que los precios tomen valores en el conjunto de presupuestos del nivel inferior.

Son muchas las variantes del problema de determinación de precios estudiadas en la literatura. La versión clásica del problema de determinación de precios descrita al principio de este capítulo ha sido tratada en profundidad en [Calvete et al., 2019] asumiendo que los clientes establecen un orden estricto de preferencia entre los productos. De este modo, conocidos los precios, de entre los productos que puede adquirir un cliente, hay un único producto más preferido, que es el adquirido por el cliente. Por tanto, el tratamiento binivel que se hace del problema sigue una aproximación optimista. En dicho trabajo se proponen varias formulaciones para el problema de fijación de precios sin empates en las preferencias y se diseñan algoritmos exactos para su resolución.

Una extensión del problema clásico de determinación de precios es el problema de determinación

de precios con empates (en inglés *rank pricing problem with ties*, o *RPPT* de forma abreviada), en el que se permite que los clientes sientan indiferencia a la hora de elegir entre un producto y otro. En este caso la elección por parte del nivel inferior no es única. Sin embargo, por la naturaleza del problema, está claro que el nivel inferior siempre elige el producto más barato de entre los productos que prefiere por igual. Este comportamiento se puede modelar considerando la aproximación pesimista del modelo binivel formulado para el establecimiento de precios.

En la tabla 1.2 se muestra la matriz de preferencias y el vector de presupuestos de un caso concreto de problema de determinación de precios con empates con 8 clientes y 5 productos extraído de [Domínguez et al., 2021]. La forma de asignar un valor de satisfacción a los productos es la misma que la que se utiliza en el ejemplo de la tabla 1.1, pero esta vez se permite que varios productos tengan el mismo valor de satisfacción asignado. En este caso también puede ser que algunos clientes no estén interesados en algunos productos y por tanto no se les asigne ningún valor de preferencia.

Tabla 1.2: Matriz de preferencias, vector de presupuestos y algunas soluciones factibles de un problema de determinación de precios con empates con 8 clientes y 5 productos.

Clientes	Productos					Presupuestos
	P_1	P_2	P_3	P_4	P_5	
C_1	● 5	3	● ● **5**	-	4	120
C_2	4	● ● ● **5**	5	-	-	95
C_3	5	4	2	● ● **5**	3	82
C_4	-	3	● 5	3	● **4**	82
C_5	-	● 5	3	● **4**	-	79
C_6	4	-	5	4	● ● **5**	65
C_7	3	4	1	● 5	● **2**	64
C_8	5	2	4	-	● ● **3**	53
	Precios					Beneficio
●	95	95	120	120	95	190
●	120	64	64	64	53	490
●	-	95	120	79	53	585

En verde y azul se muestran dos soluciones factibles binivel del problema que proporcionan un valor objetivo de 190 u.m. y 490 u.m. respectivamente. En rojo se muestra una solución óptima binivel del problema que proporciona un valor objetivo de 585 u.m.

En la solución factible marcada en verde se consideran precios elevados para los productos de forma que sólo los clientes C_1 y C_2 pueden adquirir productos. Aunque el cliente C_1 puede adquirir cualquiera de los productos elige el producto P_1 ya que su satisfacción es máxima y es más barato que el producto P_3 que también le daría satisfacción máxima.

Se observa, en este caso, que la existencia de empates en las preferencias de los clientes dificulta la elección de precios. Por ejemplo, en la solución factible marcada en azul, se han considerado precios bajos, de forma que todos los clientes pueden adquirir alguno de los producto. Como los productos más preferidos por el cliente con mayor presupuesto, C_1, son los productos P_1 y P_3 indistintamente, establecer un precio muy alto para el producto P_1 no aporta ningún beneficio ya que el cliente C_1 escoge el producto P_3 siempre que este sea más barato que el producto P_1.

El problema de determinación de precios de múltiples productos con empates ha sido tratado de manera extensa en [Domínguez et al., 2021]. En dicho trabajo se proponen varias formulaciones de un solo nivel y varios métodos de resolución exactos.

1.4. Contribución de este TFM

Lo novedoso de este TFM y la principal contribución es el estudio del problema de determinación de precios de múltiples productos con empates desde un enfoque binivel. Si un cliente está en duda por dos productos a la hora de realizar una compra, siempre elige el más barato. Esta la elección es la que peores resultados le proporciona al vendedor, por ello al tratar el modelo binivel hay que considerar la aproximación pesimista, que es la que se ajusta a la realidad. Esta forma de tratar el problema no se ha considerado en la literatura y supone una novedad en la forma de abordar el problema.

Lo desafiante del tratamiento del problema de esta manera es, precisamente, el considerar una aproximación pesimista. Como bien se ha comentado antes, el enfoque pesimista en la literatura ha supuesto muchas veces una gran dificultad añadida a la que ya de por sí tienen los modelos de optimización binivel. Esto ha provocado que, tratar un problema desde un enfoque pesimista de forma general no sea viable. En este caso, se trata de un problema particular y se va a desarrollar una aproximación adhoc.

A la hora de estudiar el problema de determinación de precios con empates, se ha aprovechado la estructura del problema para poder considerar el enfoque pesimista. El desarrollo propuesto está basado en la inclusión de una función biobjetivo en el nivel inferior que considera el objetivo del nivel inferior que trata de maximizar su satisfacción y la función objetivo del nivel superior de maximización del beneficio obtenido por la venta de los productos. Entre las posibles aproximaciones multiobjetivo, en este caso, con dos objetivos, la que refleja el comportamiento del segundo nivel es la aproximación lexicográfica, ya que existe una clara priorización entre los objetivos.

El problema biobjetivo del nivel inferior consiste, en realidad, en tantos problemas como clientes, ya que cada cliente es un seguidor independiente. Cada cliente, en primer lugar maximiza la satisfacción por la adquisición del producto entre aquellos accesibles y tras esto, entre todos los productos más preferidos, si hay empate, se minimiza el gasto por la compra. En el tratamiento del modelo binivel, para reformularlo como un problema de un solo nivel, se hace uso de resultados de teoría de dualidad para caracterizar las soluciones óptimas del nivel inferior. La idea de utilizar resultados de teoría de dualidad aparece ya en [Calvete et al., 2019]. Sin embargo, al asumir una aproximación optimista del problema, la transformación es más sencilla y solo requiere de la formulación de un problema dual. En este trabajo, al precisar utilizar la aproximación pesimista, es necesario formular dos problemas duales que ayudan a caracterizar la solución óptima lexicográfica. Este es un tratamiento novedoso del problema de determinación de precios con empates que permite calcular de forma exacta la solución óptima del problema en ejemplos pequeños, esto es, con pocos clientes y productos.

En este trabajo también se propone un algoritmo genético para resolver el problema de establecimiento de precios cuando se aumenta el número de clientes y productos. Utilizar técnicas metaheurísticas ha dado buenos resultados a la hora de trabajar con modelos de optimización binivel, como bien describen [Calvete and Galé, 2020] y justifica el intento de proponer algún algoritmo metaheurístico adecuado para el problema de determinación de precios con empates. Por ello, otra contribución de este TFM consiste en el diseño del primer algoritmo genético para resolver este tipo de problemas. El diseño es preliminar y se han considerado varias configuraciones del algoritmo. El comportamiento del algoritmo genético propuesto se ha evaluado en los problemas pequeños en comparación con las técnicas exactas de resolución. Como problemas de prueba se han utilizado los introducidos en [Domínguez et al., 2021]. Finalmente, se han resuelto problemas de mayor dimensión. La ventaja de los métodos metaheurísiticos consiste en obtener soluciones suficientemente "buenas"de forma rápida, esto es, en tiempos de cómputo razonables.

La estructura del trabajo de aquí en adelante es la siguiente. En el capítulo 2 se presenta una formulación binivel no lineal para el problema así como su posterior tratamiento haciendo uso de resultados de la teoría de dualidad. Posteriormente se linealiza el modelo para su resolución mediante algoritmos exactos. En el capítulo 3 se describe el algoritmo genético propuesto para resolver el problema de determinación de precios con empates. Por último, en el capítulo 4 se expone la experiencia computacional realizada. Se analiza el comportamiento del algoritmo genético propuesto y se compara su eficacia con la resolución exacta del modelo desarrollado en el capítulo 2. La memoria finaliza con un capítulo de conclusiones y trabajo futuro.

Capítulo 2

Formulación binivel para el problema de determinación de precios de múltiples productos con empates

2.1. Descripción del problema

El *RPPT* consiste, al igual que el caso sin empates, en determinar el precio de un conjunto de productos ofertados por una empresa con el objetivo de maximizar sus ganancias atendiendo a la selección de los clientes.

Los clientes disponen de un presupuesto con el que quieren comprar una única unidad de uno de los productos ofertados de entre los que les interesan. Una vez establecido el precio de todos los productos, cada cliente compra el que más le satisface de entre los que se puede permitir teniendo en cuenta que puede ocurrir que prefiera dos o más productos indistintamente. En caso de empate, el cliente elige el más barato de entre los que le satisfacen por igual. Si un cliente no puede permitirse ningún producto, no realiza ninguna compra. Al igual que en el caso del problema sin empates, se supone un suministro ilimitado de unidades de productos.

En este capítulo se hace un estudio del *RPPT* desde un enfoque binivel considerando la aproximación pesimista para resolver el problema de definición de una solución factible binivel cuando en el nivel inferior de decisión hay indiferencia entre varios óptimos múltiples. Para ello se ha propuesto incluir una función biobjetivo en el nivel inferior y aproximarla de manera lexicográfica. En primer lugar, para cada cliente se maximiza la satisfacción y en caso de óptimo múltiple, esto es, varios productos con el mismo valor de máxima satisfacción, se minimiza el gasto por la compra del producto. Los resultados de la teoría de la dualidad permiten caracterizar mediante un conjunto de restricciones, las soluciones óptimas del nivel inferior para cada selección de preciosy de este modo, se reformula el problema binivel como un problema de un solo nivel de decisión. Posteriormente se linealiza el modelo resultante para poder ser tratado por un software comercial.

La notación utilizada a lo largo del capítulo es la misma que se emplea en [Calvete et al., 2019] para el estudio del problema de determinación de precios sin empates.

Se denota por $K = \{1, \ldots, |K|\}$ al conjunto de clientes y por $I = \{1, \ldots, |I|\}$ al conjunto de productos. Por un lado, cada cliente $k \in K$ dispone de un presupuesto positivo b^k con el que comprar un producto de entre los productos ofertados. Por otro lado, cada cliente $k \in K$ tiene asociado un subconjunto de productos $S^k \subseteq I$ que son aquellos en los que está interesado. A cada producto i del conjunto S^k, el cliente k le da un valor de preferencia s_i^k que indica el orden de preferencia de dicho producto en la lista de preferencias del cliente. De esta forma, para dos productos $i, j \in S^k$, $s_i^k > s_j^k$ si el cliente k prefiere el producto i antes que el j.

Sin pérdida de generalidad, se puede asumir que todos los clientes están interesados en algún producto, es decir, que para todo $k \in K$, $S^k \neq \emptyset$; y que todos los productos están incluidos al menos en la lista de preferencias de algún cliente, es decir que para todo $i \in I$, existe $k \in K$ tal que $i \in S^k$. Si ocurre que un

cliente $k \in K$ no está interesado en ningún producto o un producto $i \in I$ no está incluido en ninguna lista de preferencias, dicho cliente o producto pueden eliminarse del problema de optimización.

2.2. Formulación de un solo nivel para el *RPPT* en la literatura

La formulación de un solo nivel que se propone en [Domínguez et al., 2021] para modelizar el problema de determinación de precios con empates utiliza algo más de notación de la que se ha introducido anteriormente. Sea $|L|$ el número de presupuestos distintos y se define el conjunto $L = \{1,2,\ldots,|L|\}$ como el conjunto de índices. Notemos que, $|L|(\leqslant |K|)$. Se denota por $B = \{b^1,\ldots,b^{|L|}\}$ al conjunto de diferentes presupuestos que además aparecen ordenados, esto es, si $l_1 < l_2$ entonces $b^{l_1} < b^{l_2}$.

Dado un producto $i \in I$ se define el conjunto $K_i = \{k \in K : i \in S^k\}$ como el conjunto de clientes que están interesados en el producto i.

Existe una partición del conjunto S^k, $\mathscr{S}^k = \{I_1^k,\ldots,I_{n^k}^k\}$. Cada clase de equivalencia contiene a los productos que prefiere el cliente k de manera indiferente siendo los de la clase 1 los más preferidos y los de la clase n^k los que menos. Se define para cada cliente k el conjunto $N^k = \{1,\ldots,n^k\}$ donde n^k es el número de clases de equivalencia en las que se divide el conjunto el cliente S^k.

Se define el conjunto $L_i = \{l \in L : \exists k \in K_i \text{ con } \sigma(k) = l\}$ como el conjunto de índices de presupuestos que son candidatos a ser precio óptimo del producto i. Además, para cada $k \in K_i$, $L_i^k = \{l \in L_i : l \leqslant \sigma(k)\}$ representa el conjunto de índices l de precios candidatos b^l para los que k puede comprar i en una solución factible. Finalmente, se define $L_{I_n^k} = \cup_{i \in I_n^k} L_i^k$ como el conjunto de índices $l \in L$ de precios candidatos b^l para los que k podría comprar algún producto $i \in I_n^k$.

En primer lugar, se definen la variables de decisión para cada $i \in I$ y para cada $l \in L_i$ las variables binarias v_i^l que toman valor 1 si el producto i tiene precio b^l. Para cada $k \in K$ y cada $n \in N^k$ se definen las variables binarias x_n^k que toman valor 1 si el cliente k compra un producto de la clase I_n^k y 0 en caso contrario. Por último, se definen también para $k \in K$ y cada $n \in N^k$ las variables continuas no negativas z_n^k que representan el beneficio obtenido del cliente k cuando compra un producto de la clase I_n^k.

Con toda eso, el modelo formulado de un solo nivel en [Domínguez et al., 2021] es:

$$\max_{v,x,z} \quad \sum_{k \in K} \sum_{n \in n^k} z_n^k \tag{2.1a}$$

sujeto a

$$\sum_{m \in M_i} v_i^m \leqslant 1 \qquad i \in I \tag{2.1b}$$

$$\sum_{n \in N^k} x_n^k \leqslant 1 \qquad k \in K \tag{2.1c}$$

$$x_n^k \leqslant \sum_{i \in I_n^k} \sum_{m \in M_i^k} v_i^m \qquad k \in K \quad n \in N^k \tag{2.1d}$$

$$\sum_{m \in M_i^k} v_i^m + \sum_{n\prime=n+1}^{n^k} x_{n\prime}^k \leqslant 1 \qquad k \in K \quad n < n^k \quad i \in I_n^k \tag{2.1e}$$

$$z_n^k \leqslant b^{\sigma(k)} x_n^k \qquad k \in K \quad n \in N^k \tag{2.1f}$$

$$z_n^k \leqslant b^{\sigma(k)} + \sum_{m \in M_i^k} \left(b^m - b^{\sigma(k)}\right) v_i^m \qquad k \in K \quad n \in N^k \quad i \in S_n^k \tag{2.1g}$$

$$v_i^m \in \{0,1\} \qquad i \in I \quad m \in M_i \tag{2.1h}$$

$$x_n^k \geqslant 0 \qquad k \in K \quad n \in N^k \tag{2.1i}$$

$$z_n^k \geqslant 0 \qquad k \in K \qquad n \in N^k \tag{2.1j}$$

La expresión (2.1a) es la función objetivo del problema que consiste en maximizar el beneficio objetivo por la empresa por la compra de los productos. Las restricciones (2.1b) aseguran que el precio de cada producto es único. Las restricicones (2.1c) garantizan que cada cliente compra a lo sumo un producto. Las restricciones (2.1d) impiden que un cliente k compre un producto de la clase I_n^k cuando no se lo puede permitir. Las restricciones (2.1e) son las restricciones de preferencia y garantizan que si un cliente puede permitirse comprar un producto i entonces no compra ningún otro producto j menos preferido. Las restricciones (2.1f) aseguran que si un cliente k no compra un producto de la clase I_n^k entonces $z_n^k = 0$. Las restricciones (2.1g) aseguran que cuando un cliente compra un producto $j \in S^k$, el beneficio asociado al cliente k y a la clase I_n^k es el mínimo de los precios de dicha clase.

2.3. Formulación binivel para el *RPPT*

En el nivel superior de decisión del modelo binivel se sitúa la empresa que actúa como líder y establece los precios de los productos que oferta. Para cada producto $i \in I$ se definen las variables continuas p_i que son controladas por el líder.

$$p_i = \text{precio del producto } i. \tag{2.2}$$

En el nivel inferior de decisión se sitúan los clientes, que eligen el producto que desean comprar atendiendo a su presupuesto y a la satisfacción que les proporciona cada producto. Notemos que, en el nivel inferior cada cliente resuelve un problema de optimización en el que solo intervienen las variables que controla y los precios establecidos por el líder. Esto significa que los clientes se pueden considerar independientes entre sí, ya que la decisión de un cliente no afecta ni se ve afectada por la decisión de otro cliente.

Cada cliente $k \in K$ controla el siguiente conjunto de variables de decisión que suelen denominarse variables de asignación, ya que para cada cliente se asigna el producto que compra. Para cada producto $i \in I$, se definen las variables binarias x_i^k:

$$x_i^k = \begin{cases} 1 & \text{si el cliente } k \text{ compra el producto } i. \\ 0 & \text{en caso contrario.} \end{cases} \tag{2.3}$$

El objetivo de la empresa consiste en maximizar sus ganancias, es decir, maximizar la suma de lo que pagan los clientes al comprar los productos seleccionados. Esta suma se expresa matemáticamente en función de las variables (2.2) y (2.3) como

$$\sum_{k \in K} \sum_{j \in S^k} p_j x_j^k \tag{2.4}$$

En el nivel inferior de decisión, el objetivo de cada cliente $k \in K$, consiste en maximizar su satisfacción por la compra realizada, es decir, adquirir el producto que más prefiere de entre los que se puede permitir. Esto se expresa matemáticamente en función de las variables (2.3) como

$$\sum_{j \in S^k} s_j^k x_j^k \tag{2.5}$$

A continuación, se formula el RPPT como un problema binivel bilineal/lineal con múltiples seguidores independientes:

$$\max_{p} \qquad \sum_{k \in K} \sum_{j \in S^k} p_j x_j^k \tag{2.6a}$$

sujeto a:

$$p_i \geqslant 0 \qquad i \in I \tag{2.6b}$$

donde para cada $k \in K$, $\left(x_i^k\right)_{i \in S^k}$ es una solución óptima de

$$\max_x \quad \sum_{j \in S^k} s_j^k x_j^k \tag{2.6c}$$

sujeto a:

$$\sum_{j \in S^k} x_j^k \leqslant 1 \tag{2.6d}$$

$$\sum_{j \in S^k} p_j x_j^k \leqslant b^k \tag{2.6e}$$

$$x_i^k \in \{0,1\} \qquad i \in S^k \tag{2.6f}$$

El problema binivel con múltiples seguidores independientes se ha tratado en [Calvete and Galé, 2007] En este trabajo, se demuestra que el problema 2.6 es equivalente a un problema binivel con un seguidor cuya función objetivo consiste en la suma de las funciones objetivo de todos los clientes y la región factible es la unión de las regiones de factibilidad de todos los clientes. En este TFM no se va a considerar este resultado y se va a continuar considerando el problema de cada cliente de forma individual.

