Universidad de Sevilla

Facultad de Matemáticas

Estudio de las características y técnicas de resolución de los problemas de asignación multiperiodo

Trabajo Fin de Máster

Máster Universitario en Matemáticas

Autor:
Álvaro Blázquez Molino

Resumen

En este Trabajo Fin de Máster trabajaremos con problemas de optimización combinatoria con variables enteras que pueden ser difíciles de resolver con exactitud debido a que tienen un gran número de variables o un conjunto de restricciones complicadas. Para hacer frente a estos problemas, se aplicará un método de descomposición conocido como "Relajación Lagrangiana".

El objetivo de este trabajo involucra diferentes aspectos de la investigación en optimización combinatoria. En primer lugar, se definirán los principales conceptos asociados a los problemas de programación matemática y algunos resultados. En segundo lugar, también se definirá el concepto de Relajación Lagrangiana de un problema y cómo nos puede ayudar a acotar o encontrar la solución del problema original. Finalmente, se estudiarán los Problemas de Asignación Multiperíodo, y se desarrollarán y detallarán algunos métodos y procedimientos de solución para mostrar a través de casos prácticos de estos problemas cómo los conocidos métodos iterativos que utilizan la Relajación Lagrangiana pueden ayudarnos a aproximarnos a la solución del problema original en un límite de tiempo.

Abstract

In this Master Thesis we will work with combinatorial optimization problems with integer variables that can be difficult to solve exactly due to having a large number of variables or a set of complicating restrictions. To deal with these problems, a decomposition method known as "Lagrangian Relaxation" will be applied.

The aim of this work involves different aspects of research in combinatorial optimization. Firstly, the main concepts associated with mathematical programming problems and some results will be recalled. Secondly, the Lagrangian Relaxation concept of a problem will also be defined and how it can help us to bound or find the solution of the original problem. Finally, Multiperiod Assignment Problems will be studied, and some solution methods and procedures will be developed and detailed to show through practical cases of these problems how the well-known iterative methods that use Lagrangian Relaxation can help us to approximate the solution of the original problem in a reasonable time limit.

Índice general

Prefacio

En este Trabajo Fin de Máster se trabajará con problemas de optimización combinatoria con variables enteras que pueden ser complicados de resolver de forma exacta debido a que posean un número alto de variables o un conjunto de restricciones complicantes. Para tratar estos problemas, se aplicará un método de descomposición conocido como Relajación Lagrangiana.

Entre los objetivos de este trabajo estarán, en primer lugar, revisar los principales conceptos asociados a los problemas de programación matemática y algunos resultados principales. También se definirá el concepto Relajación Lagrangiana de un problema y cómo nos puede ayudar a aproximarnos o encontrar la solución del problema original. Por último, se estudiarán un tipo de problemas llamados Problemas de Asignación Multiperiodo, y se tratará de mostrar mediante casos prácticos de estos problemas cómo los métodos iterativos conocidos que se ayudan de la Relajación Lagrangiana nos pueden ayudar a aproximar la solución del problema original en una cantidad razonable de tiempo.

En el Capítulo 1 se revisarán los conceptos necesarios para tratar los problemas que abodaremos en la práctica, como los conceptos de formulación, cota primal y relajación.

En el Capítulo 2 se profundizará en la Relajación Lagrangiana, viendo resultados importantes del Dual Lagrangiano y el algoritmo de subgradiente y los parámetros que se ajustan en él para lograr la mejor convergencia a la solución.

En el Capítulo 3 se definirán los Problemas de Asignación Multiperiodo de manera general, y se verán más en detalle los dos problemas que abordaremos, el Planar y el Axial de 3 índices, mostrando su Dual Lagrangiano y su descomposición en subproblemas, entre otras cosas.

En el Capítulo 4 se desarrollará la implementación computacional de los problemas tratados en el Capítulo 3, y se hará una comparativa entre la calidad de las soluciones y los tiempos de

resolución requeridos por el solver comercial Gurobi (al tratar de resolver el problema Axial) y los obtenidos con el algoritmo de subgradiente.

Capítulo 1

Preliminares

En este capítulo se introducirán los conceptos necesarios para definir los problemas de programación matemática que se estudiarán, así como la definición de una relajación, algo fundamental para formular la Relajación Lagrangiana. En este capítulo la mayor fuente de información será procedente de [17].

1.1. Problemas y formulaciones

Consideremos el problema de programación lineal

$$\text{(LP)} \quad \max \quad \{cx : Ax \leq b, x \geq 0\},$$

donde A es una matriz $m \times n$, c es un vector fila n-dimensional, b un vector columna m-dimensional, y x un vector columna n-dimensional de variables. Si todas las variables son enteras, tenemos el problema de programación lineal entera (IP):

$$\text{(IP)} \begin{cases} \max & cx \\ \text{s.a} & Ax \leq b \\ & x \geq 0 \quad \text{y entero} \end{cases}$$

De manera análoga se pueden definir el problema de programación lineal entera mixta (MIP) en caso de que solo algunas variables sean enteras, y el problema de programación entera binaria 0-1 (BIP), en el que las variables son binarias y por tanto pueden tomar el valor 0 o el valor 1.

Sea un conjunto finto $N = 1, ..., n$, pesos c_j para cada $j \in N$, y un conjunto F de subconjuntos factibles de N. El problema de encontrar un subconjunto de mínimo peso factible puede ser expresado como el problema de optimización combinatoria:

$$(COP) \quad \min_{S \subseteq N} \{\sum_{j \in S} c_j : S \in F\}.$$

A continuación se revisarán algunos conceptos relacionados con los poliedros y sus atributos con el fin de comprender las formulaciones que se realizarán al describir la Relajación Lagrangiana.

Definición 1. *Un **poliedro** es un subconjunto de $\mathbb{R}^n$ descrito por una cantidad finita de restricciones lineales $P = \{x \in \mathbb{R}^n : Ax \leq b\}$.*

Definición 2. *Un poliedro $P \subseteq \mathbb{R}^{n+p}$ es una **formulación** para un conjunto $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ si y solo si $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.*

Definición 3. *Dado un conjunto $X \subseteq \mathbb{R}^n$, la **envolvente convexa de** X, denotada $conv(X)$, es definida como sigue: $conv(X) = \{x : x = \sum_{i=1}^{t} \lambda_i x^i, \sum_{i=1}^{t} \lambda_i = 1, \lambda_i \geq 0$ con $i = 1, ..., n$ sobre todos los subconjuntos finitos $\{x^1, ..., x^t\}$ de $X\}$.*

Proposición 1. *Si $X = \{x \in \mathbb{Z}^n : Ax \leq b\}$, entonces $conv(X)$ es un poliedro.*

Definición 4. *Dado un poliedro P, x es un **punto extremo** de P si $x^1, x^2 \in P$ con $x = \lambda x^1 + (1 - \lambda)x^2$, $0 < \lambda < 1$ implica que $x = x^1 = x^2$.*

Proposición 2. *Consideremos $max\{cx : x \in PX\}$, donde P es un poliedro. Si existe valor óptimo finito, entonces existe un punto extremo de P que es óptimo.*

Gracias a estos resultados podemos reemplazar el problema (IP): $\{maxcx : x \in X\}$ por el problema lineal equivalente $\{maxcx : x \in conv(X)\}$. Esta reducción a un problema lineal se

mantiene para conjuntos no acotados enteros $X = \{x : Ax \leq b \quad \text{y entero}\}$ y para conjuntos enteros mixtos $X = \{(x, y) : Ax + Gy \leq b, x \geq 0 \quad \text{y entero}, \quad y \geq 0\}$ con A, G, b racionales. En cualquier caso se trata de una solución teórica, ya que en la mayor parte de los casos no hay una caracterización sencilla para todas las desigualdades que describen $conv(X)$.

Veamos ahora en qué caso podemos decir que una formulación P_1 es mejor que otra formulación P_2. Como la solución ideal $conv(X)$ satisface $X \subseteq conv(X) \subseteq P$ para cualquier formulación P, esto sugiere que la mejor formulación será aquella que más se ajuste a $conv(X)$ lo que formalmente se expresaría así:

Definición 5. *Dado un conjunto $X \subseteq \mathbb{Z}^n$, y dos formulaciones P_1 y P_2 para X, P_1 es* ***mejor formulación*** *que P_2 si $P_1 \subseteq P_2$.*

Ejemplo 1. *Consideremos el conjunto $S = \{(0,0), (0,1), (1,0), (1,1)\}$. Dos formulaciones válidas para este conjunto son:*

$$\begin{aligned} P1 = \{x \in \mathbb{R}^2 : & -x_1 \leq 0 \\ & -x_2 \leq 0 \\ & x_1 \leq 1 \\ & x_2 \leq 1\} \end{aligned}$$

$$\begin{aligned} P2 = \{x \in \mathbb{R}^2 : & -x_1 \leq 0 \\ & -x_2 \leq 0 \\ & x_1 \leq 1,5 \\ & x_2 \leq 1,5\} \end{aligned}$$

Como $P_1 \subseteq P_2$, entonces P_1 es mejor formulación que P_2 para el conjunto S, véase la Figura 1.1 y la Figura 1.2.

Para más información sobre estos problemas consultar [17] o [10].

Figura 1.1: Formulación P_1 para el conjunto S

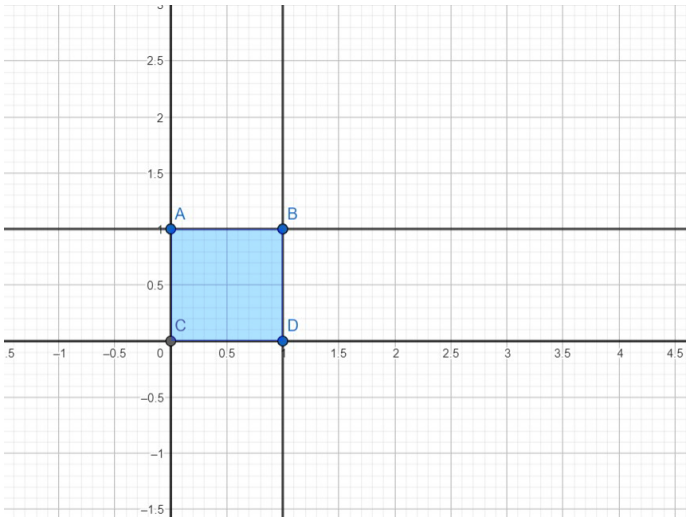
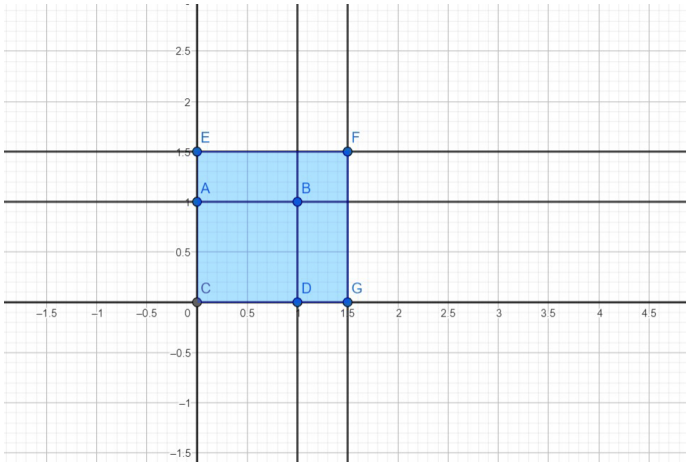


Figura 1.2: Formulación P_2 para el conjunto S



1.2. Cotas

Consideramos el problema (COP)

$$(\text{COP}) \quad Z = \max \quad \{c(x) : x \in X \subseteq \mathbb{Z}^n\}.$$

Nos preguntamos cómo se podría probar que un punto x^* es óptimo. Trataremos de encontrar una cota inferior $\underline{Z} \leq Z$ y una cota superior $\bar{Z} \leq Z$ de forma que $\underline{Z} = \bar{Z} = Z$. En la práctica,

necesitaremos emplear un algoritmo que encuentre una sucesión decreciente de cotas superiores

$$\bar{Z}_1 > \bar{Z}_2 > ... > Z_U \geq Z$$

y una sucesión creciente de cotas inferiores

$$\underline{Z}_1 < \underline{Z}_2 < ... < Z_L \leq Z$$

y detenerse cuando

$$Z_U - Z_L \leq \epsilon,$$

donde ϵ es un pequeño valor positivo que nosotros elegiremos.

Definición 6. *Sean Z_U y Z_L una cota superior y una cota inferior del problema (COP), respectivamente. Se define el* ***Gap Relativo*** *(como se define en [14]) entre dichas cotas como:*

$$gap_{UL} = \frac{Z_U - Z_L}{Z_U} \times 100\,\%$$

Cada solución factible $x^* \in X$ otorga una **cota inferior o primal** $\underline{Z} = c(x^*) \leq Z$. Por otra parte, para encontrar cotas superiores para un problema de maximización (o cotas inferiores para un problema de minimización), que llamaremos **cotas duales**, se reemplaza el problema original por un problema de optimización más simple cuyo valor óptimo sea mayor o igual que el del problema original Z. Para que este problema, al que denotaremos **problema relajado**, cumpla esta propiedad, hay dos opciones:

1) Aumentar el conjunto de soluciones factibles.

2) Reemplazar la función objetivo por una función cuyo valor sea igual o superior en cada punto factible.

