Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO DE FIN DE GRADO

GRADO EN MATEMÁTICAS

El algoritmo *Branch and Cut* en programación lineal entera

Autor: Carlos Noriega González

Tutor: Pedro César Álvarez Esteban

Curso 2022-2023

Índice general

Lista de figuras y tablas

Figuras

Tablas

Resumen

Los problemas de programación lineal entera aparecen en numerosas ocasiones modelando situaciones del mundo real donde las variables de decisión deben tomar valores enteros. A finales del siglo pasado se desarrolló el algoritmo de *Branch and Cut* combinando las ideas de dos métodos ya conocidos a mediados de siglo, como eran el algoritmo de *Branch and Bound* y el método de planos de corte. A lo largo de este trabajo presentaremos los diferentes tipos de programación lineal, introduciremos brevemente los conceptos básicos de la teoría de poliedros y del método símplex; y abordaremos los métodos de *Branch and Bound* y de planos de corte explicando los algoritmos y tratando todas sus particularidades. En el penúltimo capítulo llegaremos al objetivo central de este trabajo: entender cómo funciona el algoritmo de *Branch and Cut* y los diferentes procesos que se pueden llevar a cabo previamente para acelerar la búsqueda de la solución óptima utilizando este método. Finalmente, pondremos en práctica todo lo expuesto en el trabajo planteando un problema denominado *Team Orienteering Problem*. Resolveremos este problema utilizando dos implementaciones diferentes del *Branch and Cut*, comparando los resultados obtenidos con ambos enfoques.

Abstract

Integer linear programming problems appear on numerous occasions modelling real-world situations where the decision variables must take integer values. At the end of the last century, the *Branch and Cut* algorithm was developed combining the ideas of two methods already known in the middle of the century, such as the *Branch and Bound* algorithm and the cutting plane method. Throughout this work we will present the different types of linear programming, we will briefly introduce the basic concepts of the theory of polyhedra and the simplex method; and we will deal with the *Branch and Bound* and cutting plane methods, explaining the algorithms and all their peculiarities. In the penultimate chapter we will reach the central objective of this work: understanding how the *Branch and Cut* algorithm works and the different processes that can be carried out beforehand to speed up the search for the optimal solution using this method. Finally, we will put into practice everything exposed in the work by working with a problem called *Team Orienteering Problem*. We will solve this problem using two different implementations of the *Branch and Cut*, and we will compare the results obtained with both approaches.

Introducción

La programación lineal es una rama de la investigación de operaciones que estudia la optimización de una función lineal sujeta a un conjunto de restricciones, también lineales. A la hora de resolver un problema de programación lineal debemos determinar los valores que toman las variables de decisión involucradas. Esta rama de las matemáticas es de una gran utilidad en diversos campos, ya que nos permite modelar diferentes situaciones del mundo real relacionadas con la economía, la industria o el sector público.

En muchos de los problemas que modelan este tipo de situaciones las variables de decisión que aparecen deben tomar valores enteros, ya que o bien representan cantidades que deben ser enteras; o bien están asociadas a la toma de decisiones Sí-No, donde en este caso hablamos de variables binarias que toman valores 0-1. En este tipo de situaciones se añaden restricciones que consisten en que todas (o alguna) de las variables de decisión tomen valores enteros. La programación entera se encarga de estudiar este tipo de problemas y buscar soluciones óptimas en las que las variables satisfagan estas restricciones *enteras*.

Los métodos utilizados en la programación lineal no resultan útiles para esta clase de problemas, pues las restricciones enteras no se pueden añadir como una restricción lineal equiparable a las restricciones lineales restantes. El *redondeo* a la solución entera más próxima, partiendo de la solución óptima del problema lineal que resulta de eliminar las restricciones enteras, no constituye un método para resolver estos problemas. Esto se debe a que este *redondeo* puede estar muy alejado de la solución óptima con valores enteros que estamos buscando. Es por ello que a lo largo del último siglo se han desarrollado diferentes métodos para resolver estos problemas.