En el caso sin empates, esto es, cuando el orden de las preferencias es estricto, el problema del nivel inferior tiene solución única, como se demuestra en la proposición (2.1) de [Calvete et al., 2019], y por tanto la elección del cliente está perfectamente determinada. Como se comentó en el capítulo anterior, esta elección coincide con la aproximación optimista del problema binivel, ya que de entre todas las soluciones óptimas del nivel inferior (en este caso solo hay uno), se elige la que le proporciona al nivel superior un mejor valor para su función objetivo.

Cuando para algunos clientes haya varios productos que les satisfagan por igual, es decir, que existan empates en las preferencias, el problema correspondiente a dicho cliente en el nivel inferior tiene óptimo múltiple. Estos clientes eligen siempre de entre los productos que les satisfacen por igual el producto más barato. Esta elección coincide con la aproximación pesimista del problema binivel, ya que de entre todas las soluciones óptimas del nivel inferior, se elige la que le proporciona al nivel superior un peor valor para su función objetivo.

Este TFM se centra en el RPPT y a diferencia de los modelos propuestos en [Domínguez et al., 2021] se va a considerar el tratamiento del problema desde un enfoque binivel. Para ello además, habrá que considerar la aproximación pesimista, es decir, la que se ajusta a la situación en la que existen empates para un cliente entre los productos más preferidos de entre los que le interesan y puede comprar.

La forma de resolver un empate cuando el cliente prefiere igualmente dos productos consiste en adquirir el producto más barato. Esto se modela considerando un segundo objetivo para cada cliente que consiste en minimizar el gasto que les proporciona el máximo nivel de satisfacción, es decir, adquirir el producto más barato de entre los que más les satisfagan por igual. Esto se expresa matemáticamente en función de las variables (2.2) y (2.3) como

$$\text{lexmax} \left(\sum_{j \in S^k} s_j^k x_j^k, -\sum_{j \in S^k} p_j x_j^k \right) \tag{2.7}$$

donde la expresión (2.7) representa la aproximación lexicográfica a un problema biobjetivo en el que los dos objetivos están priorizados por el decisor. Así maximizar de manera lexicográfica, significa maximizar primero sobre el primer objetivo y a continuación, sobre el conjunto de óptimos múltiples, maximizar sobre el segundo objetivo.

Por tanto, el *RPPT* se puede formular matemática como un problema binivel bilineal/lineal con múltiples seguidores independientes y dos objetivos en el nivel inferior de decisión:

$$\max_{p,x} \quad \sum_{k \in K} \sum_{j \in S^k} p_j x_j^k \tag{2.8a}$$

sujeto a

$$p_i \geqslant 0 \qquad i \in I \tag{2.8b}$$

donde para cada $k \in K$, $\left(x_i^k\right)_{i \in S^k}$ es una solución óptima de

$$\underset{x}{\text{lexmax}} \quad \left(\sum_{j \in S^k} s_j^k x_j^k, -\sum_{j \in S^k} p_j x_j^k \right) \tag{2.8c}$$

sujeto a

$$\sum_{j \in S^k} x_j^k \leqslant 1 \tag{2.8d}$$

$$\sum_{j \in S^k} p_j x_j^k \leqslant b^k \tag{2.8e}$$

$$x_i^k \in \{0,1\} \qquad i \in S^k \tag{2.8f}$$

La expresión (2.8a) es la función objetivo de la empresa o líder. Notemos que, debajo de max se han indicado como variables de decisión p, x. Como resultado de resolver el problema (2.8c)-(2.8f) para cada cliente se obtiene una elección del producto más barato entre los más preferidos. En el caso, de que hubiera más de un producto con ese mínimo precio, se elegiría cualquiera de ellos. En este caso, se puede dejar controlar los valores x al líder porque cualquier valor proporciona el mismo gasto para el cliente.

Las restricciones (2.8b) exigen que las variables de decisión, p_i, del primer nivel sean reales y no negativas. La expresión (2.8c) es la función objetivo de cada cliente, comentada con anterioridad. Las restricciones (2.8d) garantizan que cada cliente pueda comprar como máximo un producto de entre los de su lista de preferencias. Las restricciones (2.8e) garantizan que un cliente como máximo pague por comprar un producto el presupuesto del que dispone. Por último, las restricciones (2.8f) exigen que las variables de decisión, x_i^k, controladas por el cliente sean binarias.

2.4. Reformulación y tratamiento del problema binivel

2.4.1. Reformulación como un problema binivel binario puro

En los problemas de establecimiento de precios de productos en los que los clientes disponen de un presupuesto ([Rusmevichientong et al., 2006]) se demuestra que existe una solución óptima del problema en el que los precios toman como valor alguno de los valores en el conjunto de presupuestos de los clientes. El mismo razonamiento se extiende al problema (2.8).

En primer lugar, se introduce la misma notación que en los modelos de [Calvete et al., 2019] y [Domínguez et al., 2021]. El conjunto de valores de presupuestos distintos de clientes, dado que es posible que varios clientes tengan el mismo presupuesto, se denota por $B = \left\{b^1, \ldots, b^{|L|}\right\}$. Estos presupuestos además aparecen ordenados, esto es, si $l_1 < l_2$ entonces $b^{l_1} < b^{l_2}$. El valor $|L|(\leqslant |K|)$ es el número de presupuestos distintos y se define el conjunto $L = \{1,2,\ldots,|L|\}$ como el conjunto de índices.

La siguiente función asigna a cada cliente k el índice l si el presupuesto del cliente k es b^l:

$$\begin{array}{rccc} \sigma\colon & K & \longrightarrow & L \\ & k & \longrightarrow & l \end{array}$$

esto es, $\sigma(k)$ le asigna al cliente k la posición que ocupa su presupuesto en el conjunto ordenado de presupuestos diferentes, B.

Extendiendo el resultado de [Rusmevichientong et al., 2006] se garantiza que existe una solución óptima del problema (2.8) en la que $p_i \in B$ para todo producto $i \in I$. Por tanto, resulta interesante introducir un nuevo conjunto de variables binarias para reformular el problema. Estas variables ya fueron introducidas en [Calvete et al., 2019]. Para cada índice $l \in L$ y para cada producto $i \in I$ se definen las variables

binarias v_i^l que son controladas por el líder:

$$v_i^l = \begin{cases} 1 & \text{si el producto } i \text{ tiene precio } b^l. \\ 0 & \text{en caso contrario.} \end{cases} \tag{2.9}$$

Cada producto ha de tener un único precio, por lo que para cada producto $i \in I$ a lo más una de las variables v_i^l puede tomar valor 1 en una solución factible del problema. Para asegurar esto se impone que

$$\sum_{l=1}^{|L|} v_i^l \leqslant 1 \quad \text{para cada producto } i \in I \tag{2.10}$$

Se puede expresar el precio p_i de cada producto $i \in I$ en función de las variables v_i^l como:

$$p_i = \sum_{l=1}^{|L|} b^l v_i^l \tag{2.11}$$

Además, un cliente k no puede comprar productos de su lista de preferencias que valgan más que su presupuesto $b^{\sigma(k)}$. Para asegurar esto se impone para cada cliente $k \in K$ y para cada producto $i \in S^k$:

$$x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l \tag{2.12}$$

Si el precio del producto i es mayor que $b^{\sigma(k)}$ el término de la derecha vale 0 y en consecuencia, x_i^k también toma valor 0, lo que significa que el cliente k no compra el producto i. Si el precio del producto i fuera menor que $b^{\sigma(k)}$, entonces el termino de la derecha valdría 1 y por tanto no se añade ninguna restricción adicional al valor de la variable x_i^k que puede tomar valor 0 o 1, lo que significa que el cliente k tiene la opción de comprar el producto i si así lo desea.

La sustitución de las variables que determinan los precios, $p_i, i \in I$ por las variables $v_i^l, i \in I, l \in L$ permite reformular el modelo binivel entero mixto (2.8) como un modelo binivel binario puro. Para ello se han de modificar las funciones objetivo del nivel superior y de los múltiples seguidores del nivel inferior, (2.8a) y (2.8c), respectivamente y los conjuntos de restricciones (2.8b) y (2.8e).

La formulación matemática del modelo binivel binario puro es:

$$\max_{v,x} \quad \sum_{k \in K} \sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \tag{2.13a}$$

sujeto a:

$$\sum_{l=1}^{|L|} v_i^l \leqslant 1 \qquad i \in I \tag{2.13b}$$

$$v_i^l \in \{0,1\} \qquad i \in I \qquad l \in L \tag{2.13c}$$

donde para cada $k \in K$, $\left(x_i^k\right)_{i \in S^k}$ es una solución óptima de

$$\operatorname*{lexmax}_{x} \quad \left(\sum_{j \in S^k} s_j^k x_j^k, - \sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \right) \tag{2.13d}$$

sujeto a:

$$\sum_{j \in S^k} x_j^k \leqslant 1 \tag{2.13e}$$

$$x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l \qquad i \in S^k \tag{2.13f}$$

$$x_i^k \in \{0,1\} \qquad i \in S^k \tag{2.13g}$$

La expresión (2.13a) es la función objetivo del líder. Las restricciones (2.13b) garantizan que cada producto tenga a lo más un precio. Las restricciones (2.13c) exigen que las variables de decisión, v_i^l, del primer nivel de decisión sean binarias. La expresión (2.13d) es la función objetivo de cada cliente en el segundo nivel de decisión. Las restricciones (2.13e) garantizan que un cliente pueda comprar como máximo un producto de entre los de su lista de preferencias. Las restricciones (2.13f) garantizan que un cliente solo pueda comprar un producto si este se encuentra dentro de los que se ajustan a su presupuesto. Por último, las restricciones (2.13g) exigen que las variables de decisión, x_i^k, del segundo nivel de decisión sean binarias.

2.4.2. Reformulación como un problema de un solo nivel

Para formular el problema (2.13) como un problema de un solo nivel se han extendido las ideas basadas en teoría de la dualidad propuestas en [Calvete et al., 2019] para el tratamiento del problema de determinación de precios sin empates, que en el caso de empates han de considerar un problema biobjetivo.

A continuación, se va a considerar el problema de optimización de un cliente $k \in K$. Notemos que, cuando un cliente resuelve el problema de asignación, esto es, determina el producto que compra, se conocen el valor que toman las variables del nivel superior, $\{v_i^l, i \in I, l \in L\}$. Por tanto, el valor de parte de las variables controladas por el cliente, x_i^k, $i \in S^k$ queda determinado, ya que el cliente no podrá comprar aquellos productos con un precio superior a su presupuesto.

Para cada cliente $k \in K$ se define el conjunto de productos cuyo precio es menor o igual que su presupuesto:

$$I(k) = \left\{ i \in S^k : \sum_{l=1}^{\sigma(k)} v_i^l = 1 \right\}, \tag{2.14}$$

entonces, el valor de las variables

$$\left\{ x_i^k : k \in K,\ i \in S^k \setminus I(k) \right\} \tag{2.15}$$

es 0 ya que el cliente k no puede permitirse comprar los productos del conjunto $S^k \setminus I(k)$ por tener precios estrictamente mayores que su presupuesto.

La formulación del problema de un cliente $k \in K$ del nivel inferior, considerando lo anterior es:

$$\underset{x}{\text{lexmax}} \qquad \left(\sum_{j \in I(k)} s_j^k x_j^k, - \sum_{j \in I(k)} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \right) \tag{2.16a}$$

sujeto a

$$\sum_{j \in I(k)} x_j^k \leqslant 1 \tag{2.16b}$$

$$x_i^k \in \{0,1\} \qquad i \in I(k) \tag{2.16c}$$

De acuerdo con las proposiciones (3.2) y (3.3) recogidas en [Wolsey, 2021], la matriz que determina la región factible de los problemas del nivel inferior (2.16) es totalmente unimodular. Esto permite relajar las restricciones de integridad (2.16c).

En el tratamiento del problema biobjetivo de manera lexicográfica, se considera en primer lugar, para el cliente $k \in K$ el problema con el primer objetivo, maximizar la satisfacción:

$$\max_{x} \qquad \sum_{j \in I(k)} s_j^k x_j^k \tag{2.17a}$$

sujeto a

$$\sum_{j \in I(k)} x_j^k \leqslant 1 \tag{2.17b}$$

$$x_i^k \geqslant 0 \qquad i \in I(k) \tag{2.17c}$$

Sea la variable dual u^k correspondiente a la única restricción 2.17b del problema (2.17), entonces el problema dual es:

$$\min_{u} \quad u^k \tag{2.18a}$$

sujeto a

$$u^k \geqslant s_i^k \qquad i \in I(k) \tag{2.18b}$$

$$u^k \geqslant 0 \tag{2.18c}$$

Aplicando resultados conocidos sobre dualidad, las soluciones $\left(x_i^k\right)_{i \in I(k)}$ y u^k son soluciones óptimas de los problemas primal y dual, (2.17) y (2.18), respectivamente si y solo si:

$$\sum_{j \in I(k)} s_j^k x_j^k = u^k \tag{2.19a}$$

$$\sum_{j \in I(k)} x_j^k \leqslant 1 \tag{2.19b}$$

$$u^k \geqslant s_i^k \qquad i \in I(k) \tag{2.19c}$$

$$x_i^k \geqslant 0 \qquad i \in I(k) \tag{2.19d}$$

$$u^k \geqslant 0 \tag{2.19e}$$

Utilizando (2.19a) para sustituir u^k en (2.19c) y (2.19e), se tiene que el conjunto de restricciones que caracterizan las soluciones óptimas según el primer objetivo de cada cliente en el nivel inferior de decisión es:

$$\sum_{j \in I(k)} x_j^k \leqslant 1 \tag{2.20a}$$

$$\sum_{j \in I(k)} s_j^k x_j^k \geqslant s_i^k \qquad i \in I(k) \tag{2.20b}$$

$$x_i^k \geqslant 0 \qquad i \in I(k) \tag{2.20c}$$

$$\sum_{j \in I(k)} s_j^k x_j^k \geqslant 0 \tag{2.20d}$$

La restricción (2.20d) se cumple siempre ya que los parámetros s_i^k y las variables x_i^k son siempre mayores o iguales que 0. Por tanto, se puede prescindir de estas restricciones.

En la aproximación lexicográfica del problema biobjetivo, sobre el conjunto de soluciones óptimas caracterizado por las restricciones (2.20), el nivel inferior elige aquella que optimiza el segundo objetivo, dando lugar al siguiente problema:

$$\min_{x} \quad \sum_{j \in I(k)} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \tag{2.21a}$$

sujeto a:

$$\sum_{j\in I(k)} x_j^k \leqslant 1 \tag{2.21b}$$

$$\sum_{j\in I(k)} s_j^k x_j^k \geqslant s_i^k \qquad i \in I(k) \tag{2.21c}$$

$$x_i^k \geqslant 0 \qquad i \in I(k) \tag{2.21d}$$

Para caracterizar las soluciones óptimas del problema (2.21) se va a calcular su problema dual. Sea y^k la variable dual correspondiente a la restricción (2.21b). Asociada a cada restricción (2.21c) se introduce la variable dual w_j^k, $j \in I(k)$. El problema dual del problema (2.21) es:

$$\max \qquad -y^k + \sum_{j\in I(k)} s_j^k w_j^k \tag{2.22a}$$

sujeto a

$$-y^k + s_i^k \sum_{j\in I(k)} w_j^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l \qquad i \in I(k) \tag{2.22b}$$

$$y^k \geqslant 0 \tag{2.22c}$$

$$w_i^k \geqslant 0 \qquad i \in I(k) \tag{2.22d}$$

Del mismo modo, aplicando resultados conocidos sobre dualidad, $\left(x_i^k\right)_{i\in I(k)}$ e $\left(y^k, w_i^k\right)_{i\in I(k)}$ son soluciones óptimas de los problemas primal y dual, (2.21) y (2.22), respectivamente si y solo si:

$$\sum_{j\in I(k)} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k = -y^k + \sum_{j\in I(k)} s_j^k w_j^k \tag{2.23a}$$

$$\sum_{j\in I(k)} x_j^k \leqslant 1 \tag{2.23b}$$

$$\sum_{j\in I(k)} s_j^k x_j^k \geqslant s_i^k \qquad i \in I(k) \tag{2.23c}$$

$$-y^k + s_i^k \sum_{j\in I(k)} w_j^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l \qquad i \in I(k) \tag{2.23d}$$

$$x_i^k \geqslant 0 \qquad i \in I(k) \tag{2.23e}$$

$$y^k \geqslant 0 \tag{2.23f}$$

$$w_i^k \geqslant 0 \qquad i \in I(k) \tag{2.23g}$$

Despejando el valor de la variable y^k en (2.23a) y sustituyéndolo en (2.23d) y (2.23f), se tiene que el conjunto de restricciones que caracterizan las soluciones óptimas del problema biobjetivo para cada cliente $k \in K$ de manera lexicográfica es:

$$\sum_{j\in I(k)} x_j^k \leqslant 1 \tag{2.24a}$$

$$\sum_{j\in I(k)} s_j^k x_j^k \geqslant s_i^k \qquad i \in I(k) \tag{2.24b}$$

$$\sum_{j\in I(k)} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k - \sum_{j\in I(k)} s_j^k w_j^k + s_i^k \sum_{j\in I(k)} w_j^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l \qquad i \in I(k) \tag{2.24c}$$

$$\sum_{j\in I(k)} s_j^k w_j^k - \sum_{j\in I(k)} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l\right) x_j^k \geqslant 0 \tag{2.24d}$$

$$x_i^k \geqslant 0 \qquad i \in I(k) \tag{2.24e}$$

$$w_i^k \geqslant 0 \qquad i \in I(k) \tag{2.24f}$$

Nótese que, para cada cliente $k \in K$, en el conjunto de restricciones (2.24) solo se tienen en cuenta los productos del conjunto $I(k)$, ya que las variables de asignación relativas a los productos del conjunto $S^k \setminus I(k)$ toman valor 0. En la reformulación del problema binivel como un problema de un solo nivel se sustituyen los problemas de optimización de todos los clientes en el nivel inferior de decisión por el conjunto de restricciones que caracterizan a las soluciones óptimas de dichos problemas. Por ello, hay que hacer válidas las restricciones (2.24) para todos los productos de S^k de manera que actúen cuando $i \in I(k)$ y que no actúen cuando $i \in S^k \setminus I(k)$.

Restricciones (2.24a)

En el modelo de un solo nivel se incluye la restricción (2.13f) que controla que los clientes no compren productos que estén por encima de su presupuesto y que no era necesaria cuando se empleaba el conjunto $I(k)$. Por tanto, como las variables x_i^k valen 0 si $i \in S^k \setminus I(k)$, imponer la restricción (2.24a) es equivalente imponer:

$$\sum_{j\in S^k} x_j^k \leqslant 1 \tag{2.25}$$

ya que solo se están añadiendo a la suma términos que toman el valor0.

Restricciones (2.24b)

Las restricciones (2.24b) se sustituyen por:

$$\sum_{j\in S^k} s_j^k x_j^k \geqslant s_i^k \sum_{l=1}^{\sigma(k)} v_i^l \qquad i \in S^k \tag{2.26}$$

Cuando $i \in I(k)$ se tiene que $\sum_{l=1}^{\sigma(k)} v_i^l = 1$ y se garantizan las restricciones (2.24b). En el caso de un producto $i \in S^k \setminus I(k)$, se tiene que $\sum_{l=1}^{\sigma(k)} v_i^l = 0$, por tanto estas restricciones se cumplen siempre, ya que el término de la izquierda es siempre positivo.