Definición 7. *Un problema (RP) $Z^{RP} = max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ es una* ***relajación*** *del problema (IP) $Z = max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ si:*

1) $X \subseteq T$, y

2) $f(x) \geq c(x)\ \forall x \in X$.

Proposición 3. *Si (RP) es una relajación de (IP), entonces* $Z^{RP} \geq Z$.

Demostración. Si x^* es solución óptima de (IP), $x^* \in X \subseteq T$ y $Z = c(x^*) \leq f(x^*)$. Como $x^* \in T$, $f(x^*)$ es una cota inferior de Z^{RP} y $Z \leq f(x^*) \leq Z^{RP}$. □

1.3. Relajaciones

Empezamos esta sección recordando los conceptos definidos anteriormente de **cota primal**, que es una cota inferior proporcionada por una solución factible del problema de maximización, y el concepto de **cota dual**, que es una cota superior del problema de maximización. En el caso de tener un problema de minimización, las cotas primales serán lógicamnte cotas superiores, y las cotas duales serán cotas inferiores.

Definición 8. *Dado un problema de programación entera* $Z = max\{cx : x \in P \cap \mathbb{Z}^n\}$ *con formulación* $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$, *la* ***relajación de la programación lineal*** *es* $Z^{LP} = max\{cx : x \in P\}$.

Proposición 4. *Sean* P_1, P_2 *formulaciones para el problema de programación entera* $max\{cx : x \in X \subseteq \mathbb{Z}^n\}$ *de forma que* P_1 *es mejor formulación que* P_2, *es decir,* $P_1 \subset P_2$. *Si* $Z_i^{LP} = max\{cx : x \in P_i\}$ *para* $i = 1, 2$ *son los valores asociados a cada relajación lineal, entonces* $Z_1^{LP} \leq Z_2^{LP}$ *para cualquier* c.

Las relajaciones no solo proporcionan cotas duales, sino que a veces también nos permiten probar optimalidad.

Proposición 5. *Consideremos un problema (IP) y una relajación de este problema (RP):*

1) Si una relajación (RP) es infactible, el problema original (IP) es infactible.

2) Sea x^* solución óptima de (RP). Si $x^* \in X$ y $f(x^*) = c(x^*)$, entonces x^* es solución óptima de (IP).

Demostración.

1) Al ser (RP) infactible, $T = \emptyset$ y por tanto $X = \emptyset$.

2) Como $x^* \in X$, $Z \geq c(x^*) = f((x^*) = Z^{RP})$. Como $Z \leq Z^{RP}$, obtenemos $c(x^*) = Z = Z^{RP}$.

□

1.4. Cotas Primales

Consideramos algunas formas simples de obtener soluciones factibles o cotas primales.

Definición 9. *Un problema (RE) $Z^{RE} = max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ es una restricción del problema entero $Z = max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ si:*

- $T \subseteq X$, *y*
- $f(x) \leq c(x) \quad \forall x \in X$.

Proposición 6. *Si (RE) es una restricción de (IP) con solución óptima x^{RE}, x^{RE} es factible en (IP) y $c(x^{RE}) \leq Z$.*

Algunas formas de construir restricciones son fijar valores de algunas variables, reemplazar desigualdades por igualdades o añadir restricciones adicionales. Esto se suele hacer de forma que la restricción resultante sea un problema más fácil que se pueda resolver rápidamente.

1.5. Algoritmo del simplex y de punto interior

Consideremos el problema:

$$\begin{aligned} \max \quad & cx \\ \text{s.a} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

El método simplex es un algoritmo iterativo aplicado a problemas de programación lineal. El método comienza añadiendo nuevas variables, llamadas variables de holgura, de forma que al añadirlas a las restricciones, estás pasan a ser de igualdad. El algoritmo divide en las iteraciones las variables en básicas y no básicas, y va mejorando la solución en el sentido de hacer su valor objetivo mayor hasta que no se puede continuar. Por otro lado, los algoritmos de punto interior buscan el punto óptimo por el interior del dominio, mientras que el simplex busca por los vértices. Además, el algoritmo de punto interior encuentra el óptimo en tiempo polinomial, aunque en la práctica el algoritmo del simplex converge más rápido. Para más información de estos algoritmos, ver [15].

1.6. Algoritmos de ramificación y poda

Consideramos el problema:

$$Z = max\{cx : x \in S\}$$

Nos preguntamos cómo dividir el problema en una serie de subproblemas que sean más fáciles de resolver.

Proposición 7. *Sea $S = S_1 \cup \cdots \cup S_k$ una descomposición de S en conjuntos más pequeños, y sea $Z^k = max\{cx : x \in S_k\}$ para $k = 1, \cdots, K$. Entonces, $Z = max_k Z^k$.*

Proposición 8. *Sea $S = S_1 \cup \cdots \cup S_k$ una descomposición de S en conjuntos más pequeños, y sea $Z^k = max\{cx : x \in S_k\}$ para $k = 1, \cdots, K$, $\bar{Z}^k$ una cota superior de Z^k y $\underline{Z}_k$ una cota inferior de Z^k. Entonces, $\bar{Z} = max_k \bar{Z}^k$ es una cota superior de Z y $\underline{Z} = max_k \underline{Z}^k$ es una cota superior de Z.*

Ahora veremos cómo realizar un algoritmo de ramificación y poda basado en la relajación lineal. Se inicia el algoritmo con el conjunto S con formulación P en una lista L. La cota inferior inicial es $\underline{Z} = -\infty$ y x^* inicialmente será vacío. Después se usa un algoritmo heurístico para encontrar una solución factible x^H con valor objetivo Z^H, y se actualiza $\underline{Z} = Z^H$ y $x^* = x^H$.

A continuación se realizará un paso de comprobación que llamaremos inicio del bucle, y al que volveremos de forma recurrente cada vez que se realice una poda. En este paso se comprueba si la lista L es vacía. Si lo es el algoritmo se detendrá. En caso contrario, se elimina S^i con formulación P^i de L para un cierto i. Después se resuelve la relajación lineal de la formulación P^i, con solución $x^i(LP)$ y valor objetivo $\bar{Z}^i$. en caso de P^i sea vacío se poda o descarta la ramificación por S^i por infactibilidad, y si $\bar{Z}^i \leq \underline{Z}$ se poda por acotación. Si no se poda, se

comprueba si $x^i(LP)$ es entera o no. Si lo es, se actualiza la cota primal $\underline{Z} = \bar{Z}^i$, se hace $x^* = x^i(LP)$ y se poda por optimalidad. Recordemos que cada vez que se realiza una poda se vuelve al inicio del bucle. Si por el contrario $x^i(LP)$ no es de coordenadas enteras, se devuelven dos subproblemas S_1^i y S_2^i a la lista L con formulaciones P_1^i y P_2^i y cotas superiores $\bar{Z}^i$, y se vuelve al inicio del bucle.

A partir de la primera vez que se vuelve al bucle en la que contemos con una cota superior $\bar{Z}^i$, antes de resolver la relajación lineal, se comprobará si $\bar{Z}^i \leq \underline{Z}$, y en tal caso se podará por acotación.

1.7. Algoritmos de planos de corte

Consideramos el problema:

$$(IP) \quad max\{cx : x \in X\}$$

donde $X = \{Ax \leq b, x \in \mathbb{Z}_+^n\}$.

Definición 10. *Una desigualdad $\pi x \leq \pi_0$ es una **desigualdad válida** para $X \subseteq \mathbb{R}^n$ si $\pi x \leq \pi_0 \forall x \in X$.*

Supongamos que $X = P \cap \mathbb{Z}^n$ y que conocemos una familia $\mathcal{F}$ de desigualdades válidas $\pi x \leq \pi_0$, $(\pi, \pi_0) \in \mathcal{P}$ para X. En muchos casos, $\mathcal{F}$ contiene demasiadas desigualdades (2^n o más) como para ser agregadas a priori. Además, dado una función objetivo específica, el objetivo no es encontrar la envolvente convexa completa, sino encontrar una buena aproximación de conv(X). A continuación se describe un algoritmo básico de planos de corte para (IP), $max\{cx : x \in X\}$, que genera desigualdades útiles a partir de $\mathcal{F}$.

El algoritmo de plano de corte se inicia estableciendo $t = 0$ y $P^0 = P$. En cada iteración t, se resuelve el problema lineal $Z^t = max\{cx : x \in P^t\}$. Sea x^t la solución óptima. Si x^t es de coordenadas enteras, se detiene el algoritmo y x^t es óptimo de (IP). En caso contrario, se resuelve el problema de separación de x^t y $\mathcal{F}$. Si se encuentra una desigualdad $(\pi^t, \pi_0^t) \in \mathcal{F}$ con $\pi^t x^t > \pi_0^t$, se establece $P^{t+1} = P^t \cap \{x : \pi^t x \leq \pi_0^t\}$ y se aumenta t. En caso contrario se detiene.

Si el algoritmo termina sin encontrar una solución entera para IP,

$$P^t = P \cap \{x : \pi^i x \leq \pi_0^i, \quad i = 1, \cdots, t\}$$

es una formulación mejorada que se puede dar como input a un solver que utilice un algoritmo de ramificación y corte o branch-and-cut. En la práctica suele ser mejor añadir varios cortes no

verificados en una sola iteración que añadir un solo corte.

Capítulo 2

Dualidad Lagrangiana

En este segundo capítulo profundizaremos en la Relajación Lagrangiana citando el contenido de [17]. En concreto, veremos que esta clase de relajaciones busca eliminar restricciones que complican el problema pasándolas a un término de penalización en la función objetivo. También se describirá el algoritmo de subgradiente, que en cada iteración nos proporcionará cotas que busquen la convergencia al valor óptimo del problema original, y se analizará cómo seleccionar los parámetros del algoritmo para asegurar la convergencia.

2.1. Relajación Lagrangiana

Consideremos el problema IP:

$$(\text{IP}) \begin{cases} Z = \max & cx \\ \text{s.a} & Ax \leq b \\ & Dx \leq d \\ & x \in \mathbb{Z}_+^n \end{cases}$$

Supongamos que las restricciones $Dx \leq d$ son *buenas* en el sentido de que si el problema solo tuviera esas restricciones sería fácil de resolver. Entonces, si eliminamos las restricciones complicantes o complicadas $Ax \leq b$, la relajación resultante es más fácil de resolver que el problema original (IP). Sin embargo, la cota resultante puede ser débil porque algunas restricciones retiradas sean importantes y estén siendo ignoradas. Una manera de abordar esta dificultad será utilizar la Relajación Lagrangiana.

Consideremos el problema IP de una forma algo más general:

$$(\text{IP}) \begin{cases} Z = \max & cx \\ \text{s.a} & Ax \leq b \\ & x \in X \end{cases}$$

donde $Ax \leq b$ son m restricciones complicadas. Para cualquier valor de $u = (u_1, ..., u_m) \geq 0$, definimos el problema:

$$(\text{IP(u)}) \begin{cases} Z(u) = \max & cx + u(b - Ax) \\ & x \in X \end{cases}$$

Proposición 9. *El problema (IP(u)) es una relajación de (IP)* $\forall u \geq 0$.

Demostración. Recordemos que (IP(u)) es una relajación de (IP) $\forall u \geq 0$ si:

- La región factible contiene a la región factible original. Esto se cumple pues $S = \{x : Ax \leq b, x \in X\} \subseteq X$.
- El valor objetivo es como mínimo igual de grande en (IP(u)) que en (IP) para cada una de las soluciones factibles de (IP). Si $x \in S$ y $u \geq 0$, entonces $Ax \leq b, \quad u(b - Ax) \geq 0$ y entonces $cx + u(b - Ax) \geq cx \quad \forall x \in S$.

□

En el problema (IP(u)) las restricciones complicadas se tratan añadiéndolas a la función objetivo con un término de penalización $u(b - Ax)$. Se llama a u **multiplicador de Lagrange** asociado a las restricciones $Ax \leq b$.

El problema (IP(u)) se llama **Relajación Lagrangiana** de (IP) con parámetro u. Al ser (IP(u)) una relajación de (IP), $Z(u) \geq Z$ y tenemos así una cota superior del valor óptimo de (IP). Para encontrar la mejor cota superior, es decir, la más baja, de entre todos los posibles valores de u, hay que resolver el Problema Dual Lagrangiano:

$$(\text{LD}) \quad W_{LD} = \text{mín}\{Z(u) : u \geq 0\}$$

Veamos un ejemplo numérico:

Ejemplo 2.

$$
\begin{aligned}
max \quad & 7x_1 + 2x_2 \\
s.t. \quad & -x_1 + 2x_2 \leq 4 \\
& 5x_1 + x_2 \leq 20 \\
& -2x_1 - 2x_2 \leq -7 \\
& -2x_1 \leq -2 \\
& x_2 \leq 4 \\
& x \in \mathbb{Z}^2_+
\end{aligned}
$$

Al relajar la restricción $-x_1 + 2x_2 \leq 4$, la relajación lagrangiana resultante es:

$$
\begin{aligned}
Z_{LR}(u) = max \quad & 7x_1 + 2x_2 + u(4 + x_1 - 2x_2) \\
\text{s.t.} \quad & 5x_1 + x_2 \leq 20 \\
& -2x_1 - 2x_2 \leq -7 \\
& -2x_1 \leq -2 \\
& x_2 \leq 4 \\
& x \in \mathbb{Z}^2_+
\end{aligned}
$$

Cuando las m restricciones que son dualizadas son igualdades de la forma $Ax = b$, los correspondientes multiplicadores de Lagrange $u \in \mathbb{R}^m$ no tienen restricción de signo, y el Dual Lagrangiano es

$$W_{LD} = \min_u Z(u)$$

Proposición 10. *Si* $u \geq 0$

1) $x(u)$ *es una solución óptima de (IP(u)), y*

2) $Ax(u) \leq b$, *y*

3) $(Ax(u))_i = b_i$ *para las* i *que cumplen* $u_i > 0$,

entonces $x(u)$ *es óptimo en (IP).*

Demostración.