El desarrollo del algoritmo de *Branch and Bound* y del método de planos de corte (*Cutting planes*) a mediados del siglo pasado permitió resolver problemas de programación entera de una manera más eficiente, pero resultaron poco útiles en la práctica a la hora de resolver problemas de elevada complejidad. Debido a esto se intensificó el estudio para conseguir un método eficaz para este tipo de problemas. Gracias a esta investigación nació en los años 90 el algoritmo de *Branch and Cut*, combinando las ideas de estos dos algoritmos. Este método se ha demostrado empíricamente como el mejor algoritmo generalista para este tipo de problemas.

A lo largo de este trabajo presentaremos estos tres algoritmos y todo el contenido teórico necesario para entender cómo y porqué nos permiten obtener la solución óptima a un problema de programación entera. Como conclusión, presentaremos un caso práctico donde se hace uso del algoritmo de *Branch and Cut*.

Capítulo 1

Nociones iniciales

1.1. Programación lineal y entera

El problema general que se presenta en la programación lineal es el siguiente:

$$max\{cx : Ax \leq b, x \in \mathbb{R}^n_+\} \tag{1.1}$$

En esta expresión A es una matriz de tamaño $m \times n$, b un vector columna de tamaño $m \times 1$, c un vector fila de tamaño $1 \times n$ y x un vector columna de tamaño $n \times 1$ que contiene las variables cuyo valor debemos determinar. Este vector x a determinar debe cumplir $x \in \mathbb{R}^n_+$, por lo que todas sus variables tomarán valores mayores o iguales que cero.

El objetivo por tanto de la programación lineal es el de maximizar una función lineal que denominamos función objetivo y representamos mediante el producto cx, sujeto a ciertas restricciones representadas por $Ax \leq b$. Denotaremos por Z_{LP} a este valor máximo de la función objetivo que queremos obtener. Puesto que los valores que toman las variables $(x_1, ..., x_n) = x$ están sujetos a ciertas restricciones, podemos introducir las siguientes definiciones en relación a los conjuntos de puntos que las satisfacen:

Definición 1.1. Una solución factible x^* del problema $max\{cx : Ax \leq b, x \in \mathbb{R}^n_+\}$ es aquella que satisface $Ax^* \leq b$ y la restricción de no negatividad.

Definición 1.2. Una región factible es el conjunto de todas las soluciones factibles de un problema de optimización.

A partir de este problema de programación lineal podemos añadir restricciones a algunas variables (o a todas) para que éstas tomen únicamente valores enteros. De esta forma surgen nuevos tipos de programación lineal:

- Programación lineal entera mixta : un problema de este tipo viene dado por

$$max\{cx + hy : \ Ax + Gy \leq b; \ x \in \mathbb{Z}^n_+, y \in \mathbb{R}^p_+\} \tag{1.2}$$

  En este caso, el vector y de tamaño $p \times 1$ contiene las variables continuas y mayores que 0, puesto que $y \in \mathbb{R}^p_+$, mientras que G es una matriz de tamaño $m \times p$ y h es un vector fila de tamaño $1 \times p$. A, b, c, x denotan la misma matriz y los mismos vectores que en la ecuación (1.1). En este problema debemos maximizar la expresión $cx + hy$, denotando al valor máximo alcanzado por esta expresión Z_{MILP}.

- Programación lineal entera: un problema de este tipo viene dado por

$$max\{cx:\ Ax \leq b;\ x \in \mathbb{Z}_+^n\} \tag{1.3}$$

  Podemos observar que la formulación de este problema es muy similar a la de la ecuación (1.1), con la diferencia que ahora el vector x debe tomar valores enteros y mayores que 0, puesto que $x \in \mathbb{Z}_+^n$. En este caso utilizaremos la notación Z_{IP} para hacer referencia al valor máximo de la función objetivo.

- Programación entera binaria: un problema de este tipo viene dado por

$$max\{cx:\ Ax \leq b;\ x \in \{0,1\}^n\} \tag{1.4}$$

  En la programación entera binaria las variables de decisión $x_1, ..., x_n$ toman valores 0 ó 1, y es por tanto un caso especial de la programación entera. Este tipo de programación suele aparecer en problemas relacionados con la toma de decisiones Sí o No.