Restricciones (2.24f)

Las variables w_i^k solo precisan ser definidas, dado un cliente $k \in K$ para $i \in I(k)$. Sin embargo, en la formulación del modelo binivel como un problema de un solo nivel, se han de definir para todos los valores de $i \in i, k \in K$. Por tanto, se va a imponer que las variables duales cuando no tiene sentido definirlas, esto es, para $i \in S^k \setminus I(k)$, tomen el valor 0:

$$w_i^k \leqslant M_1 \sum_{l=1}^{\sigma(k)} v_i^l \quad i \in I, k \in K \tag{2.27}$$

con M_1 una constante lo suficientemente grande.

Si $i \in S^k \setminus I(k)$, entonces $\sum_{l=1}^{\sigma(k)} v_i^l = 0$ y la restricción anterior garantiza que $w_i^k = 0$. Sin embargo, cuando $i \in I(k)$, entonces $\sum_{l=1}^{\sigma(k)} v_i^l = 1$ y la restricción anterior impone que $w_i^k \leqslant M_1$. Por tanto, M_1 debe ser una cota válida de la variable w_i^k para que la restricción no actúe cuando $i \in I(k)$. Además, la restricción (2.24f) se incluye ahora para todos los índices de productos y clientes:

$$w_i^k \geqslant 0 \quad i \in I, k \in K \tag{2.28}$$

Restricciones (2.24d)

Cuando $i \in S^k \setminus I(k)$ se tiene que las variables $x_i^k = 0$ y $w_i^k = 0$, entonces las restricciones (2.24d) son equivalentes al siguiente conjunto de restricciones:

$$\sum_{j \in S^k} s_j^k w_j^k - \sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \geqslant 0 \tag{2.29}$$

ya que, pasar en los sumatorios de $j \in I(k)$ a $j \in S^k$ solo añade términos en el sumatorio que toman valor 0.

Restricciones (2.24c)

Por último, en las restricciones (2.24c) los índices de los sumatorios también se pueden ampliar a $j \in S^k$ ya que solo se están añadiendo términos que valen 0. Sin embargo, esta restricción debe actuar únicamente cuando $i \in I(k)$.

Si $i \in S^k \setminus I(k)$, entonces el lado derecho de la restricción (2.24c) vale 0. Por tanto, para que la restricción no actúe en estos casos hay que sumar una cantidad lo suficientemente grande cuando $i \in S^k \setminus I(k)$. Esto se consigue imponiendo el siguiente conjunto de restricciones:

$$\sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k - \sum_{j \in S^k} s_j^k w_j^k + s_i^k \sum_{j \in S^k} w_j^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l + M_2 \left(1 - \sum_{l=1}^{\sigma(k)} v_i^l \right) \qquad i \in S^k \tag{2.30}$$

con M_2 una constante lo suficientemente grande.

De esta manera si $i \in I(k)$, entonces $\sum_{l=1}^{\sigma(k)} v_i^l = 1$ y la restricción actúa. Por otro lado, si $i \in S^k \setminus I(k)$, entonces $\sum_{l=1}^{\sigma(k)} v_i^l = 0$, $\sum_{l=1}^{\sigma(k)} b^l v_i^l = 0$ y la restricción correspondiente es:

$$\sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k - \sum_{j \in S^k} s_j^k w_j^k + s_i^k \sum_{j \in S^k} w_j^k \leqslant M_2 \qquad i \in S^k \setminus I(k) \tag{2.31}$$

Por tanto, el valor de M_2 ha de ser suficientemente grande para que la restricción anterior no actúe.

Reformulación del problema binivel

A continuación, se reformula el problema binivel binario puro (2.13) considerando las restricciones anteriores que extienden para cada cliente a todos los productos el conjunto de restricciones (2.24) y que caracterizan las soluciones óptimas del problema biobjetivo según la aproximación lexicográfica de cada cliente del nivel inferior de decisión:

$$\max_{v,x} \qquad \sum_{k \in K} \sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \tag{2.32a}$$

sujeto a

$$\sum_{l=1}^{|L|} v_i^l \leqslant 1 \qquad i \in I \tag{2.32b}$$

$$\sum_{j \in S^k} x_j^k \leqslant 1 \qquad k \in K \tag{2.32c}$$

$$x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l \qquad k \in K \quad i \in S^k \tag{2.32d}$$

$$\sum_{j \in S^k} s_j^k x_j^k \geqslant s_i^k \sum_{l=1}^{\sigma(k)} v_i^l \qquad k \in K \quad i \in S^k \tag{2.32e}$$

$$w_i^k \leqslant M_1 \sum_{l=1}^{\sigma(k)} v_i^l \qquad k \in K \quad i \in S^k \tag{2.32f}$$

$$\sum_{j \in S^k} s_j^k w_j^k - \sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k \geqslant 0 \qquad k \in K \tag{2.32g}$$

$$\sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k - \sum_{j \in S^k} s_j^k w_j^k + s_i^k \sum_{j \in S^k} w_j^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l + \\ + M_2 \left(1 - \sum_{l=1}^{\sigma(k)} v_i^l \right) \qquad k \in K \quad i \in S^k \tag{2.32h}$$

$$v_i^l \in \{0,1\} \qquad i \in I \quad l \in L \tag{2.32i}$$

$$x_i^k \geqslant 0 \qquad k \in K \quad i \in S^k \tag{2.32j}$$

$$w_i^k \geqslant 0 \qquad k \in K \quad i \in S^k \tag{2.32k}$$

Para resolver el *RPPT* a partir del problema de un solo nivel (2.32) es necesario encontrar cotas válidas para las constantes M_1 y M_2.

2.4.3. Búsqueda de cotas válidas M_1 y M_2

El cálculo de cotas válidas necesarias cuando se trabaja con la reformulación del problema binivel utilizando las propiedades de dualidad es un problema que [Kleinert et al., 2020] han demostrado es tan complejo, en general, como resolver el problema binivel original. Sin embargo, en algunos problemas binivel específicos es posible garantizar el cálculo de una constante válida atendiendo a las características del problema. Para el *RPPT* se van a calcular estas constantes válidas.

La constante M_1 debe ser una cota válida para cada una de las variables duales $\{w_j^k\}_{j \in S^k}$. Por otra parte, a partir de la restricción (2.29) se deduce que

$$\sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k - \sum_{j \in S^k} s_j^k w_j^k + s_i^k \sum_{j \in S^k} w_j^k \leqslant s_i^k \sum_{j \in S^k} w_j^k \tag{2.33}$$

por ser $s_i^k \sum_{j \in S^k} w_j^k \geqslant 0$. Dado que $s_i^k \leqslant |I|$ para todo $i \in I$, se tiene que:

$$\sum_{j \in S^k} \left(\sum_{l=1}^{\sigma(k)} b^l v_j^l \right) x_j^k - \sum_{j \in S^k} s_j^k w_j^k + s_i^k \sum_{j \in S^k} w_j^k \leqslant s_i^k \sum_{j \in S^k} w_j^k \leqslant |I| \sum_{j \in S^k} w_j^k \leqslant |I| \cdot C \tag{2.34}$$

donde C es una cota válida para la suma de variables duales $\sum_{j \in S^k} w_j^k$.

Por tanto, encontrar una cota válida M_2 se reduce a encontrar una cota válida C.

Las variables duales $\{w_j^k\}_{j \in S^k}$ se introducen al formular el problema dual (2.22) del problema primal (2.21). Este problema primal es el problema que cada cliente en el nivel inferior de decisión se plantea para optimizar su segundo objetivo entre los óptimos múltiples del primer objetivo y que consiste en minimizar el gasto de compra una vez elegidos los productos más satisfactorios entre los accesibles según su presupuesto. La solución óptima de este problema es decidir comprar el producto de precio mínimo.

La estrategia que se propone a continuación para encontrar cotas válidas para las constantes M_1 y M_2 se introduce en [Shen et al., 2012] para un problema de optimización completamente diferente al que se está considerando. La idea es encontrar de manera constructiva una solución factible dual con la garantía de ser óptima. Los valores de esta solución proporcionan cotas válidas para las variables duales del problema. Como el problema dual se plantea para encontrar un conjunto de restricciones que caracterice las soluciones óptimas del problema primal, es suficiente encontrar una solución óptima dual que permita hacer la caracterización.

Sean $\{p_j\}_{j \in S^k}$ los precios establecidos para los productos de S^k y $\{s_j^k\}_{j \in S^k}$ los valores de satisfacción que proporcionan los productos de S^k a un cliente k. Se denota por s_{max} a la satisfacción máxima de $\{s_j^k\}_{j \in S^k}$ y por p_{min} al precio mínimo de los productos que proporcionan esa satisfacción máxima s_{max} al cliente k, entonces la solución

$$y^k = -p_{min} + s_{max} \cdot w_0 \tag{2.35a}$$

$$w_0 = \min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + \frac{p_{min}}{s_{max}} \tag{2.35b}$$

$$w^k = (0, \ldots, 0,\ w_0,\ 0, \ldots, 0) \tag{2.35c}$$

donde w_0 es la única componente no nula del vector w^k y ocupa una posición i en el vector que verifica que el producto i le proporciona satisfacción máxima al cliente k.

En primer lugar se comprueba que (y^k, w^k) es una solución factible dual para el problema dual (2.22) correspondiente al cliente k.

En efecto, para todo los productos i que proporcionan máxima satisfacción, se tiene que

$$y^k = -p_{min} + s_{max} \cdot w_0 = -p_{min} + s_{max} \left(\min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + \frac{p_{min}}{s_{max}} \right) =$$

$$= -p_{min} + s_{max} \cdot \min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + p_{min} = s_{max} \cdot \min_{j \in S^k} \left(\frac{p_j}{s_j} \right) \geqslant 0$$

$$w^k = \left(0, \ldots, 0,\ \min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + \frac{p_{min}}{s_{max}},\ 0, \ldots, 0 \right) \geqslant 0$$

$$-y^k + s_i^k \sum_{j \in I(k)} w_j^k = -\left(-p_{min} + s_{max} \cdot w_0 \right) + s_i^k \left(\min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + \frac{p_{min}}{s_{max}} \right) =$$

$$= p_{min} - s_{max} \cdot w_0 + s_i^k \cdot w_0 \leqslant p_{min} - s_{max} \cdot w_0 + s_{max} \cdot w_0 =$$

$$= p_{min} \leqslant p_i = \sum_{l=1}^{\sigma(k)} b^l v_i^l$$

Además, el valor de la función objetivo del problema dual evaluada en esta solución es

$$-y^k + \sum_{j \in I(k)} s_j^k w_j^k = -\left(-p_{min} + s_{max} \cdot w_0 \right) + s_{max} \left(\min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + \frac{p_{min}}{s_{max}} \right) =$$

$$= p_{min} - s_{max} \cdot w_0 + s_{max} \cdot w_0 = p_{min}$$

que coincide con el valor óptimo de la función objetivo del problema primal (2.21), por lo que la solución construida (2.35) además de ser factible dual, también es óptima dual.

Se sabe que $1 \leqslant s_j^k \leqslant |I|$ para todo $j \in I(k)$, $k \in K$, en concreto, esta desigualdad también la cumple s_{max}. Además, $b^1 \leqslant p_j \leqslant b^{\sigma(k)}$ para todo $j \in I(k)$, en concreto, esta desigualdad también la cumple p_{min}.

Por tanto, se tiene que

$$w_0 = \min_{j \in S^k} \left(\frac{p_j}{s_j} \right) + \frac{p_{min}}{s_{max}} \leqslant \frac{b^{\sigma(k)}}{1} + \frac{b^{\sigma(k)}}{1} = 2b^{\sigma(k)}$$

Y como el resto de componentes de w^k son 0, esta cota es válida tanto para cada una de las componentes de w^k, como para la suma de variables duales $\sum_{j \in I(k)} w_j^k$. De esta forma, las constantes M_1 y M_2 pueden tomar valores

$$M_1 = 2b^{\sigma(k)}$$
$$M_2 = 2|I|b^{\sigma(k)}$$

2.4.4. Linealización del problema

En el modelo de un solo nivel (2.32) aparecen términos no lineales tanto en la función objetivo (2.32a) como en las restricciones (2.32h) y (2.32g). Para tratar esta no linealidad, al igual que en [Calvete et al., 2019], se introduce un nuevo conjunto de variables que permite linealizar el problema. Para cada cliente $k \in K$ y para cada producto $i \in S^k$ se definen las variables continuas z_i^k.

$$z_i^k = \text{beneficio obtenido del cliente } k \text{ por la compra del producto } i. \tag{2.36}$$

cuyo valor se define a partir de un conjunto de restricciones que se añaden a la formulación del problema. El beneficio obtenido del cliente k por la compra el producto i no puede ser mayor que el precio establecido para dicho producto. Para asegurar esto se impone que

$$z_i^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l \quad \text{para cada cliente } k \in K \text{ y para cada producto } i \in S^k \tag{2.37}$$

Además, el beneficio obtenido del cliente k por la compra el producto i ha de ser siempre menor o igual que el presupuesto del cliente y necesariamente ha de ser 0 si el cliente k no compra el producto i. Para asegurar esto se impone que

$$z_i^k \leqslant b^{\sigma(k)} x_i^k \quad \text{para cada cliente } k \in K \text{ y para cada producto } i \in S^k \tag{2.38}$$

Con estas nuevas variables, la formulación del problema linealizado es

$$\max \quad \sum_{k \in K} \sum_{j \in S^k} z_j^k \tag{2.39a}$$

sujeto a

$$\sum_{l=1}^{|L|} v_i^l \leqslant 1 \qquad i \in I \tag{2.39b}$$

$$\sum_{j \in S^k} x_j^k \leqslant 1 \qquad k \in K \tag{2.39c}$$

$$x_i^k \leqslant \sum_{l=1}^{\sigma(k)} v_i^l \qquad k \in K \quad i \in S^k \tag{2.39d}$$

$$z_i^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l \qquad k \in K \quad i \in S^k \qquad (2.39e)$$

$$z_i^k \leqslant b^{\sigma(k)} x_i^k \qquad k \in K \quad i \in S^k \qquad (2.39f)$$

$$\sum_{j \in S^k} s_j^k x_j^k \geqslant s_i^k \sum_{l=1}^{\sigma(k)} v_i^l \qquad k \in K \quad i \in S^k \qquad (2.39g)$$

$$w_i^k \leqslant M_1 \sum_{l=1}^{\sigma(k)} v_i^l \qquad k \in K \quad i \in S^k \qquad (2.39h)$$

$$\sum_{j \in S^k} s_j^k w_j^k - \sum_{j \in S^k} z_j^k \geqslant 0 \qquad k \in K \qquad (2.39i)$$

$$\sum_{j \in S^k} z_j^k - \sum_{j \in S^k} s_j^k w_j^k + s_i^k \sum_{j \in S^k} w_j^k \leqslant \sum_{l=1}^{\sigma(k)} b^l v_i^l + M_2 \left(1 - \sum_{l=1}^{\sigma(k)} v_i^l\right) \qquad k \in K \quad i \in S^k \qquad (2.39j)$$

$$v_i^l \in \{0,1\} \qquad i \in I \quad l \in L \qquad (2.39k)$$

$$z_i^k \geqslant 0 \qquad k \in K \quad i \in S^k \qquad (2.39l)$$

$$x_i^k \geqslant 0 \qquad k \in K \quad i \in S^k \qquad (2.39m)$$

$$w_i^k \geqslant 0 \qquad k \in K \quad i \in S^k \qquad (2.39n)$$

Este problema de un solo nivel es un problema de programación entera mixto que se puede resolver directamente con un software comercial como CPLEX.

Capítulo 3

Un algoritmo evolutivo para la resolución del problema de determinación de precios de múltiples productos con empates

En este capítulo se propone un algoritmo evolutivo de tipo genético para resolver el problema de determinación de precios de múltiples productos con empates descrito en los capítulos anteriores. El algoritmo ha sido programado en la versión 2022.07.2-576 del IDE RStudio con la versión 4.3.1 del lenguaje de programación R. El código programado para del algoritmo se incluye en el anexo A.

3.1. Los algoritmos evolutivos

El desarrollo de esta sección se basa en los contenidos desarrollados por Leticia Hernando durante los meses de febrero y marzo de 2022 en la asignatura *Algoritmos bioinspirados y técnicas de computación evolutiva* del *Máster Universitario en Modelización e Investigación matemática, Estadística y Computación* de la Universidad de Zaragoza [Hernando, 2022].

Los algoritmos evolutivos son una familia de algoritmos metaheurísticos utilizados en optimización con la intención de encontrar soluciones óptimas o casi óptimas de problemas para los que los métodos clásicos de resolución exacta tardan mucho tiempo en hallar a una solución óptima.

Este tipo de algoritmos se llaman evolutivos o también basados en poblaciones porque tienen un símil con los principios de la teoría de la evolución biológica. Esta teoría establece que en un ambiente con recursos limitados donde vive una población, la competición por los recursos provoca que los individuos mejor adaptados al medio sobrevivan. Los individuos de una población se reproducen dando lugar a nuevas generaciones de individuos con cierta variabilidad entre ellos. Esta variabilidad hace que unos individuos estén mejor adaptados al medio que otros. La selección natural evalúa la adaptación de todos los individuos presentes en una generación y hace que sobrevivan los individuos mejor adaptados provocando que la adaptación de la población en general aumente generación tras generación.

La estructura de flujo de un algoritmo evolutivo para un problema de optimización es la siguiente:

1. Se define una población inicial formada por N soluciones factibles del problema. Normalmente, esta población inicial se define de forma aleatoria o mediante otros algoritmos (constructivos, de búsqueda local...).

2. Se define un criterio de parada. Los criterios de parada más usados son:

   a) Un tiempo máximo de ejecución del algoritmo.

   b) Un número máximo de iteraciones a realizar.

   c) No observar una mejora significativa tras un número concreto de generaciones sucesivas.

   d) Haber alcanzado un valor deseado de la función objetivo.

3. Se varía de forma aleatoria el conjunto de soluciones disponible para obtener nuevas soluciones distintas (los individuos se reproducen para dar lugar a nuevos individuos). El objetivo que persigue la variación es combinar información de soluciones ya disponibles para obtener nuevas soluciones que posiblemente sean mejores, aunque puede ocurrir que surjan soluciones peores que las disponibles.

4. Se evalúa la función objetivo sobre el conjunto de soluciones disponible (evaluación de la adaptación de los individuos al medio). Al valor de la función objetivo que proporciona una solución se le denomina *fitness* de dicha solución.

5. Se seleccionan las N soluciones que pasan a formar la población de la siguiente iteración (selección natural de los individuos que pasan a la siguiente generación). Los criterios de selección más utilizados son:

   a) Selección elitista. Se escogen las N mejores soluciones de las que se disponga, es decir las N soluciones que tienen mejor *fitness*.

   b) Selección estocástica. Se seleccionan los individuos de forma aleatoria con probabilidades proporcionales a su *fitness*.

   c) Se seleccionan únicamente los descendientes de una generación y no sus progenitores.

En la figura 3.1 se observa gráficamente el flujo descrito anteriormente para un algoritmo evolutivo.