Por 1), $W_{LD} \leq Z(u) = cx(u) + u(b - Ax(u))$.

Por 3), $cx(u) + u(b - Ax(u)) = cx(u)$.

Por 2), $x(u)$ es factible en IP y entonces $cx(u) \leq Z$.

Entonces, $W_{LD} \leq cx(u) + u(b - Ax(u)) = cx(u) \leq Z$. Pero como $W_{LD} \geq Z$, entonces $W_{LD} = Z$ y $x(u)$ es óptimo en (IP). □

Observemos que si las restricciones dualizadas son igualdades, la condición 3) se satisface automáticamente. Por lo tanto una solución óptima de (IP(u)) es óptima para (IP) si es factible en (IP).

2.2. La fuerza del Dual Lagrangiano

En esta sección veremos una caracterización del Dual Lagrangiano mediante el siguiente teorema:

Teorema 1. $W_{LD} = max\{cx : Ax \leq b, x \in conv(X)\}$

Demostración.

Supongamos que el conjunto X tiene un número grande pero finito de puntos $\{x_1, ..., x_T\}$.

Ahora consideramos:

$$
\begin{aligned}
W_{LD} &= \min_{u \geq 0} Z(u) && (2.1)\\
&= \min_{u \geq 0} \{max_{x \in X}[cx + u(b - Ax)]\} && (2.2)\\
&= \min_{u \geq 0} \{max_{t=1,...,T}[cx^t + u(b - Ax^t)]\} && (2.3)\\
&= \min \eta && (2.4)\\
&\quad \eta \geq cx^t + u(b - Ax^t) \quad \forall t && (2.5)\\
&\quad u \in \mathbb{R}^T_+, \eta \in \mathbb{R}, && (2.6)
\end{aligned}
$$

donde la nueva variable η representa una cota superior sobre $Z(u)$. Pasando al dual del problema que ha resultado obtenemos:

$$
\begin{aligned}
w_{LD} = \quad & \max \sum_{t=1}^{T} \mu_t (cx^t) \\
& \sum_{t=1}^{T} \mu_t (Ax^t - b) \leq 0 \\
& \sum_{t=1}^{T} \mu_t = 1 \\
& \mu \in \mathbb{R}^T_+
\end{aligned}
$$

Haciendo $x = \sum_{t=1}^{T} \mu_t x^t$, junto a $\sum_{t=1}^{T} \mu_t = 1$ y $\mu \in \mathbb{R}_+^T$, obtenemos:

$$\begin{aligned} w_{LD} = \quad & \text{máx}\, cx \\ & Ax \leq b \\ & x \in conv(X). \end{aligned}$$

Más generalmente se puede mostrar que el resultado se mantiene cuando $X = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ es la región factible de cualquier IP. □

Este teorema nos muestra la fuerza de una cota que obtenemos a partir de dualizar. En algunos casos, no es mejor cota que la que proporciona la relajación lineal.

Corolario: Si $X = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ y $conv(X) = \{x \in \mathbb{R}_+^n : Dx \leq d\}$, entonces $W_{LD} = \text{máx}\{cx : Ax \leq b, Dx \leq d, x \in \mathbb{R}_+^n\}$.

Definición 11. *Una función* $f : \mathbb{R}^n \to \mathbb{R}$ *es convexa si* $\forall x^1, x^2 \in \mathbb{R}^n \quad y$

$$0 \leq \lambda \leq 1, \quad f(\lambda x^1 + (1-\lambda)x^2) \leq \lambda f(x^1) + (1-\lambda) f(x^2).$$

Hemos visto anteriormente que:

$$W_{LD} = \text{mín}_{u \geq 0}\{\text{máx}_{t=1,\ldots,T}[cx^t + u(b - Ax^t)]\}.$$

Entonces, el Dual Lagrangiano puede verse como un problema de minimización de la función lineal a trozos y convexa, pero no diferencible $Z(u)$.

2.3. Resolución del Dual Lagrangiano

La formulación de programación lineal (2.4)-(2.6) que aparece en la prueba del Teorema 1 proporciona una forma de calcular W_{LD}, pero debido a la cantidad de restricciones presentes puede dificultar la tarea. Se describe a continuación describimos un algoritmo de subgradiente diseñado para resolver el problema de minimización de una función lineal por partes y convexa:

$\text{mín}_{u\geq 0}\, f(u)$, donde $f(u) = \text{máx}_{t=1,...,T}[a^t u - b_t]$

En el caso del Dual Lagrangiano, tenemos:

$W_{LD} = \text{mín}_{u\geq 0}\, Z(u)$

con $Z(u) = \text{máx}_{t=1,...,T}[(b - Ax^t)u + cx^t]$

Definición 12. *Un **subgradiente** en u de una función convexa $f : \mathbb{R}^m \to \mathbb{R}$ es un vector $\gamma(u) \in \mathbb{R}^m$ tal que $f(v) \geq f(u) + \gamma(u)^T(v - u) \quad \forall v \in \mathbb{R}^m$.*

Para una función convexa continuamente diferenciable f, $\gamma(u) = \nabla f(u) = (\frac{\delta f}{\delta u_1}, ..., \frac{\delta f}{\delta u_m})$ es el gradiente de f en u.

Algoritmo 1: Algoritmo de subgradiente para el Dual Lagrangiano

Inicialización: $u = u^0$

Iteración k : $u = u^k$.
Resuelve el Problema Lagrangiano IP(u^k) con solución óptima $x(u^k)$.
$u_i^{k+1} = max\{u_i^k - \mu_k(b - Ax(u^k)_i), 0\}\ i = 1, ..., m$
$k \leftarrow k + 1$

El vector $b - Ax(u^k)$ es un subgradiente de $Z(u)$ en u^k. En cada iteración, el algoritmo toma un paso del punto u^k en la dirección opuesta a un subgradiente. La dificultad reside en escoger las longitudes de paso $\{\mu_k\}_{k=1}^{\infty}$.

Teorema 2.

a) Si $\sum_k \mu_k \to \infty$ y $\mu_k \to 0$ cuando $k \to \infty$, entonces $Z(u_k) \to W_{LD}$ el valor óptimo de la relajación lagrangiana.

b) Si $\mu_k = \mu_0 \rho^k$ para algún $\rho < 1$, entonces $Z(u_k) \to W_{LD}$ si μ_0 y ρ son suficientemente grandes.

c) Si $\bar{W} \geq W_{LD}$ y $\mu_k = \eta_k[Z(u^k) - \bar{W}]/\|b - Ax(u^k)\|^2$ con $0 < \eta_k < 2$, entonces $Z(u^k) \to \bar{W}$, o bien el algoritmo encuentra u^k con $\bar{W} \geq Z(u^k) \geq W_{LD}$ para algún k finito.

Este teorema nos dice que a) garantiza la convergencia, pero como la serie $\{\mu_k\}$ debe ser divergente (por ejemplo, $\mu_k = 1/k$), la convergencia es demasiado lenta para ser de interés práctico.

Por otro lado, las longitudes de paso de b) o c) conducen a una convergencia mucho más rápida, pero cada vez con un posible inconveniente.

Usando b), los valores iniciales de μ_0 y ρ deben ser lo suficientemente grandes, o de lo contrario la serie geométrica $\mu_0\rho^k$ tiende a 0 demasiado rápido y la sucesión u^k converge antes de llegar a un punto óptimo. En la práctica, en lugar de disminuir μ_k en cada iteración, se logra una disminución geométrica al reducir a la mitad el valor de μ_k cada ν iteraciones, donde ν es algún parámetro natural del problema, como lo es el número de variables.

Usando c), la dificultad es que una cota superior dual $\bar{W} \geq W_{LD}$ normalmente es desconocida. En la práctica, es más probable que conocer una buena cota inferior primal primal $\underline{W} \leq W_{LD}$. Esta cota inferior $\underline{W}$ se usa inicialmente en lugar de $\bar{W}$. Sin embargo, si $\underline{W} < W_{LD}$, el término $Z(u^k) - \underline{W}$ en el numerador de la expresión para μ_k no tiende a cero, por lo que las sucesiones $\{u^k\}$, $\{Z(u^k)\}$ no convergerán. Si se observa esto el valor de $\underline{W}$ debe aumentarse.

Hasta ahora se ha trabajado con el problema (IP) de maximización, pero como todo problema de maximización se puede transformar en un problema de minimización fácilmente, los resultados vistos también son válidos para los problemas de minimización, haciendo claro está una serie de cambios. Nos centraremos ahora en el caso de minimización, ya que los problemas que se abordarán en el Capítulo 3 son de esta clase. De esta forma, el problema:

$$(\text{IP}) \begin{cases} Z = \min & cx \\ \text{s.a} & Ax \leq b \\ & x \in X \end{cases}$$

tiene asociado para cualquier valor de $u = (u_1, ..., u_m) \geq 0$ la relajación:

$$(\text{IP(u)}) \begin{cases} Z(u) = \min & cx - u(b - Ax) \\ & x \in X \end{cases}$$

Su Dual Lagrangiano es:

$W_{LD} = \text{máx}_{u \geq 0}\, Z(u)$

$Z(u) = \text{mín}_{t=1,...,T}[-(b - Ax^t)u + cx^t]$

De esta forma, tenemos que maximizar la función $Z(u)$, que es una función cóncava.

2.4. Algunas consideraciones para el algoritmo de subgradiente

2.4.1. Una forma de seleccionar las longitudes de paso

Vamos a ver una forma de seleccionar las longitudes de paso que se detalla en [2], basada en el apartado c) del Teorema 2 para que la convergencia sea lo más rápida posible. Sea ξ_k el subgradiente $(b - Ax(u^k))$. Nos preguntamos ahora qué longitudes de paso μ_k debemos escoger en cada iteración. El valor ideal será el que minimice $\|u^* - (u^k + \mu\xi_k)\|^2$, donde u^* es el u que optimiza W_{LD}.

Este valor ideal viene dado por $\mu_k^* = \frac{\xi_k^t(u^* - u^k)}{\|\xi_k\|^2}$

Además, por la concavidad de $Z(u)$, tenemos que

$$\mu_k^* = \frac{\xi_k^t(u^* - u^k)}{\|\xi_k\|^2} \geq \frac{Z^* - Z(u_k)}{\|\xi_k\|^2}$$

donde $Z^* = \sup\{Z(u) : u \geq 0\}$.

Una posible aproximación sería

$$\mu_k = \frac{\bar{Z} - Z(u_k)}{\|\xi_k\|^2}$$

donde $\bar{Z} \geq Z^*$. Este valor lo podemos obtener con el valor objetivo de una solución factible del problema de minimización original. Pero en lugar de escoger un valor fijo para $\bar{Z}$, nos gustaría actualizar este valor periódicamente de forma que $\frac{\bar{Z} - Z^*}{\|\xi_k\|^2}$ esté en relación directa al gap de $\mu_k^* = \frac{\xi_k^t(u^* - u^k)}{\|\xi_k\|^2} \geq \frac{Z^* - Z(u_k)}{\|\xi_k\|^2}$. Así, $\bar{Z}$ vendrá dado por

$$\bar{Z} = \alpha_r Z^0 + (1 - \alpha_r) Z^c,$$

donde Z^0 es un número fijo tal que $Z^0 \geq Z^*$, Z^c es el mejor valor objetivo hasta la iteración k, y $\{\alpha_r\}$ es una sucesión monótona decreciente tal que $\alpha_0 = 1$ y $\alpha_r \to \epsilon_0 > 0$.

Relacionaremos la sucesión $\{\alpha_r\}$ con $\{u_k\}$ de la siguiente manera: si tras una cantidad de iteraciones el valor Z^c no mejora, probablemente estemos tomando longitudes de paso demasiado

grandes, por lo que necesitamos reducir $\bar{Z}$. Entonces, incrementamos el valor de r en 1 unidad, haciendo así que α_r y $\bar{Z}$ disminuyan.

Por otra parte, nos interesa evitar que $\bar{Z}$ esté por debajo del valor de Z^*, cosa que puede ocurrir por ejemplo en las primeras iteraciones cuando Z^c está demasiado por debajo de Z^*. Entonces debemos seleccionar $\{\alpha_r\}$ de forma que otorgue más peso a Z^0 en las etapas iniciales, ya que el valor óptimo Z^c en esas etapas estará alejado del óptimo Z^*, y que vaya dando, gradualmente al principio y mas rápido después, más valor a Z^c conforme este valor se aproxima a Z^*. Una sucesión $\{\alpha_r\}$ que optimiza estas exigencias es:

$$\alpha_r = \begin{cases} e^{-0{,}6933(r/r_1)^{3{,}26}} \text{ si } r < r_2 \\ \epsilon_0 \text{ si } r \geq 2 \end{cases}$$

Para la elección de los parámetros r_1 y ϵ_0 conviene probar varios valores y ver cual ofrece mejores resultados. Normalmente si Z^0 está próximo a Z^*, lo deseable será un valor alto para r_1 y uno bajo para ϵ_0, y al contrario, si Z^0 está alejado a Z^* interesará un r_1 bajo y un ϵ_0 alto. Por su parte, r_2 es el menor entero que satisface

$$\epsilon_0 \geq e^{-0{,}6933(r_2/r_1)^{3{,}26}}$$

El procedimiento por el que se llega a esta sucesión $\{\alpha_r\}$ se puede consultar en la referencia indicada al principio de esta sección.