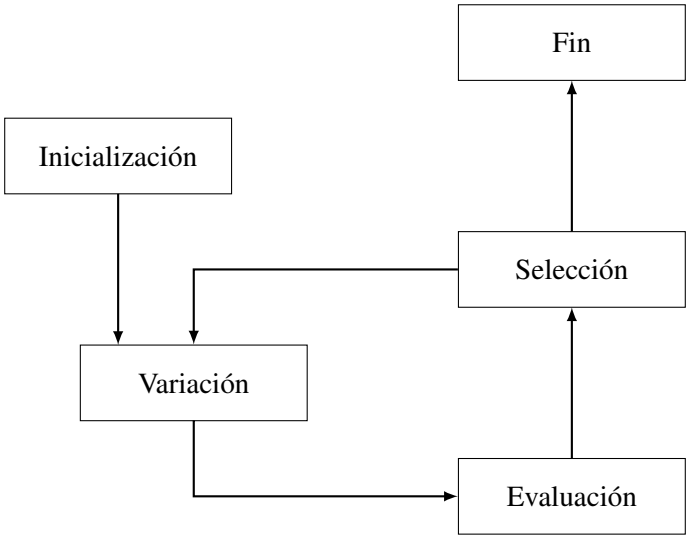


Figura 3.1: Flujo que sigue un algoritmo evolutivo general.

Los algoritmos genéticos son un tipo concreto de algoritmos evolutivos. En ellos, las soluciones o individuos se denominan cromosomas y estos su vez están formados por genes, que son cada una de las posiciones del cromosoma. El valor que almacena cada gen se denomina alelo. En lenguaje matemático, los cromosomas se codifican como vectores y los genes son cada una de las componentes del vector. Los valores almacenados en cada componente del vector son los alelos.

En los algoritmos genéticos la fase de variación, también llamada fase reproductiva, está formada por los procesos de cruce y mutación. Hay muchos tipos de operadores de cruce y mutación definidos en la literatura. Conviene estudiar la forma y las propiedades de las soluciones factibles del problema a resolver para definir de forma adecuada estos operadores, ya que al aplicarlos a las soluciones disponibles para crear nuevas soluciones conviene que las nuevas soluciones creadas también sean soluciones factibles del problema.

La fase de cruce consiste en seleccionar a un número de individuos de la población para combinarlos de la manera que determine el operador utilizado y obtener individuos nuevos que contengan parte de la

información de los progenitores. La forma de seleccionar a los progenitores también debe ser definida para cada algoritmo concreto.

La fase de mutación consiste en modificar levemente las soluciones obtenidas tras el cruce con el objetivo de introducir cierta variabilidad que permita aumentar el espacio de búsqueda de soluciones. La mutación puede hacerse de varias formas, algunos operadores de mutación mutan cada gen de forma independiente con cierta probabilidad y otro operadores mutan el cromosoma en conjunto con cierta probabilidad. Hay que asegurarse de que la probabilidad de mutación utilizada no sea ni muy alta, ya que se perdería la información de los progenitores, ni muy baja, ya que no se introduciría casi variabilidad quedando restringido el espacio de búsqueda a una zona muy reducida.

La estructura básica de un algoritmo genético se representa gráficamente en la figura 4.4b.

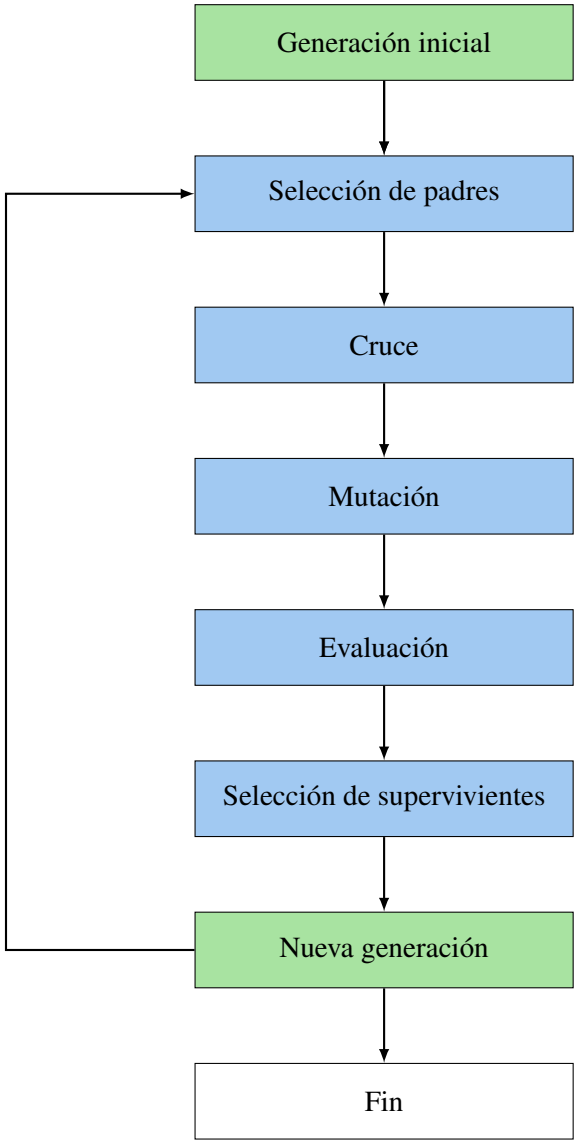


Figura 3.2: Flujo que sigue un algoritmo genético general.

Concretamente, para la resolución del problema de determinación de precios de múltiples productos con empates descrito en los capítulos anteriores se propone un algoritmo evolutivo de tipo genético.

3.2. Descripción del algoritmo genético propuesto

En esta sección se describe el algoritmo genético propuesto para resolver el problema de determinación de precios de múltiples productos con empates. Se determinan los operadores utilizados en el algoritmo, la forma de evaluar la *fitness* de los individuos y los parámetros utilizados en cada uno de los operadores.

3.2.1. Definición de los cromosomas

El objetivo del problema de determinación de precios es establecer el precio de los diferentes productos y se sabe que estos precios resultan ser valores del conjunto de presupuestos distintos de los clientes, como se ha explicado anteriormente. En el algoritmo genético propuesto, se definen los cromosomas o individuos de la población como vectores de longitud m, donde m es el número de productos del problema y cuyas componentes son valores del conjunto $L = \{1, \ldots, |L|\}$ donde $|L|$ es el número de presupuestos diferentes de los clientes del problema.

Así, si la componente 2 de un cromosoma es 7 significa que en la solución que representa dicho cromosoma, al segundo producto se le asigna el séptimo presupuesto del conjunto de presupuestos distintos $B = \left\{b^1, \ldots, b^{|L|}\right\}$ donde $b^1 < \ldots < b^{|L|}$.

En la figura 3.3 se representa gráficamente la apariencia de un cromosoma definido como un vector de longitud m cuyas componentes son elementos del conjunto $L = \{1, \ldots, |L|\}$.

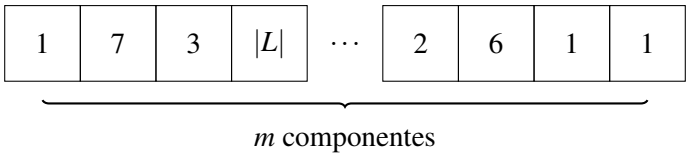


Figura 3.3: Representación gráfica de un cromosoma del algoritmo genético propuesto.

3.2.2. Definición de la función *fitness*

En el problema de determinación de precios el objetivo del nivel superior consiste en maximizar los beneficios obtenidos por la venta de los productos. En el algoritmo genético propuesto se utiliza también esta función objetivo para evaluar la *fitness* de los cromosomas que representan las diferentes soluciones obtenidas durante la ejecución del algoritmo.

Los cromosomas se han definido de tal manera que representan diferentes combinaciones de precios de los productos. Para evaluar la *fitness* de un cromosoma es necesario determinar qué producto compran los clientes dada esa combinación de precios de los productos. Una vez conocidos los productos comprados por los clientes, se suma el precio de los productos vendidos para conocer el beneficio que obtiene el nivel superior.

Matemáticamente, la función objetivo se expresa como:

$$\sum_{k \in K} \sum_{j \in S^k} z_j^k \tag{3.1}$$

donde, como se ha descrito en el capítulo 2, K denota el conjunto de clientes, S^k denota el conjunto de productos en los que está interesado el cliente k y z_j^k es el beneficio obtenido del cliente k por la compra del producto j.

Para evaluar la *fitness* de una solución, se han programado dos funciones en R.

La primera función, *compra*, determina, dada una combinación de precios, un cliente $k \in K$ y los datos que definen un problema, el producto que compra dicho cliente de acuerdo con sus preferencias, escogiendo de entre los que se puede permitir, el que más satisfacción le proporciona y en caso de que haya varios que cumplan estas características, el más barato.

La segunda función, *eval.fitness*, determina, dada una combinación de precios y los datos que definen un problema y apoyándose en la función *compra*, el beneficio obtenido por el nivel superior cuando establece la combinación de precios dada.

En el anexo A, en los fragmentos de código A.1 y A.2 se muestra el código de las funciones *compra* y *eval.fitness*,respectivamente.

3.2.3. Selección de padres para el cruce

El método de selección de padres para el cruce definido para el algoritmo genético es el que en la literatura se conoce como *selección por torneo*. Esta forma de seleccionar los padres consiste en:

1. Elegir el tamaño n del torneo. En este caso se ha elegido $n = 2$.
2. Escoger al azar n individuos de la población con o sin reemplazamiento. En este caso se ha decidido usar el método con reemplazamiento de manera que puede elegirse al mismo individuo varias veces para participar en el torneo.
3. Escoger al mejor individuo (en cuanto a *fitness*) de entre los n seleccionados para el torneo. Este individuo interviene luego en el cruce.
4. Repetir tantos torneos como individuos sean necesarios para el cruce. En este caso, como para el cruce solo se necesitan dos individuos, se realizan únicamente dos torneos.

En el algoritmo 1 se muestra el pseudocódigo que define el método de selección de padres utilizado.

Algoritmo 1: Selección de padres por torneo

Datos:
1 n: número de individuos que participan en el torneo
2 *poblacion*: población a la que pertenecen los individuos
3 k: número de padres que se quiere seleccionar
Resultado: $padre_1, \ldots, padre_k$
4 **para** i **desde** 1 **a** k **hacer**
5 $\quad candidatos \leftarrow n$ individuos de *poblacion* elegidos aleatoriamente
6 $\quad padre_i \leftarrow$ individuo con mejor *fitness* de $candidatos$
7 **fin**
8 **devolver** $padre_1, \ldots, padre_k$

3.2.4. El operador de cruce

El operador de cruce definido para el algoritmo genético es el que en la literatura se conoce como *one point crossover*. En este operador de cruce intervienen dos individuos como padres y el cruce de ambos padres produce dos hijos. El modo de hacer el cruce es:

1. Dados dos cromosomas de longitud m, se elige una componente n entre 1 y m de forma aleatoria y común para los dos padres.
2. Se dividen los padres en ese punto n elegido anteriormente, quedando por un lado las n primeras componentes y por el otro las componentes restantes desde la $(n+1)$-ésima hasta la m-ésima.
3. El primer hijo se forma combinando las primeras n componentes del primer padre junto con las componentes desde la $(n+1)$-ésima hasta la m-ésima del segundo padre.
4. El segundo hijo se forma análogamente combinando las primeras n componentes del segundo padre junto con las componentes desde la $(n+1)$-ésima hasta la m-ésima del primer padre.

En la figura 3.4 se observa gráficamente como actúa el operador de cruce *one point crossover* en un caso concreto donde los cromosomas tienen longitud 5 y cuyas componentes son elementos del conjunto $\{1,2,\ldots,7\}$.

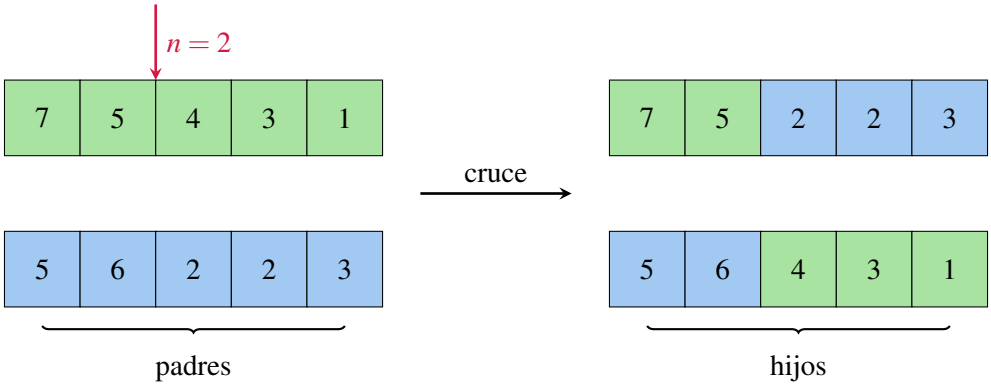


Figura 3.4: Funcionamiento del operador de cruce *one point crossover*

En el algoritmo 2 se muestra el pseudocódigo que define el operador de cruce utilizado.

Algoritmo 2: Operador de cruce

Datos: $padre1$, $padre2$
Resultado: $hijo1$, $hijo2$
1 $n \leftarrow$ coordenada aleatoria del cromosoma
2 $hijo1 \leftarrow$ (primeras n componentes del $padre1$, componentes a partir de $n+1$ del $padre2$)
3 $hijo2 \leftarrow$ (primeras n componentes del $padre2$, componentes a partir de $n+1$ del $padre1$)
4 **devolver** $hijo1$, $hijo2$

En en Anexo A, en el fragmento de código A.3 se muestra el código de la función *cruce* que ejecuta el operador de cruce definido para el algoritmo genético.

3.2.5. El operador de mutación

Dado un cromosoma formado por m genes, el operador de mutación definido para el algoritmo genético, altera cada gen de manera independiente con probabilidad de mutación p_m. La forma de mutar un gen es simplemente cambiar el valor de dicho gen por cualquier otro valor del conjunto $\{1,\ldots,|L|\}$.

Como cada gen indica el índice del presupuesto que proporciona el precio de un producto, cambiar un precio por otro sigue dando lugar a soluciones factibles del problema por lo que no hay que preocuparse de que surjan soluciones no factibles tras la mutación como sucede, por ejemplo, en casos en los que los cromosomas están definidos como permutaciones. En estos casos, si no se elige con cuidado el operador de mutación, a la hora de mutar individuos se pueden obtener soluciones no factibles para el problema.

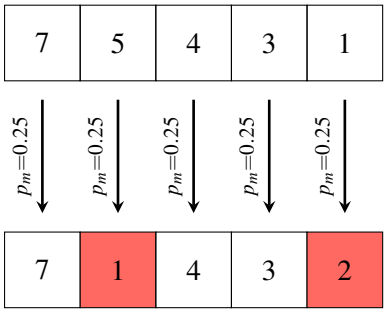


Figura 3.5: Funcionamiento del operador de mutación

En la figura 3.5 se observa gráficamente como actúa el operador de mutación definido en un caso concreto donde los cromosomas tienen longitud 5, la probabilidad de mutación es $p_m = 0.25$ y cuyas componentes son elementos del conjunto $\{1,2,\ldots,7\}$.

En el algoritmo 3 se muestra el pseudocódigo que define el operador de cruce utilizado. Y en el Anexo A, en el fragmento de código A.4 se muestra el código de la función *mutacion* que ejecuta el operador de mutación definido para el algoritmo genético.

Algoritmo 3: Operador de mutación

Datos: *individuo*, p_m
Resultado: *individuo_mutado*
individuo_mutado $\leftarrow$ *individuo*
para cada componente *i* de *individuo_mutado* **hacer**
  $x \leftarrow U(0,1)$
  si $x \leqslant p_m$ **entonces**
    cambiar el valor de la componente *i* por otro valor de conjunto $\{1,\ldots,|L|\}$ elegido aleatoriamente de forma equiprobable
  fin
fin
devolver *individuo_mutado*

3.2.6. Selección de supervivientes

Una vez obtenidos los hijos tras los diferentes cruces y mutaciones hay que seleccionar los individuos que pasan a la siguiente generación. En el algoritmo genético propuesto se establece que el criterio de selección de supervivientes es el conocido como criterio elitista, el cual consiste en seleccionar los N mejores individuos del conjunto formado por padres e hijos, siendo N es el tamaño de población en cada generación.

Algoritmo 4: Selección de supervivientes

Datos: *padres*, *hijos*, N
Resultado: *nueva_generacion*
generacion_actual $\leftarrow$ *padres* + *hijos*
nueva_generacion $\leftarrow$ N individuos con mejor *fitness* de la *generacion_actual*
devolver *nueva_generacion*

En el Anexo A, en el fragmento de código A.5 se muestra el código de la función *supervivientes* que ejecuta la selección de supervivientes definida para el algoritmo genético.

3.2.7. Algoritmo genético completo

En las subsecciones anteriores se han definido todos los elementos necesarios para la ejecución del algoritmo genético propuesto. En el Anexo A, en el fragmento de código A.6 se muestra el código de la función *genetico* que ejecuta el algoritmo genético propuesto.

Capítulo 4

Experiencia computacional

En este capítulo se presenta la experiencia computacional llevada a cabo para evaluar y comparar las diferentes técnicas de resolución del problema de determinación de precios con empates descritas en los capítulos anteriores. En primer lugar, se describe la colección de problemas utilizada para realizar la experiencia computacional. Sobre un conjunto de problemas de menor tamaño se evalúa el rendimiento del algoritmo genético en comparación con el modelo de resolución exacto atendiendo al valor óptimo obtenido y al tiempo de cálculo. Luego, se comparan varias configuraciones del algoritmo genético y finalmente, se utiliza la mejor de las configuraciones para resolver problemas de mayor tamaño.

Toda la experiencia computacional ha sido llevada a cabo en un ordenador con procesador Intel Core i7-4500U de 1.8 GHz, 8.00 GB de RAM y Windows 10 64-bits como sistema operativo.

La resolución exacta de los problemas considerados, utilizando la formulación (2.39) propuesta en el capítulo 2 y haciendo uso de técnicas de resolución de problemas de optimización lineal entera, se ha llevado a cabo utilizando la versión 20.1.0 de IBM ILOG CPLEX Optimization Studio con la configuración que este software comercial utiliza por defecto. El código programado en CPLEX se incluye en el anexo C.

Para la implementación del algoritmo genético descrito en el capítulo 3 se ha utilizado la versión 2022.07.2-576 del IDE RStudio junto con la versión 4.3.1 del lenguaje de programación R. También se han programado en lenguaje R una serie de funciones auxiliares para leer los ficheros de texto que contienen los datos de los problemas y para escribir en el formato adecuado los ficheros de datos que necesita CPLEX para resolver los problemas. El código de las funciones auxiliares programadas se incluye en el anexo B. El código programado en R para ejecutar el algoritmo genético sobre la batería de problemas se incluye en el anexo D.

4.1. Descripción del conjunto de problemas de prueba

La experiencia computacional hace uso de un banco de problemas creado y utilizado en la experiencia computacional realizada en [Domínguez et al., 2021] y que está disponible en el repositorio https://github.com/cdomsa/RPPT/. Esta colección de problemas está formada por un total de 365 problemas diferentes del *RPPT* diseñados atendiendo al número de clientes, el número de productos, el tamaño de las listas de preferencia de cada cliente y el número de empates que existen en las listas de preferencias de los clientes.

En esta memoria, para llevar a cabo la experiencia computacional, se han utilizado dos baterías de problemas. La primera batería, B_1, está formada por los 25 primeros problemas, que son los de menor tamaño en términos del número de clientes y de productos. La segunda batería de problemas, B_2, está formada por los 15 problemas siguientes dentro del banco de problemas, desde el 26 hasta el 40. A diferencia de los problemas de B_1, en los problemas de B_2 el número de productos aumenta a 25. Tanto los problemas de B_1 como los de B_2 pueden clasificarse en grupos, de 5 problemas cada uno, cuyas características se recogen en la tabla 4.1.

Tabla 4.1: Resumen de las características de los problemas utilizados para la experiencia computacional

Problemas	*Batería*	$\|K\|$	$\|I\|$	*Nº preferencias*	*Nº empates*
1-5	B_1	50	5	2	1
6-10	B_1	50	5	3	1
11-15	B_1	50	5	5	1
16-20	B_1	50	5	5	2
21-25	B_1	50	5	5	3
26-30	B_2	50	25	5	1
31-35	B_2	50	25	5	2
36-40	B_2	50	25	5	3

4.2. Comparación de la resolución exacta del modelo de un solo nivel y el algoritmo genético

En la tabla 4.2 se recogen los resultados obtenidos en la experiencia computacional con CPLEX sobre la batería B_1. Para cada uno de los 25 problemas, en la segunda columna se incluye el valor óptimo alcanzado para la función objetivo y en la tercera columna se muestra el tiempo en segundos que tarda CPLEX en alcanzar dicho valor óptimo. En las últimas columnas de la tabla se incluye la solución óptima que proporciona el valor de los índices de los presupuestos y los presupuestos que determinan el precio de cada producto.

CPLEX ha sido capaz de encontrar una solución óptima de cada problema en un tiempo razonable. El mayor tiempo de cálculo es 87 segundos y ocurre en el problema número 24, en el que hay un mayor número de empates. En la figura 4.1 se observa que cuanto más complejo es el problema, en relación al

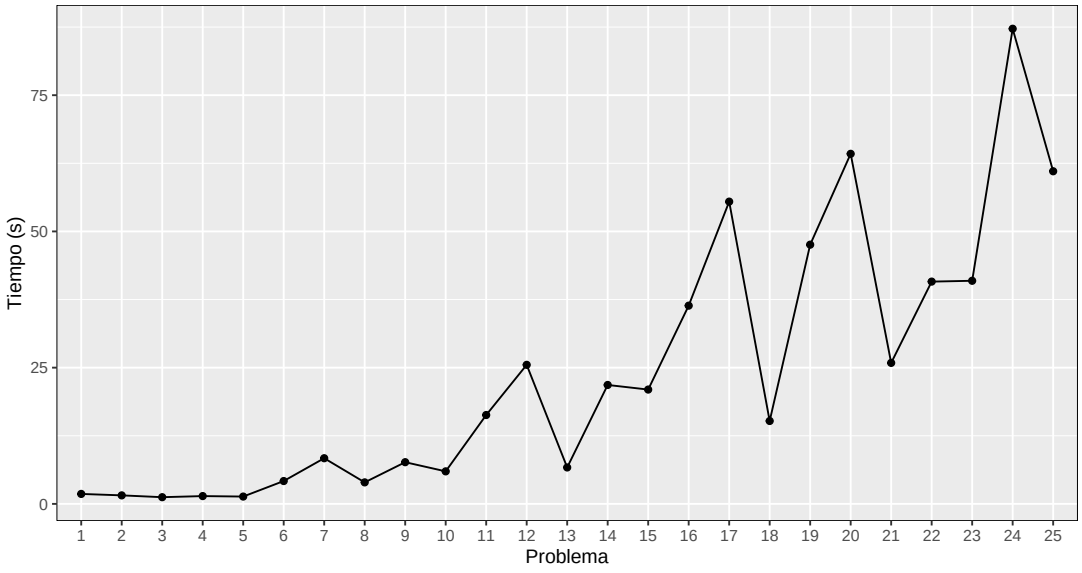


Figura 4.1: Tiempo en segundos que tarda CPLEX en encontrar una solución óptima para cada problema.

Tabla 4.2: Resultados de la experiencia computacional realizada con CPLEX sobre la batería B_1.

Problema	*Óptimo* $(u.m.)$	*Tiempo* (s)	*Índices de precios*					*Precios* $(u.m.)$				
			P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_5
1	1551	1.829	34	13	12	15	36	94	44	43	46	96
2	1802	1.564	40	19	25	21	39	95	52	66	57	92
3	1460	1.228	14	33	15	19	28	38	75	40	49	65
4	1545	1.432	21	14	27	41	40	49	34	62	99	94
5	1375	1.345	36	-	33	19	14	96	-	90	44	35
6	1796	4.194	27	36	17	13	25	72	96	48	44	66
7	1928	8.379	36	26	34	41	17	88	67	83	97	46
8	1641	3.949	34	15	17	37	29	83	40	43	89	66
9	1709	7.651	20	14	35	31	30	46	34	85	80	74
10	1519	5.970	36	14	33	18	28	96	35	90	43	71
11	1901	16.308	27	36	12	25	34	72	96	43	66	94
12	2023	25.517	34	25	37	15	32	83	66	90	40	79
13	1825	6.686	34	21	31	29	13	83	53	71	66	37
14	1867	21.823	26	13	31	41	38	61	33	80	99	91
15	1717	20.979	19	11	28	25	34	44	26	71	57	91
16	1867	36.378	26	36	20	35	11	68	96	54	95	39
17	1944	55.466	33	19	42	41	37	82	52	99	97	90
18	1773	15.215	38	13	22	34	29	91	37	55	83	66
19	1736	47.571	26	13	31	40	33	61	33	80	94	83
20	1583	64.241	33	14	28	25	34	90	35	71	57	91
21	1780	25.874	27	36	-	26	12	72	96	-	68	43
22	1866	40.790	35	19	42	42	32	87	52	99	99	79
23	1635	40.943	34	36	31	21	13	83	87	71	53	37
24	1727	87.191	29	13	35	40	26	68	33	85	94	61
25	1524	61.039	28	14	28	36	31	71	35	71	96	76

tamaño de las listas de productos de cada cliente y al número de empates presentes en dichas listas de preferencias, más tiempo le cuesta a CPLEX encontrar una solución óptima.

Como un algoritmo genético es un procedimiento estocástico, cada problema se resuelve 20 veces para observar si el procedimiento proporciona el mismo resultado, lo que da una idea de la robustez del algoritmo. Una primera configuración elegida considera un tamaño de población $N = 100$, un número de

Tabla 4.3: Resultados de la aplicación del algoritmo genético sobre la batería B_1

Problema	*Éxitos*	*%*	t_{min}	$\bar{t}$	t_{max}	it_{min}	$\overline{it}$	it_{max}	GAP_{min}	$\overline{GAP}$	GAP_{max}
1	17	85	1.042	1.876	3.613	22	39.6	92	0.645	1.139	2.128
2	19	95	0.879	1.662	3.057	19	39.2	69	0.111	0.111	0.111
3	17	85	0.726	1.677	2.775	18	37.8	67	1.507	1.507	1.507
4	20	100	1.185	2.376	4.079	24	51.5	76	0.000	0.000	0.000
5	5	25	0.782	2.207	3.210	16	51.2	84	0.291	2.337	4.000
6	6	30	1.463	2.686	4.995	40	57.3	93	0.111	0.334	0.557
7	12	60	1.203	1.736	2.544	35	53.2	81	1.660	1.679	1.815
8	12	60	1.244	2.063	2.838	37	62.9	87	1.341	2.689	3.352
9	16	80	0.969	1.776	2.576	27	52.6	81	0.351	0.439	0.527
10	3	15	2.479	2.685	2.881	77	83.7	90	0.263	2.157	3.818
11	8	40	1.310	2.293	3.520	39	59.5	88	0.421	0.649	0.894
12	5	25	1.602	2.190	3.181	42	65.4	98	0.198	0.623	1.137
13	12	60	1.174	1.915	2.847	31	55.2	82	0.110	0.343	0.438
14	14	70	1.406	1.970	2.721	40	57.8	82	0.857	1.053	1.446
15	8	40	0.924	1.597	2.344	24	44.2	63	0.175	0.456	2.155
16	10	50	1.183	1.859	2.765	33	55.3	87	0.054	0.102	0.214
17	9	45	0.827	1.663	2.087	23	46.1	63	0.103	0.272	0.412
18	18	90	0.982	1.690	2.517	27	45.7	71	3.046	3.638	4.230
19	19	95	1.231	2.248	3.255	36	66.7	99	0.173	0.173	0.173
20	16	80	0.911	1.660	3.134	25	45.2	67	0.126	0.158	0.190
21	19	95	0.758	1.468	2.911	19	42.6	85	0.056	0.056	0.056
22	8	40	1.125	2.050	3.063	33	62.2	97	0.161	0.250	0.375
23	19	95	1.312	1.900	3.078	33	56.3	96	0.061	0.061	0.061
24	6	30	1.284	1.604	2.051	37	47.0	61	0.174	0.339	0.926
25	20	100	0.763	1.361	2.000	20	39.1	59	0.000	0.000	0.000

iteraciones igual a 100 y una probabilidad de mutación $p_m = 0.25$.

En la tabla 4.3 se recogen los resultados obtenidos. Cada fila corresponde a un problema. Para cada problema, en la segunda columna se incluye el número de veces de las 20 ejecuciones en las que se ha alcanzado el óptimo. En estos problemas, el procedimiento exacto ha obtenido el valor óptimo y por eso está disponible para su comparación. En la tercera columna se muestra, el porcentaje de éxito que esto supone. Las columnas cuarta, quinta y sexta se refieren al mínimo, media y máximo, respectivamente, del tiempo total que el algoritmo genético ha requerido en las ejecuciones en las que proporciona la solución

óptima hasta finalizar todas las iteraciones. En las siguientes tres columnas se muestra el número mínimo, medio y máximo, respectivamente, de iteraciones que ha tardado el genético en alcanzar la solución óptima que proporciona en el caso de que la haya alcanzado.

Por último, cuando el algoritmo genético no encuentra una solución óptima, se define la magnitud *Gap* como el porcentaje que representa la diferencia entre el valor objetivo que proporciona la mejor solución obtenida por el genético, $\widetilde{Z}$, y el valor objetivo que proporciona una solución óptima del problema, Z^*, obtenido por la resolución exacta:

$$\%Gap = \frac{Z^* - \widetilde{Z}}{Z^*} \cdot 100$$

En las tres últimas columnas de la tabla 4.3 se calcula para las ejecuciones en las que el algoritmo genético no ha encontrado una solución óptima, el $\%Gap$ mínimo, medio y máximo, entre la mejor solución dada por el genético y la óptima encontrada por CPLEX.

A continuación, se muestran gráficamente los resultados dados en la tabla 4.3. En la figura 4.2 se representa gráficamente en color verde para cada uno de los problemas el número de ejecuciones en las que el algoritmo ha encontrado una solución óptima del problema. En gris se colorea el número de ejecuciones para las que no se ha encontrado solución óptima. Para todos los problemas ha habido ejecuciones que han alcanzado el óptimo aunque algunos han tenido más porcentaje de éxito que otros. Sólo en los problemas 4 y 25 se ha alcanzado la solución óptima en todas las iteraciones. Otros problemas con un gran porcentaje de éxito son el 2, 19, 21 y 23. Los problemas para los que se ha encontrado el óptimo un menor número de veces han sido el 5, 6, 10, 12 y 24. En media, sobre el total de ejecuciones, se ha alcanzado el óptimo el 63.6 % de las veces.

Con el número de veces que el algoritmo ha alcanzado o no ha alcanzado la solución óptima no se evalúa la robustez de algoritmo. Lo ideal sería que si el algoritmo no alcanza una solución óptima, se quede lo más cerca posible de alcanzar una. Para evaluar esto se usa la magnitud $\%Gap$ definida anteriormente.

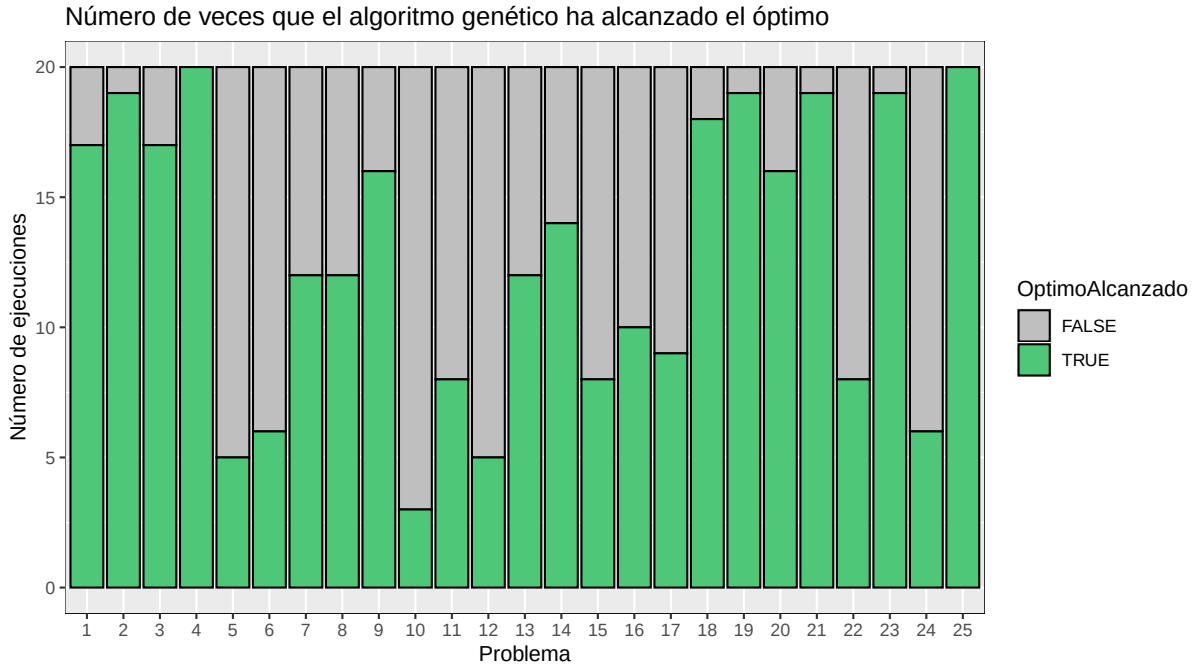


Figura 4.2: Número de ejecuciones en las que el algoritmo genético ha encontrado el óptimo de cada problema. En total se han realizado 20 ejecuciones del algoritmo sobre cada uno de los problemas.

En la figura 4.3 se representa gráficamente en forma de diagramas de cajas para cada uno de los problemas el $\%Gap$ de las mejores soluciones obtenidas en las ejecuciones en las que no se ha alcanzado

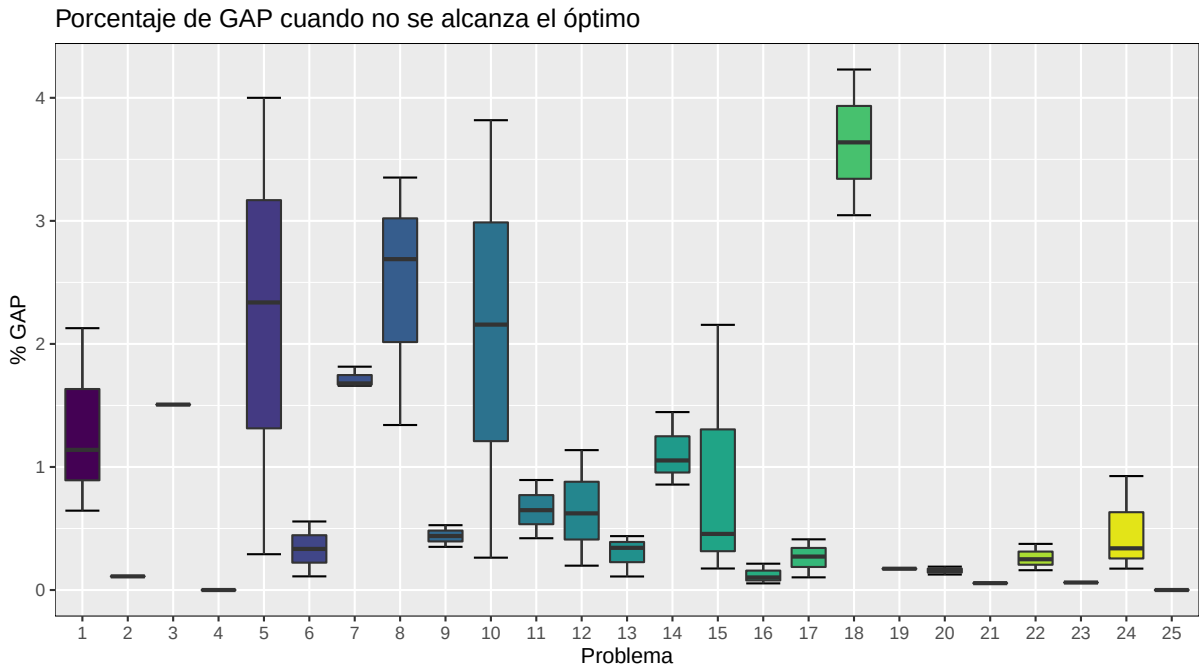


Figura 4.3: Análisis del *%Gap* entre la solución óptima obtenida por CPLEX y la mejor solución obtenida por el algoritmo genético para cada problema en las ejecuciones donde este no ha alcanzado el óptimo.

el óptimo. En la figura se han incluido los problemas 4 y 25, aunque en todas las ejecuciones realizadas siempre se ha alcanzado el óptimo y por eso se les asigna un *gap* de 0. Se observa que en algunos problemas presentan mayor dispersión que otros en cuanto al *%Gap*, significando que el algoritmo genético es menos robusto, ya que hay una mayor variabilidad en los resultados obtenidos.

Los problemas que mayor dispersión presentan son el 1, 5, 8, 10, 15, 18 y los problemas que menos dispersión presentan son el 2, 3, 19, 21 y 23. Cuando no se alcanza el óptimo, el problema que siempre se queda más lejos es el 18 y los que se quedan más cerca son el 21 y el 23. No obstante es interesante observar que siempre que no se alcanza el óptimo, el *%Gap* de la mejor solución obtenida para cualquier problema es menor del 4.5 %, lo que pone de manifiesto la robustez del algoritmo ya que este proporciona soluciones óptimas o soluciones factibles muy cercanas a ser óptimas.

En la figura 4.4 se compara el tiempo en segundos que tarda CPLEX en resolver cada problema con el tiempo medio que tarda el algoritmo genético en realizar las 100 iteraciones sobre cada problema. Se observa que en media el genético tarda mucho menos en resolver cada problema, lo que supone una ventaja con respecto a CPLEX cuando se incrementa la dificultad de este problema. En algunos ensayos preliminares con problemas más grandes se ha observado que estas diferencias son más notables al aumentar el número de clientes y/o productos.

En la tabla 4.4 se hace una comparación entre la solución óptima dada por el algoritmo genético y la solución óptima proporcionada por CPLEX. Las diferencias que se observan en cada problema son debidas a la existencia de óptimo múltiple. Por ejemplo, en el primer problema al algoritmo genético devuelve la solución $(37, 13, 12, 15, 34)$ y CPLEX devuelve la solución $(34, 13, 12, 15, 36)$. En este problema ningún cliente compra los productos 1 y 5 por lo que basta asignarles un precio lo suficientemente alto como para que ningún cliente los compre sin importar el precio.