Con todo esto, lo primero que se realiza en el algoritmo es elegir unos multiplicadores de Lagrange iniciales u_0. Se suelen seleccionar todos los multiplicadores nulos. En general, dado un u_k, se obtiene la solución $x(u^k)$ y el subgradiente $\xi_k = (b - Ax(u^k))$ correspondiente. Acto seguido se calcula μ_k de la expresión $\mu_k = \frac{\bar{Z} - Z(u_k)}{\|\xi_k\|^2}$ y obtenemos los multiplicadores de la siguiente iteración u_{k+1} mediante $u_i^{k+1} = max\{u_i^k - \mu_k(b - Ax(u^k)_i), 0\}$ para cada $i = 1, ..., m$. Por el bien de la convergencia, se llamará a esta iteración un éxito o un fracaso en función de si $Z(u_{k+1}) - Z^c \geq \epsilon > 0$ se cumple o no, donde ϵ es un valor de tolerancia previamente fijado. Si se experimentan una cantidad ν de fracasos consecutivos, donde ν también es un valor entero que fijamos al inicio del experimento, entonces incrementaremos el contador r en 1 unidad, actualizamos α_r y retornamos al multiplicador previo que otorgaba la mejor solución hasta el momento. Esto evita que la sucesión $\{u_k\}$ se pierda durante demasiadas iteraciones sin mejorar. Continuamos así hasta que r llegue a r_2. En este momento, α_r se hace igual a ϵ_0 y se congela ese valor, de forma que $\bar{Z} = \epsilon_0 Z^0 + (1 - \epsilon)Z^c$, y se pasarán a seleccionar las longitudes de paso como:

$\mu_k = \frac{1}{\beta_k}[\frac{\bar{Z} - Z(u_k)}{\|\xi_k\|^2}]$

donde $\beta_k = 1$ inicialmente, y se incrementa en 2 unidades cada ν iteraciones sin importar si es un éxito o un fracaso. A partir de esta segunda fase tras llegar r a r_2, el retorno a la mejor solución se realiza mientras β_k sea menor que un entero $\bar{\beta} \geq 1$. Una vez se supera este valor no se realiza el reseteo a la mejor solución hasta el momento.

2.4.2. Condiciones de parada

Antes de comenzar a iterar, se deben establecer una serie de condiciones para que el algoritmo cese. Algunas de las más utilizadas son:

1. Condición sobre las longitudes de paso: Si en alguna iteración k la longitud μ_k es menor que un ϵ previamente fijado se detendrán las iteraciones.

2. Condición sobre el gap relativo: En cada iteración, se tienen una cota superior, procedente de una solución factible, y la mejor cota inferior conseguida hasta el momento. Si el Gap Relativo entre estas cotas $\frac{\bar{Z}_U - \underline{Z}_L}{\bar{Z}_U} \times 100\,\%$, donde $\bar{Z}_U$ es la mejor cota superior hasta el momento y $\underline{Z}_L$ es la mejor cota inferior hasta el momento, es menor que un valor determinado que fijamos previamente, se detendrá el algoritmo.

3. Si el algoritmo realiza un número de iteraciones que hemos elegido previamente como el número máximo de iteraciones que se permitirán, entonces parará.

4. Se puede establecer un tiempo límite tras el que se detenga el proceso iterativo.

2.4.3. División en subproblemas enteros o continuos

En el paso del algoritmo en el que se resuelve el Problema Lagrangiano $IP(u^k)$ se pueden realizar modificaciones. Con el fin de hacer el algoritmo más rápido, se puede resolver la relajación lineal $IP(u^k)$, que otogará una cota inferior menor, es decir, menos buena, que la que proporcionaría $IP(u^k)$ con las variables enteras, pero la calculará más rápido. Las opciones de combinar la resolución de los problemas $IP(u^k)$ con variables continuas o enteras son muy diversas, se puede hacer de la forma que deseemos. Por ejemplo, se puede realizar un número fijado de iteraciones resolviendo la relajación lineal, y realizar las posteriores iteraciones con variables enteras. También se puede iterar con la relajación lineal hasta que el Gap de Optimalidad sea menor que una valor fijado, y a partir de ahí usar variables enteras.

Capítulo 3

Problemas de Asignación Multiperiodo

En este capítulo se definirán los problemas de asignación multiperiodo, y se desarrollarán los problemas Axial y Planar de 3 índices, que son los problemas a los que se le aplicará la Relajación Lagrangiana y el algoritmo de subgradiente vistos en el Capítulo 2. Por último se definirá el Dual Lagrangiano de estos problemas y su descomposición en subproblemas más sencillos.

3.1. Permutaciones y el problema de asignación

Los problemas de asignación consisten en asignar n elementos, como pueden ser por ejemplo agentes, a otros n elementos, como por ejemplo tareas. Una forma en matemáticas de describir las asignaciones es mediante las permutaciones.

Definición 13. *Consideremos un conjunto X de n elementos. Una* ***permutación*** *φ es una aplicación biyectiva que asigna a cada índice $i = 1, ..., n$ un elemento del conjunto X. Se puede representar como $\left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \varphi(1) & \varphi(2) & \cdots & \varphi(n) \end{smallmatrix}\right)$.*

Tal representación significa que i es asignado a $\varphi(i)$ para $i = 1, ..., n$.

Cada permutación φ del conjunto $\{1, 2, \cdots, n\}$ corresponde de forma única a una matriz $n \times n$ de permutación $X_\varphi = (x_{ij})$ donde:

$$x_{ij} = \begin{cases} 1 & \text{si} \quad j = \varphi(i) \\ 0 & \text{en caso contrario.} \end{cases}$$

Veamos un ejemplo de problema de asignación. Supongamos que n agentes deben ser asignados a n tareas, de forma que cada agente solo puede ser asignado a una tarea, y cada tarea ha de ser realizada por un único agente. Consideremos una matriz de costes $n \times n$, $C = (c_{ij})$, donde c_{ij} es el coste de asignar el agente i a la tarea j, y supongamos que queremos minimizar el coste total de las asignaciones, es decir, queremos encontrar la permutación que encuentre

$$\min_{\varphi \in \mathcal{S}_n} \sum_{i=1}^{n} c_{i\varphi(i)}.$$

Siendo $X = (x_{ij})$ una matriz de permutación, se puede formular el problema de asignación como

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 & i = 1, 2, ..., n \\
& \sum_{i=1}^{n} x_{ij} = 1 & j = 1, 2, ..., n \\
& x_{ij} \in \{0, 1\} & i, j = 1, 2, ..., n.
\end{aligned}$$

Un problema de asignación multiperiodo es un problema consistente en asignar n agentes a n tareas en n instantes de tiempo. Sea c_{ijk} el coste de asignar el agente i a la tarea j en el instante k. Buscamos encontrar una asignación φ de los agentes a las tareas y una asignación ψ de los agentes a los instantes de tiempo. Esto nos lleva a Problema de Asignación Axial de 3 índices:

$$\min_{\varphi, \psi \in \mathcal{S}_n} \sum_{i=1}^{n} c_{i\varphi(i)\psi(i)},$$

donde $\mathcal{S}_n$ denota el conjunto de todas las permutaciones de los enteros $\{1, 2, ..., n\}$.

Este Problema Axial, junto a otro problema multiperiodo conocido como el Problema Planar, se formularán en las posteriores secciones y serán los modelos que se implementarán en el estudio computacional. Sin embargo, estos nos son los únicos problemas de asignación multiperiodo que existen. Esta clase de problemas es muy diversa y tiene múltiples aplicaciones prácticas. Veamos a continuación algunos modelos para ciertos problemas y la motivación de su planteamiento y resolución.

3.1.1. Aplicaciones de problemas de asignación

Los problemas de asignación tienen diversas aplicaciones en diferentes campos. En primer lugar se plantea el problema de horarios en la universidad. En este problema se busca compatibilizar los horarios de alumnos, profesores y demás personal de acuerdo a una serie de normas o restricciones. Algunas de ellas son:

- No se permiten colisiones. Una colisión ocurre cuando dos o más cursos se programan a la vez para el mismo profesor, en el mismo aula o para el mismo grupo de alumnos. También cuando dos o más profesores se asignan al mismo grupo de alumnos para impartir la misma asignatura; y por último, cuando dos o más aulas se asignan al mismo grupo de alumnos y al mismo curso.

- Un horario debe estar completo. Un horario se considera completo cuando aparecen todos los cursos planificados para cada grupo de estudiantes en el horario, con la cantidad correcta de períodos de tiempo para cada curso y cada parte de cada curso. Además, cuando a cada miembro del personal docente se le asigna el número total de períodos de enseñanza que requiere la institución.

- Un horario debe poder adaptarse a solicitudes de sesiones de períodos de enseñanza consecutivos. Dependiendo del curso y el número de períodos de enseñanza asignados por semana, un profesor puede optar por impartir el curso en sesiones de un solo período o de varios períodos. El horario debe ser capaz de programar un curso determinado en cualquier esquema que el profesor responsable elija seguir.

- Un horario debe poder adaptarse a solicitudes de sesiones repetidas de un curso dado o parte de un curso. Esta regla se refiere a la necesidad presentada por algunos cursos de sesiones de repaso o trabajos de laboratorio diseñados para grupos pequeños. En este caso, la misma sesión debe repetirse varias veces para acomodar el número total de estudiantes registrados.

El problema se puede consultar en detalle en [5], artículo en el que se desarrolla la formulación con más variables y restricciones que las aquí presentadas.

Otro problema es el de realizar una planificación eficiente de actividades culturales y eventos, teniendo en cuenta diferentes partes involucradas como ayuntamientos, empresas y administración, entre otras cosas. Este problema se aborda en [12], motivado por el deseo de los municipios de brindar una oferta cultural atractiva con respecto a una combinación de preferencias, principalmente relacionadas con la elección de empresas y del calendario. Por su

parte, las compañías artísticas tratan de maximizar sus beneficios actuando el mayor tiempo posible en los municipios más atractivos. La administración tiene como objetivo maximizar el beneficio social. Así, se propone medir la calidad de las soluciones relacionando esos factores.

Algunas restricciones que se modelan en el artículo son:

- Una misma compañía no puede actuar el mismo día.
- Ningún agente puede actuar por j-ésima ocasión en un lugar más de 1 vez.
- Se prohíbe que dos actuaciones coincidan el mismo día y en el mismo lugar.
- Ningún sitio puede exceder el presupuesto permitido para una modalidad, y se otorga un descuento en el coste por la j-ésima repetición de un agente en el sitio.

También se puede encontrar aplicaciones dirigidas a servicios de cuidado de hogares, como se aborda en [13]. El problema consiste en asignar personal médico a hogares de clientes para llevar a cabo determinados cuidados. En este problema se establecen restricciones como la obligación de que todos los pacientes sean atendidos algún día, establecer la dependencia de la realización de unos cuidados de otros cuidados previos, o la prohibición de que un equipo vuelva al hospital y luego salga de este el mismo día.

Otras aplicaciones están relacionadas con los modelos Planar y Axial que se definirán en las siguientes secciones. En [4] se recogen las aplicaciones que mencionaremos a continuación, y recoge referencias a estudios de estas aplicaciones. El modelo Planar tiene aplicaciones directas en modelos de horarios y en la asignación de intervalos de tiempo para sistemas de telecomunicaciones, que usan satélites que deben comunicarse con la Tierra. El modelo Axial tiene aplicaciones en diversos campos, siendo útil por ejemplo para invertir capital en diferentes localizaciones durante una cantidad de tiempo, para modelar el ensamblaje de placas de circuitos o para la laminación de lingotes.

3.2. Problema de asignación Planar de 3 índices

Definimos el Problema de Asignación Planar de 3 índices (Planar 3AP) en lo que sigue. Se dice que n permutaciones $\varphi_1, \varphi_2, ..., \varphi_n$ son mutuamente distintas si $\varphi_r(i) \neq \varphi_s(i)$ para cualquier $i = 1, 2, ..., n$ y $r \neq s$. Dados n^3 coeficientes de costes c_{ijk} $(i, j, k = 1, 2, ...n)$, el problema consiste en encontrar n permutaciones mutuamente distintas tal que:

$$\sum_{k=1}^{n} \sum_{i=1}^{n} c_{i\varphi_k(i)k}$$

sea mínimo. Su correspondiente programación lineal entera es:

$$\text{mín} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{ijk} = 1 \qquad i,j = 1,2,...,n \tag{3.2}$$

$$\sum_{j=1}^{n} x_{ijk} = 1 \qquad i,k = 1,2,...,n \tag{3.3}$$

$$\sum_{i=1}^{n} x_{ijk} = 1 \qquad j,k = 1,2,...,n \tag{3.4}$$

$$x_{ijk} \in \{0,1\} \qquad i,j,k = 1,2,...,n \tag{3.5}$$

La variable x_{ijk} tomará el valor 1 si se asigna el agente i a la tarea j en el instante k. De esta forma, la función objetivo (3.1) nos indica que hay que minimizar los costes de las combinaciones que hemos escogido en las asignaciones. La restricción (3.2) nos indica que cada agente debe realizar cada tarea exactamente en un único instante de tiempo. De forma análoga, (3.3) nos fuerza a que cada agente debe estar realizando una y solo una tarea en cada instante de tiempo. Por último, (3.4) nos dice que cada tarea se debe estar llevando a cabo en cada instante de tiempo por un único agente.