4.3. Evaluación de varias configuraciones del algoritmo genético

Tras haber realizado la experiencia computacional con al algoritmo genético, se observa que los resultados son mejorables en términos del valor de la función objetivo. Como modificar el diseño del algoritmo genético y proponer un algoritmo metaheurístico más ajustado al tipo de problema que se pre-

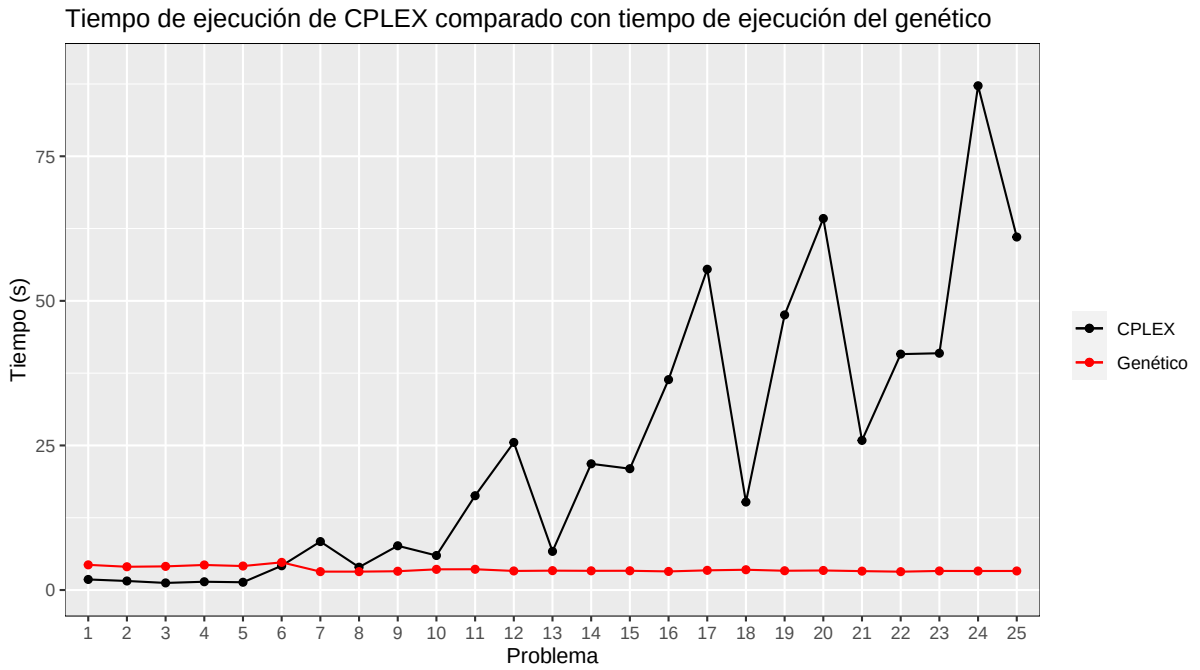


Figura 4.4: Comparación entre el tiempo en segundos que tarda CPLEX en resolver un problema frente al tiempo medio de resolución del mismo problema por el algoritmo genético.

tenden resolver escapa del alcance de este TFM y requeriría bastante trabajo adicional, en este apartado se analiza si hay otra configuración de parámetros más eficiente sin modificar el diseño del algoritmo.

Se propone un diseño factorial 2^3 para analizar la importancia de cada uno de los parámetros que determinan la configuración el algoritmo genético. En particular, se consideran los tres siguientes parámetros como factores del diseño:

- Tamaño de población con valores $N = 100$ y $N = 200$.
- Número de iteraciones realizadas con valores 100 y 200.
- Probabilidad de mutación con valores $p_m = 0.05$ y $p_m = 0.25$.

En total se han considerado 8 configuraciones posibles del algoritmo. Cada una de estas configuraciones se ha ejecutado 10 veces sobre cada uno de los problemas de la batería B_1. De cada ejecución, se han extraído los siguientes datos, que se podrán utilizar como variables respuesta del diseño:

- Tiempo total de ejecución en segundos.
- Mejor valor objetivo encontrado.
- Primera iteración en la que ha encontrado el mejor valor objetivo.
- Tiempo en segundos que ha tardado en encontrar por primera vez el mejor valor objetivo.

Además, se ha almacenado la primera solución encontrada que proporciona el mejor valor objetivo.

En la figura 4.5, se muestran los gráficos de interacciones entre los factores que contribuyen a elegir la configuración más ventajosa en términos de tres medidas diferentes del comportamiento del algoritmo, el tiempo medio de ejecución en segundos, el porcentaje medio de *gap* obtenido en las ejecuciones y el porcentaje de ejecuciones en las que el algoritmo genético alcanza el óptimo.

En la figura 4.6 se representan las interacciones de los factores dos a dos sobre el tiempo total de ejecución en segundos del algoritmo. Lo único reseñable es la interacción entre el número de iteraciones y el tamaño de la población, lo cual era esperable. Cuando el tamaño de población aumenta, el pasar de

Tabla 4.4: Comparativa entre una solución óptima proporcionada por el algoritmo genético y la solución óptima proporcionada por CPLEX para cada uno de los problemas.

(a) Algoritmo genético

Problema	*Índices de precios*				
	P_1	P_2	P_3	P_4	P_5
1	37	13	12	15	34
2	40	19	25	21	39
3	14	29	15	19	28
4	21	14	27	40	41
5	37	36	33	19	14
6	27	36	17	13	25
7	36	26	34	41	17
8	34	15	17	37	29
9	20	14	35	31	30
10	36	14	33	18	28
11	27	36	12	25	34
12	34	25	37	15	32
13	34	21	31	29	13
14	26	13	31	41	38
15	19	11	28	25	34
16	26	36	20	35	11
17	33	19	41	41	37
18	38	13	22	34	29
19	26	13	31	40	33
20	33	14	28	25	34
21	27	36	37	26	12
22	35	19	42	42	32
23	34	36	31	21	13
24	29	13	35	40	26
25	36	14	28	30	31

(b) CPLEX

Problema	*Índices de precios*				
	P_1	P_2	P_3	P_4	P_5
1	34	13	12	15	36
2	40	19	25	21	39
3	14	33	15	19	28
4	21	14	27	41	40
5	36	-	33	19	14
6	27	36	17	13	25
7	36	26	34	41	17
8	34	15	17	37	29
9	20	14	35	31	30
10	36	14	33	18	28
11	27	36	12	25	34
12	34	25	37	15	32
13	34	21	31	29	13
14	26	13	31	41	38
15	19	11	28	25	34
16	26	36	20	35	11
17	33	19	42	41	37
18	38	13	22	34	29
19	26	13	31	40	33
20	33	14	28	25	34
21	27	36	-	26	12
22	35	19	42	42	32
23	34	36	31	21	13
24	29	13	35	40	26
25	28	14	28	36	31

100 a 200 iteraciones incrementa mucho más el tiempo de cálculo del algoritmo genético. En las dos combinaciones de dos factores no se observan interacciones.

En la figura 4.7 se representan las interacciones de los factores dos a dos sobre el porcentaje de *%Gap* obtenido en las soluciones proporcionadas por el genético. Se puede concluir que los mejores resultados en cualquier caso se producen con tamaños de población 200. En este caso, la mayor interacción se observa también entre el número de iteraciones y el tamñao de población. Cuando se aumenta el número de iteraciones la mejora en el *%Gap* es más acusada cuando se tiene un tamaño de población $N = 100$ que con población 200. Esto significa, que en el caso de considerar 100 iteraciones, es importante elegir un tamaño de población 200.

Por último, en la figura 4.8 se representan las interacciones de los factores dos a dos sobre el porcentaje de veces que el genético ha alcanzado una solución óptima. De nuevo, la mayor interacción se observa entre el número de iteraciones y el tamaño de población. Con un tamaño de población 200, no es tan importante realizar 200 iteraciones, como lo es cuando el tamaño es 100. Esto último tiene sentido, ya que si se le permite al algoritmo genético trabajar con más soluciones en cada generación y explorar durante más generaciones, cabe esperar que el porcentaje de éxito de encontrar una solución óptima sea mayor.

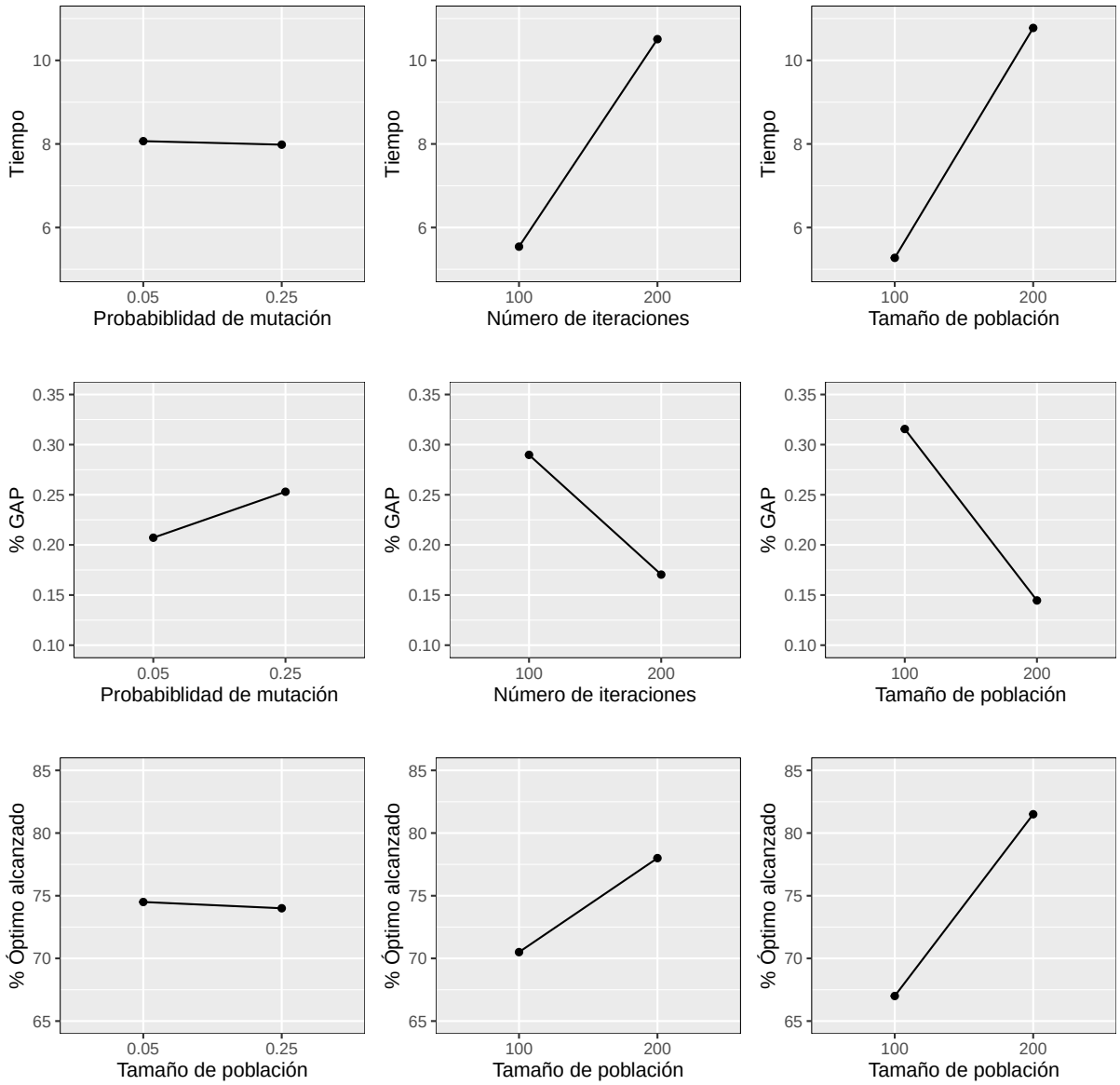


Figura 4.5: Gráficos de efectos principales de los factores *probabilidad de mutación*, *número de iteraciones* y *tamaño de población* sobre el tiempo medio de ejecución en segundos, el porcentaje medio de *gap* obtenido en las ejecuciones y el porcentaje de ejecuciones en las que el algoritmo genético alcanza el óptimo.

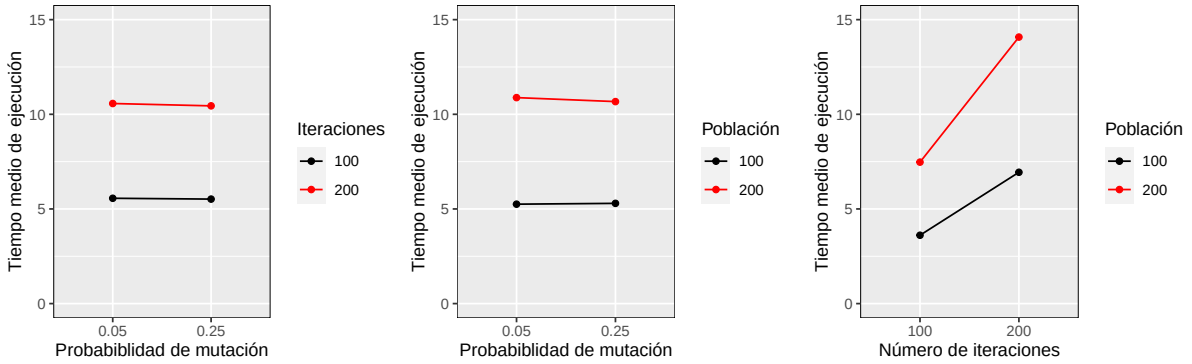


Figura 4.6: Gráficos de interacción entre los factores *probabilidad de mutación*, *número de iteraciones* y *tamaño de población* para la media del tiempo de ejecución del algoritmo genético.

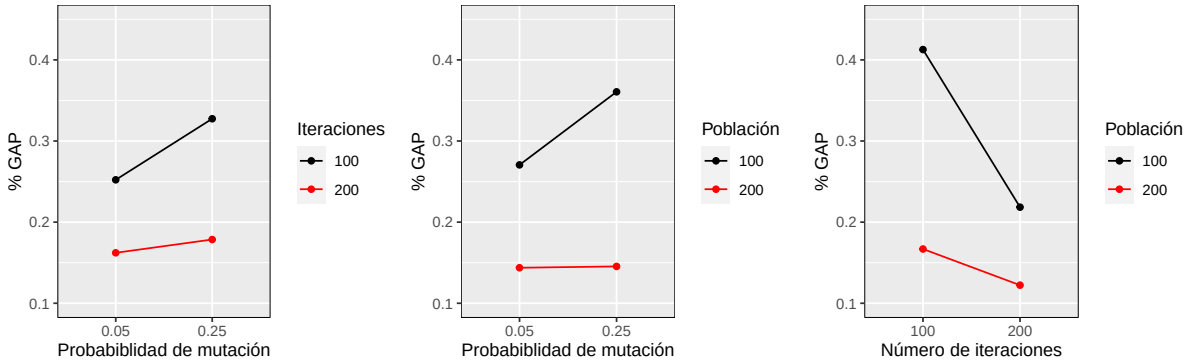


Figura 4.7: Gráficos de interacción entre los factores *probabilidad de mutación*, *número de iteraciones* y *tamaño de población* para el porcentaje medio de *gap* obtenido en las ejecuciones del algoritmo genético.

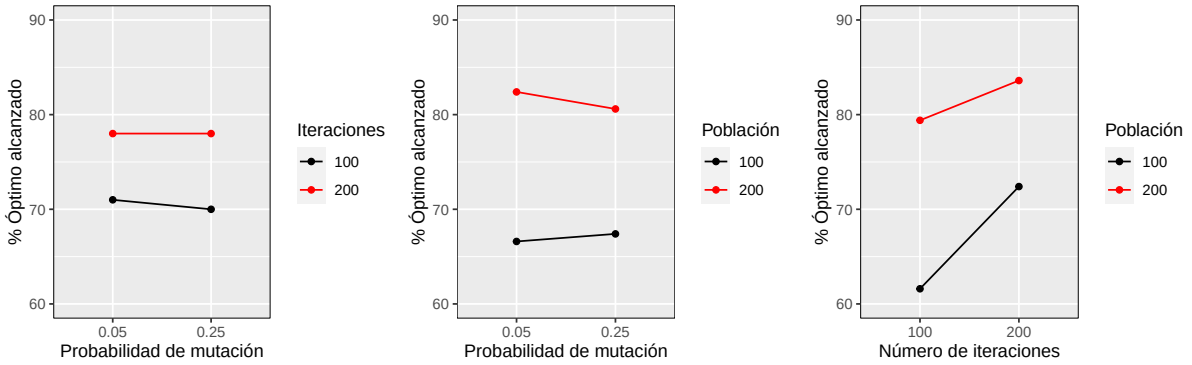


Figura 4.8: Gráficos de interacción entre los factores *probabilidad de mutación*, *número de iteraciones* y *tamaño de población* para el porcentaje medio de ejecuciones del algoritmo genético en las que se alcanza el óptimo.

4.4. Comportamiento del algoritmo genético en problemas de mayor tamaño

Tras la experiencia computacional llevada a cabo con los problemas de la batería B_1, cabe preguntarse cual es el comportamiento del algoritmo genético en problemas de mayor tamaño, ya que se espera que la resolución exacta de estos problemas lleve más tiempo. Para evaluar este comportamiento se hace uso de la batería de problemas B_2 descrita al principio del capítulo.

En primer lugar, se han resuelto todos los problemas de manera exacta con CPLEX obteniendo para cada uno de ellos un valor óptimo, salvo para el problema número 40, para el que CPLEX ha devuelto el valor objetivo de la mejor solución obtenida tras 600 segundos de ejecución. Este valor objetivo tiene un *gap* del 0.05 % con respecto a la mejor cota del valor óptimo conocida hasta ese momento. Para cada problema se obtiene también el tiempo que tarda CPLEX en encontrar un valor óptimo (o un mejor valor objetivo en su defecto). A continuación se ha ejecutado el algoritmo genético 10 veces sobre cada uno de los problemas con la configuración de parámetros tamaño de población $N = 200$, probabilidad de mutación $p_m = 0.25$ y número de iteraciones igual a 200. Se ha registrado el mejor valor encontrado al final de la ejecución y el tiempo que tarda en acabar. En la tabla 4.5 se muestran algunos de los datos recogidos de esta experiencia computacional.

Tabla 4.5: Tabla comparativa entre la resolución exacta y la resolución con el algoritmo genético de los problemas de la batería B_2.

Problema	t_{min}	$\bar{t}$	t_{max}	sol_{min}	$\overline{sol}$	sol_{max}	sol_{CPLEX}	t_{CPLEX}
26	17.54	20.67	23.77	1900	1928.20	1995	2142	27.84
27	17.49	20.73	24.17	1977	2017.90	2047	2211	41.34
28	16.97	19.75	24.62	1862	1884.90	1898	2047	28.67
29	20.29	25.96	44.00	2066	2100.00	2139	2310	25.10
30	17.39	25.01	38.54	2359	2379.20	2442	2575	19.23
31	19.63	21.01	24.91	1840	1880.60	1921	2053	42.08
32	19.50	21.34	23.29	1871	1907.30	1939	2099	118.06
33	17.68	18.25	18.69	1833	1864.20	1907	2028	28.84
34	17.41	18.83	21.23	1981	2007.30	2054	2212	43.78
35	17.86	19.82	23.50	2207	2242.90	2267	2446	113.14
36	17.66	19.50	22.21	1749	1784.40	1826	2013	183.74
37	17.31	19.66	23.73	1821	1843.20	1863	2059	542.77
38	16.86	19.90	25.04	1613	1655.40	1684	1813	485.91
39	18.41	19.80	23.42	1854	1898.50	1951	2093	191.64
40	19.03	20.22	24.38	2057	2093.30	2125	*2312	600.24

En ningún caso el algoritmo genético alcanza el valor óptimo para el problema y solo alcanza el mejor valor encontrado en 1 de las 10 ejecuciones. Aún así, la variabilidad en el valor objetivo de las soluciones encontradas no es grande lo que permite afirmar que el algoritmo es robusto aunque no muy eficiente a la hora de alcanzar el óptimo.

De nuevo, la ventaja del uso del algoritmo genético frente a la resolución exacta es el tiempo de

ejecución ya que en cuanto el problema se complica al aumentar el número de empates, la resolución exacta requiere mucho más tiempo. En la figura 4.9 se muestra gráficamente la comparativa entre los tiempos de ejecución de CPLEX y del algoritmo genético para cada problema.

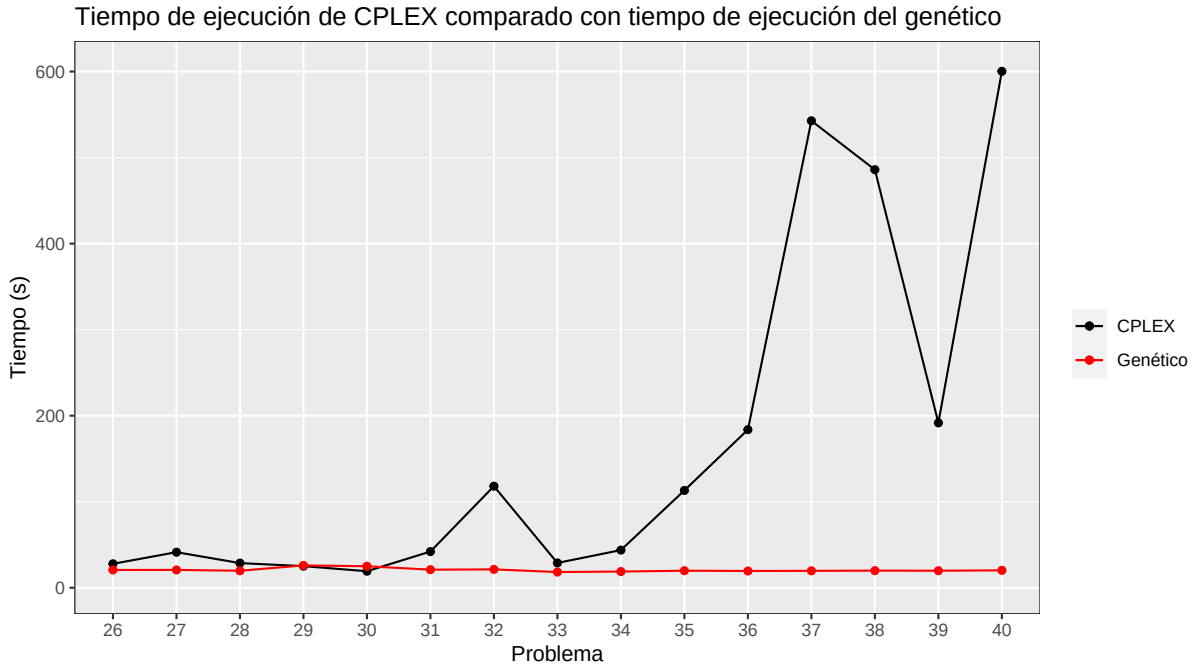


Figura 4.9: Comparativa de tiempos de ejecución de la resolución exacta con CPLEX y de la resolución con el algoritmo genético de la batería de problemas B_2

En la figura 4.10 se muestra el porcentaje de *gap* obtenido en las diferentes ejecuciones del algoritmo genético en comparación con el valor objetivo conseguido por CPLEX para cada uno de los problemas. Se observan algunas diferencias para algunos problemas concretos como el 36 y el 37 que son los que tienen siempre mayores valores de *gap*. Pero en general, para todos los problemas se obtienen valores de *gap* que oscilan entre el 6 % y el 12 %. Esto ilustra la robustez del algoritmo incluso cuando los problemas aumentan de tamaño o se complican en cuanto a la existencia de mayor número de empates.

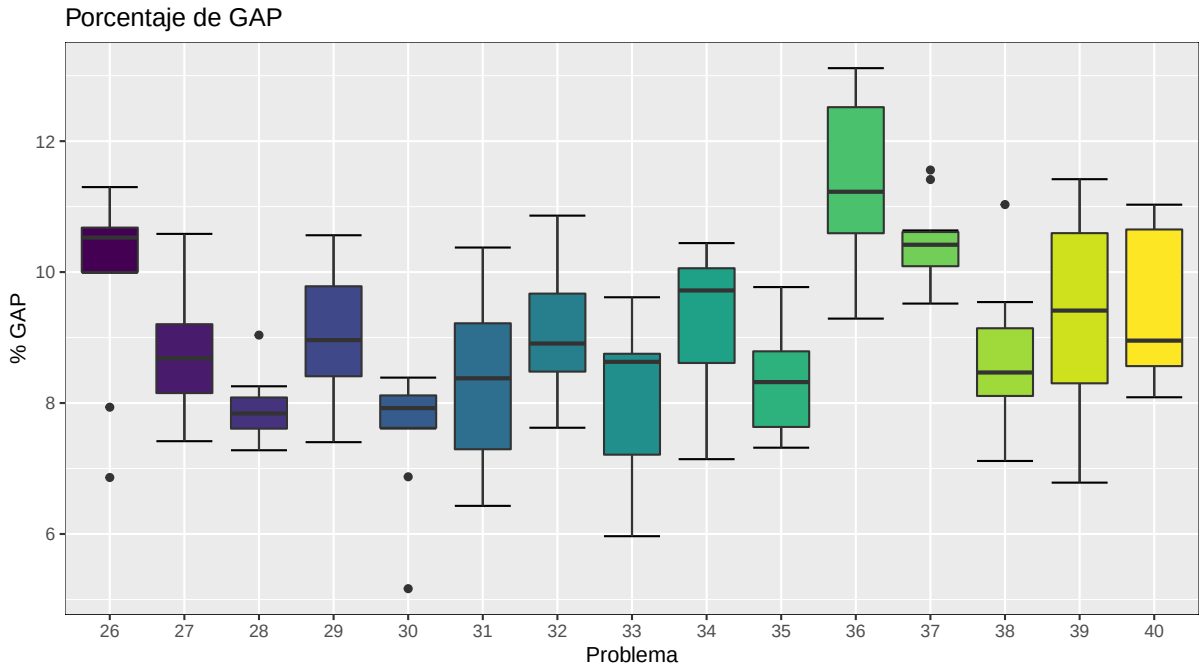

Porcentaje de GAP
% GAP
12
10
8
6
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
Problema

Figura 4.10

Capítulo 5

Conclusiones y trabajo futuro

La contribución más importante del TFM ha sido proponer una formulación alternativa a las que se han estudiado en la literatura para el *RPPT* desde un enfoque binivel considerando una aproximación pesimista. La reformulación del problema como un modelo de un solo nivel ha requirido el uso de técnicas de optimización clásica, como resultados de la teoría de la dualidad, como el manejo de un problema biobjetivo. Esta reformulación ha permitido resolver de manera exacta y con éxito problemas de pequeño tamaño (50 clientes y 5 productos) con diferentes características en términos del tamaño de las listas de preferencias de los clientes y del número de empates. Estos problemas pequeños se han resuelto con el software comercial CPLEX en menos de 87 segundos. En problemas más grandes (50 clientes y 25 productos) la resolución exacta supone tiempos de ejecución significativamente mayores debido al incremento en el número de restricciones del modelo cuyo orden depende del número de productos y número de clientes.

Esto sugiere, como trabajo futuro, la necesidad de trabajar en la mejora de la formulación de un solo nivel binaria pura (2.39) para evitar estos inconvenientes y poder resolver problemas de mayor tamaño en tiempos razonables. Para ello, una estrategia a seguir es introducir las restricciones de manera dinámica con ayuda de las funciones *call-back* de CPLEX en concreto de las *lazy constraints*. Las *lazy constraints* son parte del conjunto de restricciones del modelo. CPLEX resuelve en principio un modelo incompleto prescindiendo de estas restricciones y las utiliza antes de devolver cualquier solución del modelo para verificar si la solución obtenida las cumple o no. De esta manera, el tiempo de cómputo se reduce considerablemente. Otra posible estrategia a seguir es la búsqueda de desigualdades válidas que permitan fortalecer la formulación y hacerla más eficiente.

Otra de las aportaciones notables de este TFM ha sido proponer un algoritmo genético para resolver el *RPPT*. El uso de este algoritmo permite resolver problemas de forma más rápida que con la formulación exacta, lo cual supone una gran ventaja. De hecho, el algoritmo genético puede utilizarse para hacer una búsqueda de una solución factible cercana al óptimo en caso de que este no se alcance.

Como propuesta de mejora se puede destacar, en primer lugar, una línea de trabajo que consistiría en adaptar el algoritmo a la naturaleza del problema que se está resolviendo ya que el algoritmo propuesto es un algoritmo genético general que no aprovecha la estructura intrínseca del problema. Concretamente, se podría centrar la atención en el desarrollo de operadores de cruce y mutación adaptados. Otra propuesta de mejora puede ser incluir un pre-procesamiento de los datos que permita reducir la combinatoria de precios posibles para los productos, lo que aceleraría el algoritmo.

A pesar de todas estas posibles mejoras del algoritmo hay que destacar que, aún no siendo un algoritmo muy desarrollado y adaptado al problema, el comportamiento que ha tenido a la hora de resolver los problemas ha sido bastante bueno. Para los problemas de la batería B_1 el porcentaje de éxito de alcanzar el óptimo ha sido, en media, sobre el total de ejecuciones, de un 63.6 %; siendo mayor del 85 % en 9 de los 25 problemas. Para los problemas de la batería B_2, a pesar de no haber alcanzado el óptimo en ningún caso, el algoritmo ha sido capaz de encontrar siempre una solución con un $\%Gap$ de entre un 6 % y un 12 % en general. Este comportamiento del algoritmo da lugar a pensar que si se introducen todas las mejoras mencionadas anteriormente, se obtendría un algoritmo muy competitivo.

Bibliografía

[Calvete et al., 2019] Calvete, H. I., Domínguez, C., Galé, C., Labbé, M., and Marín, A. (2019). The rank pricing problems: Models and branch-and-cut algorithms. *Computers and Operations Research*, 105:12–31.

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

[Calvete and Galé, 2020] Calvete, H. I. and Galé, C. (2020). Algorithms for linear bilevel optimization. In *Bilevel Optimization*, chapter 10, pages 293–312. Springer.

[Calvete et al., 2021] Calvete, H. I., Galé, C., Iranzo, J. A., and Mateo, P. M. (2021). A decision tool based on bilevel optimization for the allocation of water resources in a hierarchical system. *International Transactions in Operational Research*.

[Dempe and Zemkoho, 2020] Dempe, S. and Zemkoho, A. (2020). *Bilevel Optimization*, volume 161 of *Springer Optimization and Its Applications*. Springer, 1 edition.

[Domínguez et al., 2021] Domínguez, C., Labbé, M., and Marín, A. (2021). The rank pricing problems with ties. *European Journal of Operations Research*, 294:492–506.

[Hernando, 2022] Hernando, L. (2022). Apuntes de la asignatura "Algoritmos bioinspirados y técnicas de computación evolutiva" impartida en el curso 2021/2022 en el Máster Universitario en Modelización e Investigación matemática, Estadística y Computación de la Universidad de Zaragoza.

[Kleinert et al., 2020] Kleinert, T., Labbé, M., Plein, F., and Schmidt, M. (2020). Technical note—there's no free lunch: On the hardness of choosing a correct Big-M in bilevel optimization. *Operations Research*, 68(6):1716–1721.

[Labbé and Violin, 2016] Labbé, M. and Violin, A. (2016). Bilevel programming and price setting problems. *Annals of Operation Research*, 240:141–169.

[Rusmevichientong et al., 2006] Rusmevichientong, P., Van Roy, B., and Glynn, P. W. (2006). A nonparametric approach to multiproduct pricing. *Operation Research*, 54(1):82–98.

[Shen et al., 2012] Shen, S., Smith, J. C., and Goli, R. (2012). Exact interdiction models and algorithms for disconnecting networks via node deletions. *Discrete Optimization*, 9:172–188.

[Wolsey, 2021] Wolsey, L. A. (2021). *Integer Optimization*. Wiley, 2 edition.

Anexos

Anexo A

Código R para el algoritmo genético

```r
compra <- function(n, solucion, datos_problema){
  # Esta función permite determinar que producto compra un cliente n
  # dada una combinación de precios para una serie de productos dentro
  # del marco de un problema de determinación de precios con empates.

  # Argumentos:
    # n: cliente
    # solucion: combinación de precios para los productos del problema
    # datos problema: datos que definen el problema de determinación de
    # precios

  # se extraen los datos que definen el problema
  productos <- c(1:datos_problema[["I"]])
  b <- datos_problema[["B"]]
  b.dist <- datos_problema[["B.dist"]]
  s <- datos_problema[["P"]]

  if(sum(b[n] >= b.dist[solucion[productos]]) == 0) {
    # si todos los precios son mayores que el presupuesto del cliente,
    # entonces el cliente no compra nada
    compra <- NA
  } else {
    # si existe algún precio menor o igual que el presupuesto del cliente
    # entonces se extraen las satisfacciones de los productos que puede
    # comprar el cliente
    sat_puede <- s[n, which(b[n] >= b.dist[solucion[productos]])]
    if(any(sat_puede != 0)){
      # si existe algún producto que puede comprar con satisfacción
      # distinta de 0 entonces:
      # se extraen los productos que puede comprar el cliente n y que
      # le proporcionan  máxima satisfacción
      mayor_sat <- which(s[n,] == max(sat_puede))
      # se extrae el producto de precio mínimo de entre los que puede
      # comprar y le  proporcionan máxima satisfacción
      compra <- mayor_sat[which.min(solucion[mayor_sat])]
      # este es por tanto el producto que compra el cliente n
    } else {
      # si todos los productos que puede comprar el cliente le
      # proporcionan satisfacción 0, entonces el cliente no compra nada
      compra <- NA
    }
  }
  return(compra)
}
```

Fragmento de código A.1: Función programada en R que determina el producto que compra un cliente en base a sus criterios dada una combinación de precios para los productos.

```r
eval.fitness <- function(solucion, datos_problema){
  # Esta función permite determinar el beneficio obtenido por la venta
  # de una serie de productos una vez determinados sus precios dentro
  # del marco de un problema de determinación de precios con empates.

  # Argumentos:
    # solucion: combinación de precios para los productos del problema
    # datos problema: datos que definen el problema de determinación de precios

  # se extraen los datos que definen el problema
  clientes <- c(1:datos_problema[["K"]])
  productos <- c(1:datos_problema[["I"]])
  b <- datos_problema[["B"]]
  b.dist <- datos_problema[["B.dist"]]
  s <- datos_problema[["P"]]

  # se determina el producto que compra cada cliente
  compras <-
    unlist(
      lapply(X = clientes, FUN = compra,
             solucion = solucion,
             datos_problema = datos_problema)
    )

  # se calcula el beneficio por los productos vendidos
  beneficio <- sum(b.dist[solucion[compras]], na.rm = TRUE)

  return(beneficio)
}
```

Fragmento de código A.2: Función programada en R que calcula el beneficio obtenido por el nivel superior cuando establece unos precios determinados.

```r
cruce <- function(individuo1, individuo2){
  # Esta función es el operador de cruce para el algoritmo genético.
  # Es el operador conocido como one point crossover. En él participan
  # dos individuos como padres y se forman dos hijos.

  # Argumentos:
    # individuo1: primer individuo que actúa como padre en el cruce
    # individuo2: segundo individuo que actúa como padre en el cruce

  longitud <- length(individuo1)
  # se selecciona una componente aleatoria n común para los dos individuos
  pos <- sample(x = c(1:longitud), size = 1)
  if(pos == longitud){
    # si n es igual a la longitud, los hijos que son idénticos a los padres
    hijo1 <- individuo1
    hijo2 <- individuo2
  } else {
    # en caso de que n sea menor que la longitud de los individuos:
    # el primer hijo se forma combinando las n primeras componentes del primer
    # padre junto con las componentes desde la (n+1)-ésima del segundo padre
    hijo1 <- c(individuo1[1:pos], individuo2[(pos + 1):longitud])
    # el segundo hijo se forma combinando las n primeras componentes del segundo
    # padre junto con las componentes desde la (n+1)-ésima del primer padre
    hijo2 <- c(individuo2[1:pos], individuo1[(pos + 1):longitud])
  }
  hijos <- rbind(hijo1, hijo2)
  return(hijos)
}
```

Fragmento de código A.3: Función programada en R que ejecuta el operador de cruce definido para el algoritmo genético.

```r
mutacion <- function(individuo, pm, conjunto){
  # Esta función es el operador de mutación para el algoritmo genético

  # Argumentos:
    # individuo: el individuo que se quiere mutar
    # pm: probabilidad de mutación
    # conjunto: conjunto de elementos con los que se forma un cromosoma

  # para cada componente del vector
  for(j in c(1:length(individuo))){
    # si mutamos
    if(runif(1) < pm){
      # cambiamos la componente del vetor
      individuo[j] <- sample(x = conjunto, size = 1)
    }
  }
  # se devuelve el individuo mutado
  return(individuo)
}
```

Fragmento de código A.4: Función programada en R que ejecuta el operador de mutación definido para el algoritmo genético.

```r
supervivientes <- function(poblacion, fitness, matriz.hijos, fitness.hijos,
                           tam.pobla){
  # Esta función ejecuta la selección de supervivientes de una generación
  # a otra en el algoritmo genético. La selección es elitista, se elige
  # a los mejores individuos.

  # Argumentos:
    # población: población actual
    # fitness: valores de fitness de los individuos de la poblacion actual
    # matriz.hijos: hijos obtenidos tras los cruces y las mutaciones
    # fitness.hijos: valores de fitness de los hijos

  # se juntan los individuos de la población actual junto a los hijos
  # que se han obtenido tras los cruces y las mutaciones
  generacion <- rbind(poblacion, matriz.hijos)
  # se juntan los valores de fitness de los individuos de la generacion
  fitness.generacion <- rbind(fitness, fitness.hijos)
  # se calcula el orden de los individuos de la generación en función
  # de la fitness
  orden <- order(fitness.generacion, decreasing = TRUE)
  # se seleccionan los mejores individuos de la generación para formar
  # la nueva población
  poblacion_ordenada <- generacion[orden[1:tam.pobla],]
  # se selecciona la fitness de los individuos seleccionados
  fitness_ordenada <- as.matrix(fitness.generacion[orden[1:tam.pobla]],
                                ncol = 1)

  return(list("poblacion" = poblacion_ordenada,
              "fitness" = fitness_ordenada))
}
```