El Planar 3AP está muy relacionado con los cuadrados Latinos. Un cuadrado Latino es un conjunto de $n \times n$ entradas l_{ij} que toman los valores de 1 a n. Cada fila y cada columna de un cuadrado Latino contiene exactamente una entrada con el valor k, con $1 \leq k \leq n$. Cada solución factible del Planar 3AP se puede representar como un cuadrado Latino. Sea $L = (l_{ij})$ un cuadrado Latino de tamaño n. Entonces, para $i,j = 1,2,...,n$, l_{ij} es el único valor de índice k tal que $x_{ijk} = 1$ en una solución factible de Planar 3AP.

Un ejemplo de cuadrado latino sería:

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix}$$

Este cuadrado latino corresponde a la asignación:

- agente 1, tarea 1 e instante de tiempo 1.
- agente 1, tarea 2 e instante de tiempo 2.

- agente 1, tarea 3 e instante de tiempo 3.
- agente 2, tarea 1 e instante de tiempo 2.
- agente 2, tarea 2 e instante de tiempo 3.
- agente 2, tarea 3 e instante de tiempo 1.
- agente 3, tarea 1 e instante de tiempo 3.
- agente 3, tarea 2 e instante de tiempo 1.
- agente 3, tarea 3 e instante de tiempo 2.

En concreto, este cuadrado latino es un cuadrado latino reducido, es decir, un cuadrado latino que tiene la permutación identidad en la primera fila y en la primera columna.

El problema Planar 3AP tiene muchas aplicaciones a la hora de organizar y repartir tareas en un horario para distintos empleados, pero es un problema $\mathcal{NP}$-duro complicado de resolver (ver [17]). Y es que el número de soluciones factibles, es decir, de cuadrados latinos existentes, aumenta muy rápido en función de n. Por ejemplo, si $n = 9$, el número de soluciones factibles sería aproximadamente de 5510^{26} de acuerdo con Bammel y Rothstein en [1]. En [9] se muestra el número de cuadrados latinos reducidos hasta orden 11, y se afirma que el número de cuadrados latinos hasta orden n es igual al número de cuadrados latinos reducidos hasta ese orden multiplicado por $n!(n-1)!$. Se comprueba en la siguiente tabla que el número de cuadrados latinos reducidos (y por tanto también los totales) aumenta a una velocidad muy grande conforme aumenta n. Por este motivo usaremos la relajación lagrangiana, que puede ayudarnos a obtener soluciones cercanas a la óptima en un tiempo aceptable.

Tabla 3.1: Número de cuadrados latinos reducidos de orden n

n	cuadrados latinos reducidos
1	1
2	1
3	1
4	4
5	56
6	9408
7	16942080
8	535281401856
9	377597570964258816
10	7580721483160132811489280
11	5363937773277371298119673540771840

No se conocen muchos algoritmos para resolver el Planar 3AP. El primer algoritmo de ramificación y acotación se remonta a [16], quien calculó cotas inferiores mediante reducción

de filas y columnas. El algoritmo Branch and Bound (ramificación y acotación) es una técnica utilizada en la optimización combinatoria para encontrar soluciones óptimas a problemas de búsqueda exhaustiva. El algoritmo divide el espacio de búsqueda en regiones más pequeñas (ramificación) y utiliza cotas superiores e inferiores para descartar regiones que no contienen soluciones óptimas (acotación), evitando así explorar innecesariamente todo el espacio de búsqueda. En [7] se implementa un algoritmo basado en los cuadrados latinos. Se fijaron en que un movimiento en la vecindad de un cuadrado Latino queda determinado cambiando el contenido de una determinada celda (i, j). Esto afecta, al menos, entre 3 y $2n-1$ otras celdas que deben adaptarse. Gracias a esta estructura, se dan propiedades que favorecen la computación. Los resultados numéricos de este algoritmo muestran un buen equilibrio entre el tiempo de cálculo y calidad de la solución para instancias del Planar 3AP de tamaño hasta $n = 14$. Más resultados sobre algoritmos para el Planar 3AP pueden encontrarse en [6].

3.3. Problema de asignación Axial de 3 índices

El Problema de asignación Axial de 3 índices (Axial 3AP) se formula de la siguiente manera. Sean n^3 coeficientes de costes c_{ijk} con $i, j, k = 1, 2, ..., n$. Buscamos dos permutaciones φ y ψ tales que $\sum_{i=1}^{n} c_{i\varphi(i)\psi(i)}$ sea mínimo, es decir,

$$\min_{\varphi,\psi\in\mathcal{S}_n} \sum_{i=1}^{n} c_{i\varphi(i)\psi(i)}$$

donde $\mathcal{S}_n$ denota el conjunto de todas las permutaciones de los enteros $\{1, 2, ..., n\}$. Como las dos permutaciones que describen una solución factible pueden ser elegidas arbitrariamente, el Axial 3AP tiene $(n!)^2$ soluciones factibles. Podemos escribir el problema como la programación lineal entera:

$$\text{mín} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \tag{3.6}$$

$$\text{s.t.} \quad \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 \qquad i = 1, 2, ..., n \tag{3.7}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk} = 1 \qquad j = 1, 2, ..., n \tag{3.8}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \qquad k = 1, 2, ..., n \tag{3.9}$$

$$x_{ijk} \in \{0, 1\} \qquad i, j, k = 1, 2, ..., n \tag{3.10}$$

La variable x_{ijk} tomará el valor 1 si se asigna el agente i a la tarea j en el instante k, y la función objetivo (3.6) representa el coste de las asignaciones que se escogen, de manera idéntica a la del Planar 3AP. La diferencia está en las restricciones. La restricción (3.7) nos dice que cada agente solo debe tener asignada una única tarea y en un único instante de tiempo. Recordemos que en el Planar 3AP cada agente llevaba a cabo todas las tareas en los diferentes instantes de tiempo, pero en este caso cada agente solo realizará una tarea y en un único instante de tiempo. De igual forma, por (3.8) cada tarea es realizada por un único agente en un único instante de tiempo, y por (3.9) en cada instante de tiempo solo se está realizando una tarea y por parte de un solo agente.

Como se explica en [4], como observamos por las restricciones, existirán menos soluciones factibles que en el Planar 3AP al ser las restricciones más duras. Sin embargo, el Axial 3AP sigue siendo un problema $\mathcal{NP}$-duro. Al ser complicado de resolver para valores grandes de n, también trataremos de aproximar la solución óptima mediante la relajación lagrangiana.

Entre las heurísticas conocidas para aproximar el Axial 3AP se encuentran algunas que realizan una búsqueda adaptativa aleatoria greedy, con buenos resultados computacionales, y algoritmos híbridos. También existen algoritmos heurísticos en el caso en que los costes se puedan descomponer de la forma $c_{ijk} = a_i b_j c_k$, y heurísticas y cotas inferiores para la generalización en la que la descomposición es de la forma $c_{ijk} = a_i d_{jk}$. En [4] se referencian muchos trabajos sobre estos algoritmos para su consulta.

3.4. El Dual Lagrangiano de los problemas de asignación multiperiodo Planar y Axial

En esta sección vamos a formular el Dual Lagrangiano de los problemas Planar 3AP y Axial 3AP vistos enteriormente en este capítulo. Recordemos que para obtener una relajación lagrangiana podemos relajar tantas restricciones como queramos.

En el caso del problema Planar 3AP relajaremos por ejemplo la restricción (3.2):

$$
\begin{aligned}
Z(u) = \text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} + \sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij}(1 - \sum_{k=1}^{n} x_{ijk}) & \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ijk} = 1 & i,k = 1,2,...,n \\
& \sum_{i=1}^{n} x_{ijk} = 1 & j,k = 1,2,...,n \\
& x_{ijk} \in \{0,1\} & i,j,k = 1,2,...,n
\end{aligned}
$$

Su Problema Dual Lagrangiano es:

$$
\begin{aligned}
W_{LD} = \max_{u} \quad (\text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} + \sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij}(1 - \sum_{k=1}^{n} x_{ijk})) & \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ijk} = 1 & i,k = 1,2,...,n \\
& \sum_{i=1}^{n} x_{ijk} = 1 & j,k = 1,2,...,n \\
& x_{ijk} \in \{0,1\} & i,j,k = 1,2,...,n
\end{aligned}
$$

Además, la relajación lagrangiana $Z(u)$ se puede dividir en n subproblemas $Z_k(u)$ de forma que $Z(u) = \sum_{k=1}^{n} Z_k(u) + \sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij}$, donde

$$
\begin{aligned}
Z_k(u) = \text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ijk} - u_{ij})x_{ijk} & \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ijk} = 1 & i = 1,2,...,n \\
& \sum_{i=1}^{n} x_{ijk} = 1 & j = 1,2,...,n \\
& x_{ijk} \in \{0,1\} & i,j = 1,2,...,n
\end{aligned}
$$

En el caso del problema Axial 3AP relajaremos, por ejemplo, las restricciones (3.7) y (3.8):

$$\begin{aligned} Z(u,v) = \text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \\ & + \sum_{i=1}^{n} u_i(1 - \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk}) + \sum_{j=1}^{n} v_j(1 - \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk}) \\ \text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \qquad\qquad k = 1,2,...,n \\ & x_{ijk} \in \{0,1\} \qquad\qquad i,j,k = 1,2,...,n \end{aligned}$$

Su Problema Dual Lagrangiano es:

$$\begin{aligned} W_{LD} = \max_{u} \quad (\text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}x_{ijk} \\ & + \sum_{i=1}^{n} u_i(1 - \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk}) + \sum_{j=1}^{n} v_j(1 - \sum_{i=1}^{n}\sum_{k=1}^{n} x_{ijk})) \\ \text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \qquad\qquad k = 1,2,...,n \\ & x_{ijk} \in \{0,1\} \qquad\qquad i,j,k = 1,2,...,n \end{aligned}$$

Al igual que en el caso del Planar, la relajación lagrangiana $Z(u,v)$ se puede dividir en n subproblemas $Z_k(u,v)$ de forma que $Z(u,v) = \sum_{k=1}^{n} Z_k(u,v) + \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$, donde

$$\begin{aligned} Z_k(u,v) = \text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ijk} - u_i - v_j)x_{ijk} \\ \text{s.t.} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ijk} = 1 \\ & x_{ijk} \in \{0,1\} \qquad\qquad i,j = 1,2,...,n \end{aligned}$$

Capítulo 4

Estudio computacional

En este capítulo se resolverán usando un solver comercial los modelos planteados para los problemas de asignación multiperiodo Axial y Planar. También se resolverá el Dual Lagrangiano del modelo Axial 3AP mediante un algoritmo de subgradiente implementado en Python, con el que se obtendrán soluciones exactas y aproximadas, valorando su rendimiento en función de varios parámetros.

4.1. Python y el solver Gurobi

La implementación del código necesario para resolver los problemas de asignación multiperiodo Axial y Planar 3AP se ha llevado a cabo en el lenguaje de programación Python. Python es un lenguaje de programación interpretado que da gran importancia a la legibilidad de su código. Posee una licencia de código abierto y es uno de los lenguajes de programación más populares. Python es un lenguaje de programación multiparadigma, es decir, que en lugar de forzar a los programadores a emplear un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional.

Por otra parte, se hará uso del solver Gurobi, un software enfocado en la optimización matemática que puede resolver problemas de Programación Lineal y Programación Lineal Entera Mixta, entre otros. Llamaremos a este solver desde Python.

4.2. Instancias

A la hora de generar instancias para los problemas, bastará escoger una matriz de costes para cada tamaño n. Sus entradas serán valores generados aleatoriamente entre 1 y 100, $c_{ijk} \in [1, 100]$. Esta matriz de costes será la misma para cada n, ya que establecemos antes de generar la matriz una semilla para que el experimento sea reproducible.

4.3. Cotas Primales iniciales

Para la implementación del algoritmo de subgradiente es fundamental contar con una cota primal de inicio. Esto nos motiva a aplicar un algoritmo heurístico que, idealmente, nos proporcione una cota primal cercana al óptimo. Una heurística es un tipo de algoritmo que permite encontrar soluciones aproximadas o subóptimas para un problema, especialmente cuando se carece de algoritmos exactos o cuando la búsqueda exhaustiva de todas las posibilidades es impracticable en términos de tiempo o recursos computacionales. A diferencia de los algoritmos determinísticos, las heurísticas no garantizan encontrar la solución óptima, pero permiten encontrar una solución satisfactoria o lo más cercana posible a la solución óptima en un tiempo razonable.

Cabe recordar que, como los problemas que nos ocupan son de minimización, las cotas primales son cotas superiores, a diferencia de lo visto en el Capítulo 1, en el que se trataban problemas de maximización. Con el objetivo de conseguir una cota primal inicial satisfactoria para los modelos Planar 3AP y Axial 3AP, se implementará en Python un algoritmo heurístico voraz (o *greedy*) para el problema Axial 3AP, y otro algoritmo que proporciona el valor objetivo de un cuadrado latino en el caso del Planar 3AP. Un algoritmo *greedy* es un enfoque heurístico que toma decisiones en cada etapa del problema seleccionando la opción óptima localmente en ese momento, con la esperanza de que dichas elecciones locales conduzcan a una solución globalmente óptima.