Fragmento de código A.5: Función programada en R que ejecuta la selección de supervivientes definida para el algoritmo genético.

```r
genetico <- function(problema, tam.pobla, num.padres, pm, num.iter){
  # Esta función ejecuta un algoritmo genético para resolver un problema de
  # determinación de precion con empates dados los datos que lo definen.

  # Argumentos:
    # problema: datos que definen el problema de determinación de precios
    # tam.pobla: tamaño de la población que pasa de una generación a otra
    # num.padres: número de padres que participan en el cruce en cada generación
    # pm: probabilidad de mutación
    # num.iter: número de iteraciones (generaciones) que ejecuta el algoritmo

  # funciones y operadores necesarios para el algoritmo
  source("compra.R")
  source("fitness.R")
  source("cruce.R")
  source("mutacion.R")
  source("supervivientes.R")

  n <- problema[["problema"]]
  # optimo_cplex <- round(tabla_cplex[n, c("Optimo")])

  # tiempo inicial
  tiempo_inicial <- Sys.time()

  # Datos del problema de precios
  num.clientes <- problema[["K"]]
  num.productos <- problema[["I"]]
  s <- problema[["P"]]
  b <- problema[["B"]]
  clientes <- c(1:num.clientes)
  productos <- c(1:num.productos)
  b.dist <- sort(unique(b))

  if(num.padres%%2 == 1){
    num.padres <- num.padres - 1
  }

  # Inicialización de la población inicial (aleatoria)
  poblacion <- matrix(0, nrow = tam.pobla, ncol = num.productos)
  for(i in c(1:tam.pobla)){
    poblacion[i,] <- sample(x = c(1:length(b.dist)), size = num.productos,
                            replace = TRUE)
  }

  # Evaluación de la población inicial
  fitness <-  as.matrix(apply(X = poblacion, MARGIN = 1, FUN = eval.fitness,
                              datos_problema = problema))

  best_fitness <- 0
  iter_best <- 0
  tiempo_best <- 0
  # iteracion_optimo <- NA
  # tiempo_optimo <- NA
  # Bucle principal (criterio de parada: número de iteraciones)
  for(iteracion in c(1:num.iter)){

    # Bucle de cruce para generar los hijos (Se generan 100 hijos)
    matriz.hijos <- matrix(0, nrow = num.padres, ncol = num.productos)
    for (kk in c(1:(num.padres/2))){
      # Seleccion de padres
      # Método: por torneo (se cogen 2 y se selecciona el mejor)
      candidatos <- sample(x = c(1:tam.pobla), size = 2, replace = TRUE)
      padre1 <- candidatos[order(fitness[candidatos], decreasing = TRUE)[1]]
```

```
      candidatos <- sample(x = c(1:tam.pobla), size = 2, replace = TRUE)
      padre2 <- candidatos[order(fitness[candidatos], decreasing = TRUE)[1]]
      # Cruce
      hijos <- cruce(individuo1 = poblacion[padre1,],
                     individuo2 = poblacion[padre2,])
      # Se guardan los hijos obtenidos en matriz.hijos
      matriz.hijos[((kk - 1)*2 + 1):((kk - 1)*2 + 2),] <- hijos
    }

    # Mutación
    matriz.hijos <- t(apply(X = matriz.hijos, MARGIN = 1, FUN = mutacion,
                            pm = 0.25, conjunto = c(1:length(b.dist))))

    # Evaluacion de hijos
    fitness.hijos <-  as.matrix(apply(X = matriz.hijos, MARGIN = 1,
                                      FUN = eval.fitness,
                                      datos_problema = problema))

    # Selección de supervivientes para la siguiente generación
    superv <- supervivientes(poblacion = poblacion,
                             fitness = fitness,
                             matriz.hijos = matriz.hijos,
                             fitness.hijos = fitness.hijos,
                             tam.pobla = tam.pobla)
    poblacion <- superv[["poblacion"]]
    fitness <- superv[["fitness"]]

    if(max(fitness) > best_fitness){
      best_solucion <- poblacion[which.max(fitness),]
      best_fitness <- fitness[which.max(fitness)]
      iter_best <- iteracion
      tiempo_best <- as.numeric(difftime(time1 = Sys.time(),
                                         time2 = tiempo_inicial,
                                         units = "secs"))

    }

    # if(max(fitness) == optimo_cplex & is.na(iteracion_optimo)){
    #   iteracion_optimo <- iteracion
    #   tiempo_optimo <- as.numeric(difftime(time1 = Sys.time(),
    #                                        time2 = tiempo_inicial,
    #                                        units = "secs"))
    # }
  }

  i <- which.max(fitness)
  return(list("solucion" = poblacion[i,],
              "fitness" = fitness[i],
              "tiempo" = as.numeric(difftime(time1 = Sys.time(),
                                             time2 = tiempo_inicial,
                                             units = "secs")),
              # "optimo_cplex" = optimo_cplex,
              # "iteracion_optimo" = iteracion_optimo,
              # "tiempo_optimo" = tiempo_optimo,
              "best_solucion" = best_solucion,
              "best_fitness" = best_fitness,
              "iter_best" = iter_best,
              "tiempo_best" = tiempo_best
              )
        )
}
```

Fragmento de código A.6: Función programada en R que ejecuta el algoritmo genético.

Anexo B

Código R de algunas funciones auxiliares

```r
char2vec <- function(string){
  # Esta función convierte una cadena de caracteres de números separados
  # por espacios en un vector de números
  return(
    as.numeric(
      unlist(
        regmatches(x = string,
                   m = gregexpr(pattern = "[[:digit:]]+", text = string))
      )
    )
  )
}

transformar <- function(m, I){
  if(m == 0){
    return(0)
  } else {
    return(I - (m - 1))
  }
}

sigma <- function(k, B, B.dist){
  return(which(B[k] == B.dist))
}

filenames <- list.files("rppt_data", full.names = TRUE)
filenames <- (mixedsort(filenames))
list.of.files <- list()
for(i in c(1:length(filenames))){
  list.of.files <-
    append(x = list.of.files,
           values = readChar(filenames[i], file.info(filenames[i])$size))
}

extraer_datos_problema <- function(problema){
  problema <- str_replace_all(problema, c("[\n]" = " ",
                                          "[:]" = "",
                                          "[\\]]" = "",
                                          "[\\[]" = "" ))
  problema <- str_trim(string = problema, side = "both")

  problema <- gsub(pattern = "\\s+", replacement = " ", x = problema)
  problema <- paste(problema, "FIN", sep = " ")
  K.char <- str_match(string = problema,
                      pattern = "K \\s*(.*?)\\s* I")[1,2]
  I.char <- str_match(string = problema,
                      pattern = "I \\s*(.*?)\\s* PREFERENCES")[1,2]
```

```
  P.char <- str_match(string = problema,
                      pattern = "PREFERENCES \\s*(.*?)\\s* BUDGETS")[1,2]
  B.char <- str_match(string = problema,
                      pattern = "BUDGETS \\s*(.*?)\\s* FIN")[1,2]

  K <- char2vec(K.char)
  I <- char2vec(I.char)
  P <- char2vec(P.char)
  B <- char2vec(B.char)
  B.dist <- unique(sort(B))
  sigma <- unlist(lapply(X = c(1:length(B)), FUN = sigma,
                         B = B, B.dist = B.dist))

  # trasformación de preferencias
  P <- unlist(lapply(X = P, FUN = transformar, I = I))

  return(list("K" = K,
              "I" = I,
              "P" = matrix(data = P, nrow = K, ncol = I, byrow = TRUE),
              "B" = B,
              "B.dist" = B.dist,
              "sigma" = sigma))
}
```

Fragmento de código B.1: Funciones auxiliares programadas en R para la lectura de los datos de los problemas extraídos del banco de problemas.

```r
unlink(x = "archivos_dat", recursive = TRUE)
dir.create(path = "archivos_dat", recursive = TRUE)

for(n in c(26:40)){
  # nombre del archivo
  nombre_archivo <- paste0("archivos_dat/problema", as.character(n),".dat")

  # escritura del número de clientes |K|
  write(x = paste0("n = ", conjunto_problemas[[n]][["K"]], ";"),
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)

  # escritura del número de productos |I|
  write(x = paste0("m = ", conjunto_problemas[[n]][["I"]], ";"),
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)

  write(x = paste0("M = ", length(conjunto_problemas[[n]][["B.dist"]]), ";"),
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)

  # escritura del vector de presupuestos b
  write(x = "b = [",
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)
  write(x = conjunto_problemas[[n]][["B"]],
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE,
        sep = ",")
  write(x = "];",
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)

  # escritura del vector de presupuestos b.dist
  write(x = "bdis = [",
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)
  write(x = conjunto_problemas[[n]][["B.dist"]],
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE,
        sep = ",")
  write(x = "];",
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)

  # escritura del vector sigma
  write(x = "sigma = [",
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE)
  write(x = conjunto_problemas[[n]][["sigma"]],
        file = nombre_archivo,
        ncolumns = 1000,
        append = TRUE,
```

```
64          sep = ",")
65    write(x = "];",
66          file = nombre_archivo,
67          ncolumns = 1000,
68          append = TRUE)
69
70    # escritura de la matriz de preferencias s
71    write(x = "s = [",
72          file = nombre_archivo,
73          ncolumns = 1000,
74          append = TRUE)
75    for(i in c(1:nrow(conjunto_problemas[[n]][["P"]]))){
76      write(x = "[",
77            file = nombre_archivo,
78            ncolumns = 1000,
79            append = TRUE)
80      write(x = conjunto_problemas[[n]][["P"]][i,],
81            file = nombre_archivo,
82            ncolumns = 1000,
83            append = TRUE,
84            sep = ",")
85      write(x = "],",
86            file = nombre_archivo,
87            ncolumns = 1000,
88            append = TRUE)
89    }
90    write(x = "];",
91          file = nombre_archivo,
92          ncolumns = 1000,
93          append = TRUE)
94  }
```

Fragmento de código B.2: Fragmento de código auxiliar programado en R para la escritura de los ficheros *.dat* que requiere CPLEX.

```r
leer_solucion_cplex <- function(archivo, directorio, bateria_problemas){

  conexion <- paste0(directorio, archivo)

  n <-
    as.numeric(
      unlist(
        regmatches(x = archivo,
                   m = gregexpr(pattern = "[[:digit:]]+",
                                text = archivo))
      )
    )

  solucion_cplex <- readChar(con = conexion,
                             nchars =  file.info(conexion)$size)

  solucion_cplex <- str_replace_all(string = solucion_cplex,
                                    c("[\n]" = " ",
                                      "[\r]" = "",
                                      # "[:]" = "",
                                      "[\\]]" = "",
                                      "[\\[]" = "" ))

  solucion_cplex <- str_trim(string = solucion_cplex, side = "both")

  solucion_cplex <- paste(solucion_cplex, "FIN", sep = " ")

  solucion_cplex

  optimo <- str_match(string = solucion_cplex,
                      pattern = paste0("Valor de la funcion objetivo: ",
                                       "\\s*(.*?)\\s* Tiempo empleado:"))[1,2]
  optimo <- as.numeric(optimo)

  tiempo <- str_match(string = solucion_cplex,
                      pattern = paste0("Tiempo empleado: ",
                                       "\\s*(.*?)\\s* Solucion v:"))[1,2]
  tiempo <- as.numeric(tiempo)

  sol.v <- str_match(string = solucion_cplex,
                     pattern = "Solucion v: \\s*(.*?)\\s* Solucion z:")[1,2]
  sol.v <- gsub(pattern = "\\s+", replacement = " ", x = sol.v)
  sol.v <- char2vec(string = sol.v)
  sol.v <- matrix(data = sol.v,
                  nrow = length(conjunto_problemas[[n]][["B.dist"]]),
                  ncol = conjunto_problemas[[n]][["I"]], byrow = TRUE)

  indice_precios <- c()
  precios <- c()
  for(i in c(1:conjunto_problemas[[n]][["I"]])){
    if(any(sol.v[,i] == 1)){
      indice_precios <- append(indice_precios, which(sol.v[,i] == 1))
      precios <-
        append(precios,
               conjunto_problemas[[n]][["B.dist"]][which(sol.v[,i] == 1)])
    } else {
      indice_precios <- append(indice_precios, NA)
      precios <- append(precios, NA)
    }
  }

  return(list("problema" = n,
              "optimo" = optimo,
```

```
64                 "tiempo" = tiempo,
65                 "indices" = paste(indice_precios, collapse=" "),
66                 "precios" = paste(precios, collapse=" ")
67             )
68   )
69 }
```

Fragmento de código B.3: Funciones auxiliares programadas en R para la lectura de la solución proporcionada por CPLEX para los problemas.

Anexo C

Código CPLEX para la resolución de los problemas

```
execute PARAMS {
  // Tiempo limite en segundos
  cplex.tilim = 600;
  // Numero de procesadores disponebles (0 es la opcion por defecto y significa
  // que usa los que quiera)
  cplex.threads = 0;
}

// n = numero de clientes (|K|)
int n = ...;
range clientes = 1..n;
// m = numero de productos (|I|)
int m = ...;
range productos = 1..m;
// M = numero de presupuestos distintos
int M = ...;
range ind_pres = 1..M;

// vector de presupuestos de los clientes, el valor almacenado en la
// posicion k es el presupuesto del cliente k
float b[clientes] = ...;

// vector de presupuestos distintos ordenados de menor a mayor
float bdis[ind_pres] = ...;

// vector de indices, si en la posicion k se encuentra el elemento i significa
// que el cliente k tiene el i-esimo presupuesto del vector b_dist
int sigma[clientes] = ...;

// matriz de preferencias, los elementos de la matriz de preferencias pueden
// tomar valores enteros entre 0 y m
int s[clientes][productos] = ...;

dvar boolean v[ind_pres][productos];
dvar float+ z[clientes][productos];
dvar boolean x[clientes][productos];
dvar float+ w[clientes][productos];

maximize sum(k in clientes, j in productos)(z[k][j]);

subject to {

  forall(i in productos){
      sum(l in ind_pres)(v[l][i]) <= 1;
```

```
45  }
46
47  forall(k in clientes){
48    sum(i in productos)(x[k][i]) <= 1;
49  }
50
51  forall(k in clientes, i in productos){
52    z[k][i] <= sum(l in 1..sigma[k])(bdis[l] * v[l][i]);
53  }
54
55  forall(k in clientes, i in productos){
56    z[k][i] <= b[k] * x[k][i];
57  }
58
59  forall(k in clientes, i in productos){
60    x[k][i] <= sum(l in 1..sigma[k])(v[l][i]);
61    x[k][i] <= s[k][i];
62    x[k][i] <= z[k][i];
63  }
64
65  forall(k in clientes, i in productos){
66    sum(j in productos)(s[k][j] * x[k][j]) >= s[k][i] *
67    sum(l in 1..sigma[k])(v[l][i]);
68  }
69
70  forall(k in clientes, i in productos){
71    sum(j in productos)(z[k][j]) - sum(j in productos)(s[k][j] * w[k][j]) +
72    s[k][i] * sum(j in productos)(w[k][j]) <=
73    sum(l in 1..sigma[k])(bdis[l] * v[l][i]) + m*2*b[k] *
74    (1 - sum(l in 1..sigma[k])(v[l][i]));
75  }
76
77  forall(k in clientes){
78    sum(j in productos)(s[k][j] * w[k][j]) - sum(j in productos)(z[k][j]) >= 0;
79  }
80
81  forall(k in clientes, i in productos){
82    w[k][i] <= 2*b[k] * sum(l in 1..sigma[k])(v[l][i]);
83    w[k][i] <= 2*b[k] * s[k][i];
84  }
85
86  }
```

Fragmento de código C.1: Modelo programado en CPLEX para la resolución exacta de un problema de determinación de precios con empates utilizando la formulación (2.39)

```
main {
  for (var i=26; i<=40; i++){
    // Se genera el problema
    var source = new IloOplModelSource("formulacion1.mod");
    var def = new IloOplModelDefinition(source);
    var opl = new IloOplModel(def, cplex);
    var data = new IloOplDataSource("Problemas//problema"+i+".dat");
    opl.addDataSource(data);
    opl.generate();


    // Se resuelve el problema
    var tiempoAntes = new Date();
    cplex.solve();
    var tiempoDespues = new Date();

    // Se define el fichero .txt para la solucion
    var fichero = new IloOplOutputFile("soluciones_grandes//solucion_prueba"+i+"
    .txt");

    // Se escribe la solucion en el fichero
    fichero.writeln("Valor de la funcion objetivo:\n", cplex.getObjValue());
    fichero.writeln("Valor de gap:\n", cplex.getMIPRelativeGap());
    fichero.writeln("Tiempo empleado:\n", (tiempoDespues.getTime()-tiempoAntes.
    getTime())*1e-3);
    fichero.writeln("Solucion v:\n", opl.v);
    fichero.writeln("Solucion z:\n", opl.z);
    fichero.writeln("Solucion x:\n", opl.x);
    fichero.writeln("Solucion w:\n", opl.w);

    // Se cierra el fichero
    fichero.close();
  };
}
```

Fragmento de código C.2: Fragmento de código utilizado para resolver a la vez todos los problemas de la batería de problemas

Anexo D

Código R para la ejecución del algoritmo genético

```
tamaño_poblacion <- c(100,200)
numero_de_iteraciones <- c(100,200)
prob_mut <- c(0.05,0.25)
numero_de_ejecuciones <- 10

datos_genetico <-
  data.frame("Problema" = numeric(0),
             "TamañoPoblacion" = numeric(0),
             "NIter" = numeric(0),
             "Pm" = numeric(0),
             "Ejecucion" = numeric(0),
             "BestFit" = numeric(0),
             "TBest" = numeric(0),
             "IterBest" = numeric(0),
             "Tiempo" = numeric(0),
             "P1" = numeric(0),
             "P2" = numeric(0),
             "P3" = numeric(0),
             "P4" = numeric(0),
             "P5" = numeric(0),
             "OptimoCPLEX" = numeric(0),
             "IteracionCPLEX" = numeric(0),
             "Tiempo_to_CPLEX" = numeric(0))

for(problema in c(26:70)){
  print(paste("Resolviendo problema", problema, sep = " "))

  for(tamaño in tamaño_poblacion){
    print(paste("Con tamaño de poblacion", tamaño, sep = " "))

    for(niter in numero_de_iteraciones){
      print(paste("Con numero de iteraciones", niter, sep = " "))

      for(prob in prob_mut){
        print(paste("Con pm", prob, sep = " "))

        for(j in c(1:numero_de_ejecuciones)){
          print(paste("Ejecución", j, sep = " "))
          solucion_genetico <- genetico(conj_problemas[[problema]],
                                        tam.pobla = tamaño,
                                        num.padres = tamaño/2,
                                        pm = prob,
                                        num.iter = niter)
          datos_genetico_tuned <-
```

```
            datos_genetico_tuned %>%
            add_row("Problema" = problema,
                    "TamañoPoblacion" = tamaño,
                    "NIter" = niter,
                    "Pm" = prob,
                    "Ejecucion" = j,
                    "BestFit" = solucion_genetico[["best_fitness"]],
                    "TBest" = solucion_genetico[["tiempo_best"]],
                    "IterBest" = solucion_genetico[["iter_best"]],
                    "Tiempo" = solucion_genetico[["tiempo"]],
                    "P1" = solucion_genetico[["best_solucion"]][1],
                    "P2" = solucion_genetico[["best_solucion"]][2],
                    "P3" = solucion_genetico[["best_solucion"]][3],
                    "P4" = solucion_genetico[["best_solucion"]][4],
                    "P5" = solucion_genetico[["best_solucion"]][5],
                    "OptimoCPLEX" = solucion_genetico[["optimo_cplex"]],
                    "IteracionCPLEX" = solucion_genetico[["iteracion_optimo"]],
                    "Tiempo_to_CPLEX" = solucion_genetico[["tiempo_optimo"]])

        }


      }


    }


  }
}
```

Fragmento de código D.1: Fragmento de código auxiliar programado en R para la ejecución del algoritmo genético sobre la batería de problemas.