Un ejemplo de solución factible para el problema Axial vendría dada por asignar el agente i a la tarea i en el instante de tiempo i. Otra posibilidad es la que se muestra en el Algoritmo 2. En este algoritmo se asigna a cada agente i la tarea j en el instante de tiempo k que sea de menor coste entre todas las del agente, y una vez seleccionados j y k, se bloquea la posibilidad de seleccionar esa tarea y ese instante de tiempo en los agentes restantes, asignándoles un nuevo coste M (que solo se usa en el algoritmo de manera auxiliar) que es superior a cualquier coste $c_{ijk} \quad \forall i, j, k = 1, \cdots, n$.

Algoritmo 2: Algoritmo Greedy para el Axial 3AP

input : Se solicita una matriz de costes $n \times n \times n$, $C = (c_{ijk})_{n \times n \times n}$
output: Una cota superior o primal Z_{UB}

```
Se fija un valor M tal que M > c_ijk   ∀i, j, k = 1, ··· , n
for cada agente i = 1, ··· , n do
    Asigna el agente i a la tarea j y el tiempo k que cumplen que c_ijk es el menor valor
     entre los c_ij*k*   ∀j*, k* = 1, ··· , n
    for cada agente i* = 1, ··· , n do
        for cada tiempo k* = 1, ··· , n do
            c_i*jk* = M
    for cada agente i* = 1, ··· , n do
        for cada tarea j* = 1, ··· , n do
            c_i*j*k = M
```

Algoritmo 3: Algoritmo para el Planar 3AP

input : Se solicita una matriz de costes $n \times n \times n$, $C = (c_{ijk})_{n \times n \times n}$
output: Una cota superior o primal Z_{UB}

```
contador=0
for cada agente i = 1, ··· , n do
    for cada tiempo k = 1, ··· , n do
        if k + contador < n then
            Asigna la tarea i e instante k a la tarea k + contador
        if k + contador ≥ n then
            Asigna la tarea i e instante k a la tarea k + contador − n
    contador = contador + 1
```

En el caso del Planar 3AP, no resulta evidente obtener una cota primal. Una opción que implementamos en Python es la descrita en el Algoritmo 3, que selecciona un cuadrado latino como solución factible y evalúa su valor objetivo. El cuadrado latino que selecciona el algoritmo es el siguiente:

1. Para el agente 1: asigna la tarea 1 al tiempo 1, la tarea 2 al tiempo 2, ...

2. Para el agente 2: asigna la tarea 2 al tiempo 1, la tarea 3 al tiempo 2, ...

3. Para el agente 3: asigna la tarea 3 al tiempo 1, la tarea 4 al tiempo 2, ...

Existen otros métodos más complicados para el Planar 3AP, como el descrito en [8]. En este artículo se expone un algoritmo de ramificación y acotación para el Planar 3AP, de forma que, en lugar de ramificar una sola variable, ramifica un conjunto de variables que se encuentran en una misma restricción. El procedimiento para encontrar una cota superior encuentra primero una solución inicial y posteriormente la mejora, mientras que para las cotas inferiores combina heurísticas duales y la relajación lagrangiana. En particular, en el procedimiento de mejora de la cota superior, dada una solución factible, es decir, un cuadrado latino, se le aplica una mejora local que encuentra una solución *vecina* con menor valor objetivo. Se representa la solución factible como un cuadrado latino, es decir, en forma de tabla $n \times n$ en el que en la entrada (i, j) se encuentra el índice k al que se asignan el agente i y la tarea j, y se cambia el valor de alguna entrada. Eso convierte la solución en infactible. Entonces, se desplaza desde la entrada modificada mediante filas y columnas hasta que se encuentra el índice k que está repetido, y se cambia por el que fue modificado. Al acabar, se tiene un cuadrado latino diferente al original, y se evalúa su valor objetivo para ver si es menor que el del cuadrado latino original.

4.4. Cotas primales y duales del algoritmo exacto

Para cada problema de asignación multiperiodo, se definieron funciones `PLANAR` y `AXIAL` que, mediante el solver Gurobi, y al darle como datos de entrada la matriz de costes $(c_{ijk})_{n\times n\times n}$, resuelve los problemas Planar 3AP y Axial 3AP con un algoritmo exacto de ramificación y corte. Además del valor óptimo de la relajación lineal del problema, Gurobi obtiene cotas superiores e inferiores que cuando coinciden nos garantizan haber encontrado el valor óptimo del problema. A estas funciones se les puede poner un tiempo límite, de forma que si llegado ese tiempo todavía no han coincidido las cotas superior e inferior, la función detendrá los cálculos y devolverá el valor de estas cotas inferior y superior que tenía en ese instante.

4.5. Parámetros iniciales del algoritmo de subgradiente

Ahora que ya tenemos las herramientas para hallar el óptimo, la relajación lineal y cotas primales y duales de los problemas Planar 3AP y Axial 3AP, pasamos a implementar el algoritmo de subgradiente. Debemos fijar una serie de parámetros ya descritos en el Capítulo 2 antes de comenzar a iterar. Estos son:

- Multiplicadores de Lagrange iniciales u^0. Los elegiremos nulos de inicio, $u^0 = 0$.
- `Max_Iter`: El número de iteraciones máximo que se realizarán hasta detener el algoritmo.
- `Scale`: es un parámetro proporcional a la longitud de paso en cada iteración. Desempeña un papel similar al de η_k en el apartado c) del Teorema 2 del Capítulo 2, o al de $\frac{1}{\beta_k}$ descrito en la sección 2.3. Su valor debe estar entre 0 y 2.
- `SameLimit`: número máximo de iteraciones seguidas en las que la cota inferior no se actualiza. De llegar a esta cantidad de iteraciones seguidas en las que ocurra esto, se dividirá el `Scale` entre 2 y el `SameLimit` se reseteará a 0. Tiene el papel de ν de la sección 2.3.
- ϵ: valor positivo, de forma que si el Optimality Gap es menor que esta cantidad, entonces se detendrá el algoritmo. En nuestro caso elegiremos $\epsilon = 10^{-4}$.
- ϵ_{step}: valor positivo, de forma que si la longitud de paso calculada en alguna iteración es inferior a este valor, se detendrá el algoritmo.

4.6. Algoritmo de subgradiente

A continuación se explica cómo se aplicará el algoritmo de subgradiente visto en el Capítulo 2.

PASO 0:

Fijamos los parámetros iniciales tal y como se establecen en la Sección 4.5.

PASO 1:

Nos situamos en la iteración k. Contamos con los multiplicadores u^k, con los que se resolverá la relajación lagrangiana con la función `lower_boundP` o `lower_boundA`, según se trate el modelo Planar o Axial, y se obtendrá una cota dual o inferior. Las funciones `lower_boundP` y `lower_boundA` tienen como dato de entrada los multiplicadores de Lagrange u, junto a la matriz de costes (c_{ijk}), y devuelve la solución de la relajación lagrangiana de los problemas

Planar 3AP y Axial 3AP, así como su valor objetivo. Este supone una cota dual o inferior. Antes de empezar las iteraciones también es necesaria una cota inferior inicial. En nuestro experimento se escogerá una cota inferior trivial, que es el valor 0. En otros casos también serviría el valor objetivo óptimo de la relajación lineal del problema original aunque en algunas ocasiones, como en los problemas que nos ocupan, la fuerza del dual lagrangiano no permite que el valor de la relajación lineal sea superado. Además, este trabajo también pretende comprobar la viabilidad del algoritmo de subgradiente como alternativa al cálculo de la relajación lineal mediante el solver Gurobi.

En caso de que la cota inferior obtenida de la relajación lagrangiana sea mayor que la cota inferior que se poseía al iniciar la iteración, se actualizará como la mejor cota inferior hasta el momento. Si la cota inferior no se actualiza, se sumará una unidad al valor `same`, que es una variable auxiliar que definimos antes de comenzar a iterar y tiene como valor inicial 0. Esta variable servirá para llevar la cuenta del número de veces que no se actualiza la cota inferior. Si el valor de `same` iguala a `SameLimit`, entonces, tal y como explicamos antes, se dividirá el valor de `scale` entre 2 para calcular la longitud de paso y se reseteará la cuenta de `same` a 0.

PASO 2:

Con la solución $x(u^k)$ del problema relajado calcularemos los `slacks` o subgradientes ξ_k como: $b - Ax(u^k)$ en el caso general. Si el subgradiente es nulo, entonces significa que la solución $x(u^k)$ de la relajación lagrangiana en esta iteración cumple las restricciones relajadas y por lo tanto es factible en el problema original, por lo que su valor objetivo en el problema original se evaluará con la función que definimos para encontrar cotas primales: `upper_boundP` o `upper_boundA`. Estas funciones tienen como argumento la matriz de costes (c_{ijk}) y una solución factible x, y devuelven el valor objetivo en el problema original de la solución factible x. Estas funciones sirven para evaluar el valor objetivo de una solución factible en el caso de que el algoritmo de subgradiente encuentre alguna en alguna iteración.

Si la cota superior de la solución factible es mejor, i.e., inferior a la que teníamos hasta el momento, la sustituirá como mejor cota superior hasta el momento. Si por el contrario el subgradiente no es nulo, la longitud de paso μ_k se calculará como

$$\mu_k = \texttt{scale} * \frac{Z_{UB} - Z_D}{\|\xi_k\|^2}$$

donde Z_{UB} es la cota superior y Z_D es el valor objetivo de la relajación lagrangiana con multiplicadores u^k resuelto anteriormente.

PASO 3:

Con las longitudes de paso ya calculadas, que son las descritas en el apartado c) del teorema 2 de la sección 2.3, se obtienen los multiplicadores de Lagrange de la siguiente iteración:

$$u^{k+1} = u^k + \mu_k \xi_k$$

Estos multiplicadores son irrestrictos en signo debido a que las restricciones relajadas tanto en el Planar 3AP como en el Axial 3AP son de igualdad. Con estos multiplicadores u^{k+1} se dará paso al Paso 1 de la siguiente iteración, siempre que no se cumpla alguna condición de parada: se verá si el paso μ_k es menor que 10^{-8} y si el Optimality Gap es menor que el $\epsilon = 10^{-4}$ que fijamos al principio, y de cumplirse alguna de estas condiciones, se detendrá el algoritmo. En caso contrario, se continuará iterando. El proceso también terminará si se alcanza el número máximo de iteraciones fijado.

4.7. Algoritmo de reparación

En esta sección se explica cómo se ha implementado un algoritmo que toma la mejor solución que ha encontrado el algoritmo de subgradiente aplicado al modelo Axial 3AP y la transforma en una solución factible lo más parecida posible.

PASO 1:

El algoritmo empieza con una solución inicial con todas las entradas nulas. Para cada agente i, obtiene el valor más grande de la solución iterada en ese agente y lo llama x_{max}. Este valor generalmente será 1.

PASO 2:

Para cada tarea y cada instante de tiempo, compara los costes de las entradas en las que la solución iterada vale x_{max}, y selecciona y guarda los índices de la tarea y el instante de tiempo (j, k) de menor coste, en caso de que se pueda. Esto es debido a que puede que no se seleccione ninguna tarea ni instante de tiempo por el siguiente motivo: en el caso en el que no se seleccione ningún (j, k), significará que ninguna de las entradas con valor x_{max} en la solución iterada se puede seleccionar por factibilidad, pues han sido bloqueados por los anteriores índices seleccionados en iteraciones previas. En este caso, se tendrán que seleccionar los índices de la entrada (j, k) de menor coste en el agente i, aunque tenga un valor 0 en la solución iterada.

PASO 3:

Una vez tenemos el par de índices (j, k) para el agente i, se establece el valor de la solución reparada $x_{ijk} = 1$, se bloquean todas las entradas de los agentes restantes correspondientes a la tarea j y el instante de tiempo k, y se pasa al siguiente agente. Al finalizar con la asignación del último agente, el algoritmo ha elaborado una solución parecida a la iterada que además es factible.

En la implementación, el algoritmo va recorriendo los agentes por orden, de modo que al haber menos entradas bloqueadas en los primeros, la solución se parecerá más a la iterada en los primeros agentes y en los últimos se parecerá menos. Se podría recorrer los agentes en otro orden en función de si se quisiera priorizar más que se parezca en según qué agentes. Otra posible variación del algoritmo sería recorrer las tareas o los instantes de tiempo e ir seleccionando y bloqueando los otros índices.

4.8. Variantes de resolución

Para resolver la relajación lagrangiana, no solo se pueden usar las funciones `lower_boundP` o `lower_boundA`. También se programaron las funciones `lower_boundPk` o `lower_boundAk` que resuelven los k subproblemas en los que se puede dividir la relajación lagrangiana y devuelven el valor objetivo del subproblema. Una vez resueltos los subproblemas, se agregan sus valores para obtener el mismo valor que se obtenía usando las anteriores funciones pero de forma más eficiente.

También se puede alternar la resolución de los problemas con las variables enteras o continuas como se explicó en la Sección 2.4.4.

4.9. Resultados

Los resultados que se detallan a continuación se han obtenido un ordenador con un procesador Intel Core i7 de 2GHz y 8GB de memoria RAM. La versión de Gurobi instalada es la 10.0.1.

4.9.1. Resolución con algoritmo exacto

Vamos a ver los resultados obtenidos al resolver diversas instancias de los modelos Axial 3AP y Planar 3AP con Gurobi. Veremos para cada modelo una tabla cuyas columnas corresponden a los siguientes valores:

- n: el tamaño de la instancia.
- $g\bar{U}R$: el gap entre la cota superior que encuentra Gurobi y el valor de la relajación lineal.
- gUL: el gap entre la cota superior y la cota inferior que encuentra Gurobi.
- $Tiempo$: el tiempo en segundos que emplea Gurobi hasta encontrar la solución óptima. Está limitado a 300 segundos para el Axial y 60 segundos en el Planar, de modo que si llega a este tiempo, acabará la iteración que esté realizando y devolverá las cotas superior e inferior que tenía en ese momento.
- $Tiempo_{RL}$: el tiempo en segundos empleado en calcular la relajación lineal. Está limitado a 120 segundos en el Axial y a 60 segundos en el Planar.
- z^*: óptimo calculado por Gurobi.
- z_{RL}: valor de la relajación lineal.

n	$g\bar{U}R$	gUL	$Tiempo$	$Tiempo_{RL}$	z^*	z_{RL}
5	0	0	0.01	0.01	44.62	44.62
10	0	0	0.08	0.11	19.29	19.29
15	14.2	0	0.64	0.22	35.64	30.58
20	5	0	1.05	0.46	36.39	34.57
55	2.48	1.8	$\geq$310.31	10.06	-	59.76
125	-	-	-	$\geq$120	-	-

Tabla 4.1: Resultados del algoritmo exacto del Axial 3AP

n	$g\bar{U}R$	gUL	$Tiempo$	$Tiempo_{RL}$	z^*	z_{RL}
5	0	0	0.02	0.14	712.66	712.66
6	2.88	0	0.33	0.09	946.2	918.98
8	2.64	0	0.57	0.05	1552.17	1511.04
11	4.15	0	17.44	0.11	2429.74	2328.89
15	12.63	12.05	$\geq$ 60	0.44	-	3421.4
60	-	-	-	$\geq$ 60	-	-

Tabla 4.2: Resultados del algoritmo exacto del Planar 3AP

En ambos modelos se observan buenos resultados de tiempo y *gap* pequeños en las instancias de menor tamaño n, y a medida que este tamaño aumenta, se observa que el tiempo y el *gap* también lo hacen. Para $n = 55$ en el Axial 3AP y para $n = 15$ en el Planar 3AP, se observa que el tiempo empleado para obtener el óptimo supera a los tiempos que habíamos fijado previamente, pero sigue siendo capaz de resolver la relajación lineal. Además, en estas instancias los *gap* son de 2.48 en el Axial y de 12.63 en el Planar, por lo que Gurobi sigue dando una buena acotación

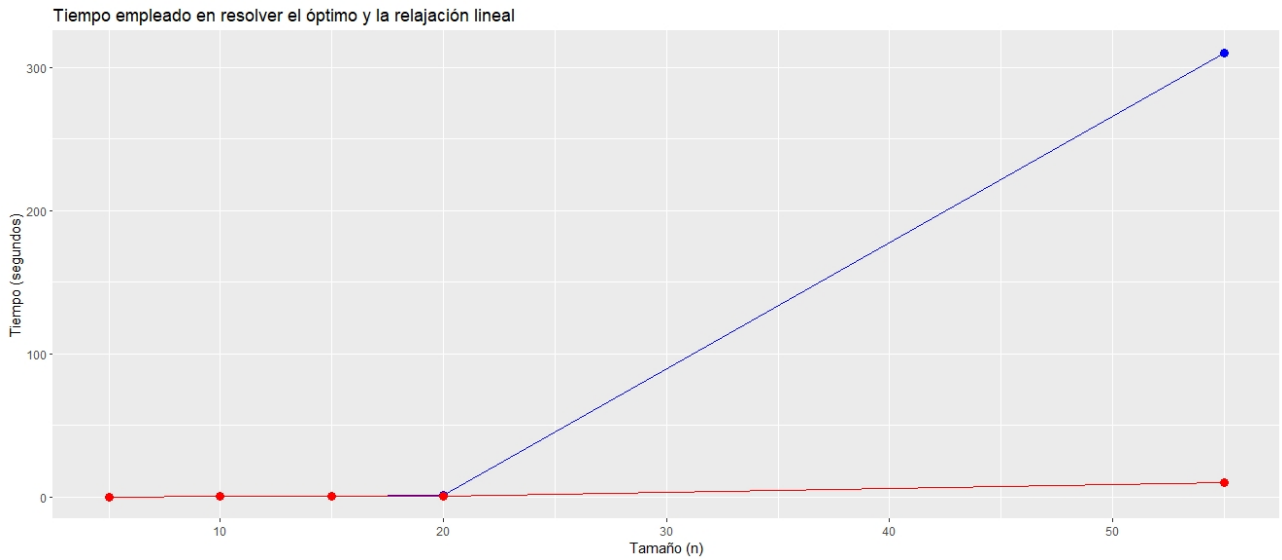


Figura 4.1: Modelo Axial 3AP. En azul, tiempo del óptimo. En rojo, tiempo de la relajación lineal.

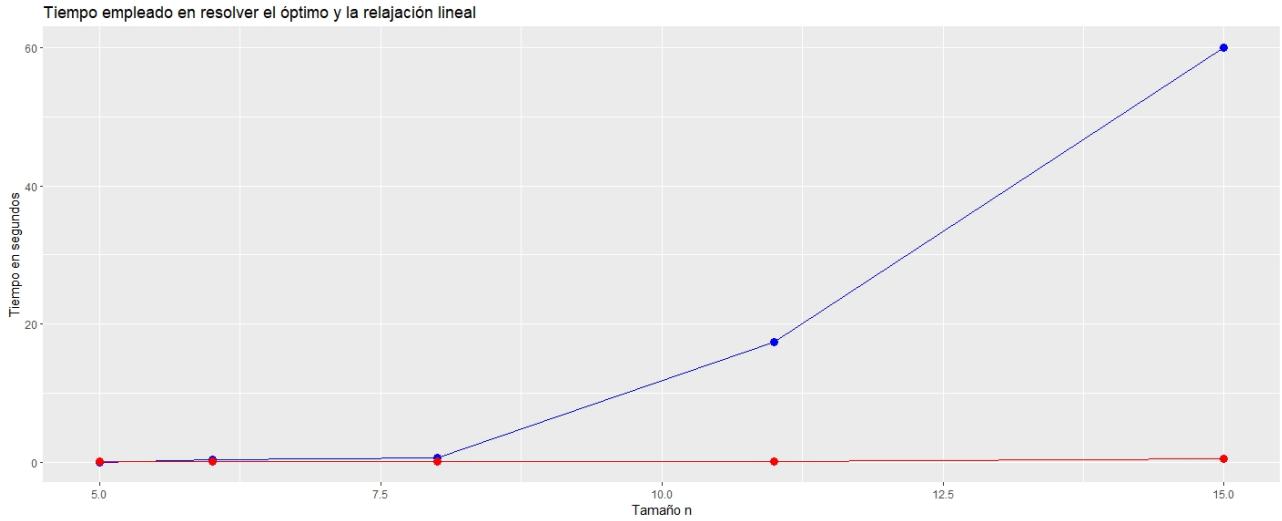


Figura 4.2: Modelo Planar 3AP. En azul, tiempo del óptimo. En rojo, tiempo de la relajación lineal.

con *gaps* bajos. En cambio, para $n = 125$ en el Axial 3AP y para $n = 60$ en el Planar 3AP, los tiempos de la relajación lineal que establecimos como máximos se superan, y el problema con variables enteras tarda demasiado en terminar su iteración, por lo que no se recogen las cotas o *gaps*. En las siguientes gráficas se puede ver la evolución de los tiempos de resolución en función del tamaño n.

4.9.2. Resolución con algoritmo de subgradiente

A continuación se exponen los resultados obtenidos al aplicar el algoritmo de subgradiente a las instancias del Axial 3AP. Para el modelo Planar 3AP resulta muy complejo encontrar un algoritmo de reparación de solución factible, y la cota superior implementada que corresponde al cuadrado latino explicada en 4.3 no es eficiente. Esto es debido a que encontrar cuadrados latinos no es nada trivial. Por eso la implementación del algoritmo de subgradiente del Planar 3AP en nuestro caso no proporciona resultados interesantes y no se verá.

El algoritmo de subgradiente ha sido aplicado dividiendo la relajación lagrangiana de la forma mostrada en la Sección 3.4. Veamos qué valores recoge la tabla de resultados:

- n: el tamaño de la instancia.
- Z_{UB}: la mejor cota superior obtenida tras aplicar el algoritmo de subgradiente, es decir, la menor cota superior de las obtenidas mediante el algoritmo voraz o mediante el algoritmo de reparación de la solución iterada.
- Z_{LB}: la mejor cota inferior obtenida tras aplicar el algoritmo de subgradiente, es decir, la cota inferior más alta iterada.
- gUR: el gap entre la mejor cota superior Z_{UB} y el valor de la relajación lineal, que es la mejor cota inferior de todas las instancias.
- gLR: el gap entre la cota inferior Z_{LB} y la cota inferior que encuentra la relajación lineal. Se calcula para comprobar cómo de cerca se queda la cota iterada de la cota de la relajación lineal
- Tiempo: el tiempo en segundos que emplea el algoritmo de subgradiente hasta que se cumpla una condición de parada.
- `Max_Iter`: número máximo de iteraciones permitidas.
- Iteraciones: la cantidad de iteraciones realizadas hasta la finalización del algoritmo.
- ϵ_{step}: valor del parámetro.

n	Z_{UB}	Z_{LB}	gUR	gLR	Tiempo	`max_iter`	Iteraciones	ϵ_{step}
5	44.62	44.62	0	0	0.56	500	64	10^{-8}
10	19.29	19.29	0	0	1.38	500	20	10^{-8}
15	126.97	30.5	75.91	0.26	43.7	500	217	10^{-8}
20	83.84	34.46	58.77	0.32	176.86	500	384	10^{-8}
55	170.97	56.82	65.05	4.92	812.8	500	87	10^{-8}
125	218.59	125	-	-	108.56	1	1	10^{-8}

Tabla 4.3: Resultados del algoritmo de subgradiente del Axial 3AP

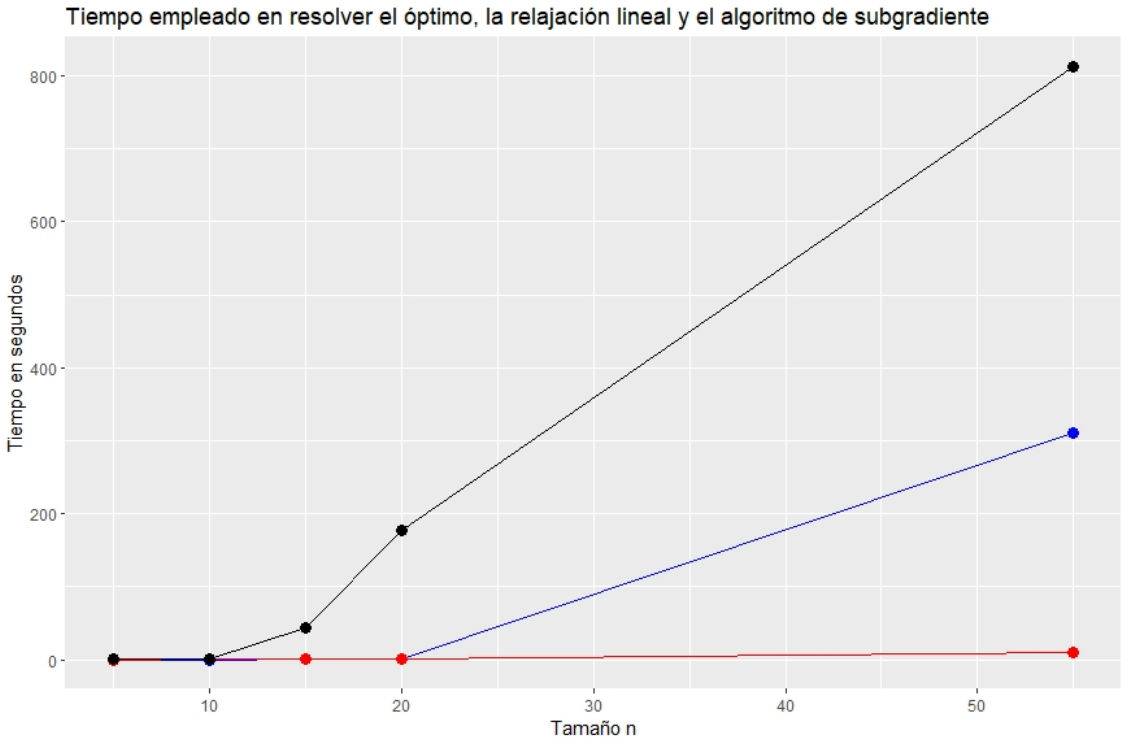


Figura 4.3: Modelo Axial 3AP. En azul, tiempo del óptimo. En rojo, tiempo de la relajación lineal. En negro, tiempo del algoritmo de subgradiente.

Nos fijamos en primer lugar en gUR, el gap entre la mejor cota superior Z_{UB} y el valor de la relajación lineal. Para instancias de menor tamaño, el algoritmo encontró el óptimo, por lo que el gap fue nulo. Con instancias mayores, el gap aumentó a valores grandes, superando el 50 %. Sin embargo, en el caso de la instancia de tamaño $n = 125$ se optó por realizar una sola iteración, dado que el gran tamaño de la instancia provocaba que cada iteración requiriese mucho tiempo. Para esta instancia, la relajación lineal no pudo ser obtenida en un tiempo menor a 120 segundos, pero gracias a la iteración del algoritmo, en 108.56 segundos se pudo obtener una cota inferior. No solo eso, sino que el gap entre la mejor cota superior del algoritmo de reparación y la cota inferior obtenidas es de un 41.89 %, que es un gap mejor (ya que es menor) que los gUR obtenidos en las instancias medianas de tamaño 15, 20 o 55.

En segundo lugar, los valores de gLR arrojaron muy buenos resultados, siendo todos los valores muy bajos. En ninguna instancia se superó el 5 %, lo que indica que la aproximación de la cota inferior iterada a la relajación lineal de cada instancia es muy buena.

Por otro lado, los tiempos de ejecución, al igual que en el caso del algoritmo exacto de Gurobi, también aumentan drásticamente conforme aumenta el tamaño n. En algunos casos, se puede bajar el número máximo de iteraciones para reducir el tiempo de ejecución, como se hizo en el caso de la instancia de tamaño $n = 125$, sin que la calidad de la cota inferior se resienta demasiado. Los tiempos empleados por el algoritmo para cada instancia siempre resultaron superiores a los tiempos empleados por Gurobi en calcular el óptimo y la relajación lineal, como se comprueba en la Figura 4.3, excepto en el caso en el que se limitó el número máximo de iteraciones, que no se recoge en el gráfico.

Una posible forma de mejorar los tiempos de ejecución pasaría por modificar las condiciones de parada, ya sea disminuyendo `Max_Iter`, o aumentando el valor de ϵ_{step}. Esto es debido a que durante la impresión en pantalla de las iteraciones, se observó que cuando el paso o `step` era muy pequeño, la mejor cota inferior variaba muy poco, de modo que ese extra de tiempo que se tardaba compensaba poco en relación a la calidad que ganaba la cota, que ya en iteraciones tempranas era cercana a la de la relajación lineal.
A continuación se presentan los resultados al aumentar el ϵ_{step} del valor 10^{-8} al valor 10^{-4}.

n	Z_{UB}	Z_{LB}	gUR	gLR	Tiempo	`max_iter`	Iteraciones	ϵ_{step}
5	44.62	44.62	0	0	0.48	500	49	10^{-4}
10	19.29	19.29	0	0	1.21	500	20	10^{-4}
15	126.97	30.5	75.91	0.26	20.6	500	103	10^{-4}
20	83.84	34.45	58.77	0.35	74.58	500	164	10^{-4}

Tabla 4.4: Resultados del algoritmo de subgradiente del Axial 3AP

Se puede apreciar que al aumentar el valor de ϵ_{step}, el algoritmo deja de iterar antes, ahorrando mucho tiempo, pero la cotas que proporciona son prácticamente idénticas. Veamos ahora qué ocurre al aumentarlo más, con el valor 10^{-2}.

n	Z_{UB}	Z_{LB}	gUR	gLR	Tiempo	`max_iter`	Iteraciones	ϵ_{step}
5	44.62	44.62	0	0	0.58	500	48	10^{-2}
10	19.29	19.29	0	0	1.25	500	20	10^{-2}
15	61.25	30.35	50.07	0.75	9.22	500	47	10^{-2}
20	83.84	34.27	58.77	0.87	25.06	500	54	10^{-2}

Tabla 4.5: Resultados del algoritmo de subgradiente del Axial 3AP

Se observa que los resultados en las instancias más pequeñas son prácticamente idénticos. Sin embargo, en las instancias mayores, la cota inferior pierde un poco de calidad, aumentando gLR ligeramente. Sin embargo, el tiempo de ejecuación se reduce notablemente, e incluso en el caso de la instancia de $n = 15$, se encontró una solución reparada mejor que la obtenida cuando ϵ_{step} era menor, de forma que se redujo gUR considerablemente de 75.91 % a 50.07 %.

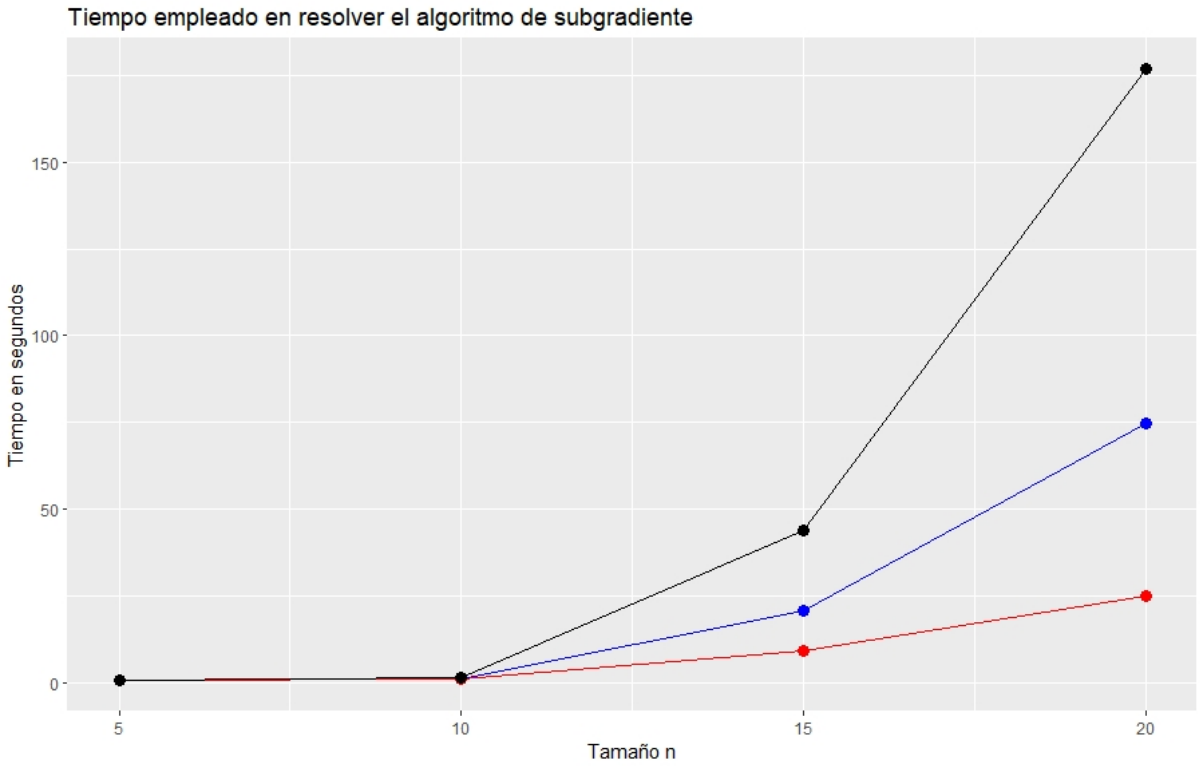


Figura 4.4: Modelo Axial 3AP. En negro, tiempo del algoritmo con $\epsilon_{step} = 10^{-8}$. En azul, tiempo del algoritmo con $\epsilon_{step} = 10^{-4}$. En rojo, tiempo del algoritmo con $\epsilon_{step} = 10^{-2}$.

Todos los resultados hasta el momento fueron obtenidos al dividir la relajación lagrangiana en subproblemas para que el algoritmo fuera más eficiente. Para comparar cómo hubiera sido resolver la relajación directamente, se aplicó el algoritmo a las mismas instancias de esta forma para comparar los tiempos, que se recogen en la Tabla 4.6. Se observa que los tiempos de ejecución son mayores que los reportados en la Tabla 4.3.

n	Z_{UB}	Z_{LB}	gUR	gLR	Tiempo	`max_iter`	Iteraciones	ϵ_{step}
5	44.62	44.62	0	0	0.56	500	49	10^{-8}
10	19.29	19.29	0	0	1.41	500	20	10^{-8}
15	126.97	30.5	75.91	0.26	47.51	500	217	10^{-8}
20	83.84	34.46	58.77	0.32	202.67	500	384	10^{-8}

Tabla 4.6: Resultados del algoritmo de subgradiente del Axial 3AP sin dividir la relajación en subproblemas

Como se observa, manteniendo las mismas instancias y los mismos valores de `max_iter` y de `Scale`, los tiempos en este caso fueron superiores a los obtenidos al dividir la relajación en subproblemas.

Existe otra forma de realizar el algoritmo que puede mejorar los tiempos. Esta manera más eficiente de realizar el algoritmo consiste en resolver la relajación o subproblemas en los que se divide la relajación con variables continuas, es decir, su relajación lineal. Aunque las cotas inferiores serían a priori peores, pueden aproximarse muy bien requiriendo menos tiempo. Los resultados obtenidos se recogen en la Tabla 4.7. Se comprueba que los tiempos de ejecución son menores que los reportados en la Tabla 4.3.

n	Z_{UB}	Z_{LB}	gUR	gLR	Tiempo	`max_iter`	Iteraciones	ϵ_{step}
5	44.62	44.62	0	0	0.55	500	49	10^{-8}
10	19.29	19.29	0	0	1.37	500	20	10^{-8}
15	126.97	30.5	75.91	0.26	41.74	500	217	10^{-8}
20	83.84	34.46	58.77	0.32	169.36	500	384	10^{-8}

Tabla 4.7: Resultados del algoritmo de subgradiente del Axial 3AP al resolver la relajación con variables continuas

En [3] se exponen más resultados computacionales del Problema Axial de interés.

Conclusiones

Este trabajo comenzó con el objetivo de profundizar en la Relajación Lagrangiana e implementar un algoritmo de subgradiente que aproximase la solución de problemas multiperiodo. A lo largo de los diferentes capítulos se ha conseguido en primer lugar estudiar los conceptos teóricos que involucra la Relajación Lagrangiana, y también la teoría y la importancia de los problemas de asignación multiperiodo. En el capítulo final se ha implementado la resolución de modelos con el solver Gurobi, y también se ha programado el algoritmo de subgradiente y se han obtenido diversos resultados en función del modo de aplicación y de la fijación de parámetros.

Gracias a este trabajo mis conocimientos en materia de optimización combinatoria han aumentado, tanto a nivel teórico, entendiendo cómo funciona la Relajación Lagrangiana, el algoritmo de subgradiente, los modelos de asignación multiperiodo, etc; como a nivel práctico: nunca había trabajado con Python y he sido capaz de implementar el algoritmo de subgradiente, algoritmos tipo voraz y algoritmos de reparación de solución factible y la resolución mediante Gurobi de los modelos de asignación multiperiodo, entendiendo también su funcionamiento interno mediante la búsqueda de cotas superiores e inferiores. También me he ayudado de programas como Excel, Geogebra, RStudio, TeXstudio y JabRef.

Índice de figuras

Índice de tablas

Bibliografía

[1] S.E. Bammel and J. Rothstein. The number of 9 x 9 latin squares. *Discr. Math*, 1995.

[2] Mokhtar S. Bazaraa and Hanif D. Sherali. On the choice of step size in subgradient optimization. *European Journal of Operational Research*, 7(4):380–388, 1981.

[3] R. E. Burkard and R. Rudolf. Computational investigations on 3-dimensional axial assignment problems. *JORBEL - Belgian Journal of Operations Research, Statistics, and Computer Science*, 32(1, 2):85?98, Jan. 2020.

[4] R.E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.

[5] S Daskalaki, T Birbas, and E Housos. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*, 153(1):117–135, 2004. Timetabling and Rostering.

[6] S.A. Dichkovskaya and M.K Kravtsov. Investigation of polynomial algorithms for solving the three-index planar assignment problem. 2006.

[7] D. Magos. Tabu search for the planar three-index assignment problem. *J. Glob.Optim.*, 1996.

[8] D. Magos and P. Miliotis. An algorithm for the planar three-indexassignment problem. *European Journal of Operational Research*, 1994.

[9] Brendan D. McKay, Alison Meynert, and Wendy Myrvold. Small latin squares, quasigroups, and loops. *Journal of Combinatorial Designs*, 2007.

[10] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[11] S. Nickel, J. Puerto, and A. Rodriguez-Chia. Location problems with multiple criteria. In Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama, editors, *Location Science*, chapter 9, pages 205–247. Springer International Publishing Switzerland, 2015.

[12] Francisco A. Ortega, Miguel A. Pozo, and Justo Puerto. Modelling and planning public cultural schedules for efficient use of resources. *Computers and Operations Research*, 58:9–23, 2015.

[13] Dilson Lucas Pereira, Julio Cesar Alves, and Mayron Cesar de Oliveira Moreira. A multiperiod workforce scheduling and routing problem with dependent tasks. *Computers and Operations Research*, 118:104930, 2020.

[14] Miguel A. Pozo, Justo Puerto, and Antonio M. Rodríguez-Chía. The ordered median tree of hubs location problem. *TOP*, 29:78–105, 2021.

[15] Robert J. Vanderbei. *Linear ProgrammingFoundations and Extensions*. 2014.

[16] M. Vlach. Branch-and-bound method for the three-index assignment problem. *Ekonomicko-Matematicky Obzor*, 1967.

[17] L.A. Wolsey. *Integer Programming.* Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.