En este trabajo nos centraremos en la programación lineal entera, y el algoritmo *Branch and Cut*, traducido en español como *ramificación y corte*. Antes de llegar a ese punto debemos estudiar en primer lugar nociones básicas sobre la teoría de poliedros, así como conceptos como la dualidad o la relajación, que nos serán de gran ayuda para presentar los dos algoritmos sobre los que se ha desarrollado el *Branch and Cut*, como son el algoritmo de *Branch and Bound*(en español, *ramificación y poda*) y el algoritmo de *Cutting planes*, conocido en español como *planos de corte*. Por último, haremos un breve repaso al método simplex, el método por excelencia para la resolución de problemas de programación lineal.

1.2. Teoría de poliedros

En este segundo apartado presentaremos la noción de poliedro y numerosos conceptos relacionados con él que nos seran útiles a lo largo de todo el trabajo.

Definición 1.3. Un poliedro es un subconjunto P de $\mathbb{R}^n$ descrito por un conjunto finito de restricciones lineales $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. Un poliedro es racional si A y b tienen coeficientes racionales.

La descripción de un poliedro se puede llevar a cabo utilizando las restricciones lineales definidas en $Ax \leq b$, pero también haciendo uso de los puntos y rayos extremos del poliedro P, como afirma el *Teorema de Minkowski*. Antes de presentar este teorema definiremos los conceptos utilizados en su enunciado.

Definición 1.4. (*Punto extremo*) Dado un poliedro P, x es un punto extremo de P si no se puede expresar como combinación convexa de dos puntos distintos de P; es decir, $\nexists x^1, x^2 \in P$ tal que $x = \lambda x^1 + (1-\lambda)x^2$ para un $\lambda \in (0,1)$.

Observación 1.5. *El concepto de punto extremo corresponde a la idea geométrica de un vértice del poliedro que estamos considerando.*

Definición 1.6. (*Rayo*) Sea un poliedro $P = \{x : Ax \leq b, x \in \mathbb{R}^n\}$, definimos $P^0 = \{r \in \mathbb{R}^n :\ Ar \leq 0\}$. Entonces $r \in P^0 \setminus \{0\}$ es un rayo de P

Definición 1.7. (*Rayo extremo*) Un rayo r de P es un rayo extremo de P si no se puede expresar como combinación cónica de dos puntos distintos de P; es decir, $\nexists r^1, r^2$ rayos de P y constantes positivas $\lambda, \gamma > 0$ tales que $r = \lambda r^1 + \gamma r^2$.

Observación 1.8. *El conjunto $P^0 = \{r \in \mathbb{R}^n : Ar \leq 0\}$ presentado en la definición de rayo se denomina cono de recesión del poliedro P. Este conjunto también lo podemos definir como el siguiente conjunto:*

$$rec(P) := \{r \in \mathbb{R}^n : x + \lambda r \in P \ \forall x \in P, \ \lambda \in \mathbb{R}_+\}$$

A continuación presentaremos el *Teorema de Minkowski* sin su correspondiente demostración, la cual podemos encontrar en la sección 3.5 del libro [3]. Este teorema nos permite representar el poliedro P como combinación convexa de sus puntos extremos más una combinación cónica de sus rayos extremos, es decir:

$$P = conv(x^1, ..., x^k) + cone(r^1, ..., r^j)$$

Teorema 1.1. *(Teorema de Minkowski) Si $P = \{x : Ax \leq b, x \in \mathbb{R}^n\} \neq \emptyset$ y $rank(A) = n$, entonces*

$$P = \{x \in \mathbb{R}^n : x = \sum_{k \in K} \lambda_k x^k + \sum_{j \in J} \mu_j r^j, \sum_{k \in K} \lambda_k = 1, \lambda_k \geq 0 \, \forall k \in K, \mu_j \geq 0 \, \forall j \in J\}$$

donde $\{x^k\}_{k \in K}$ es el conjunto de puntos extremos de P y $\{r^j\}_{j \in J}$ es el conjunto de rayos extremos de P.

A lo largo del trabajo mencionaremos la noción de *dimensión* de un poliedro $P \subseteq \mathbb{R}^n$, la cual se corresponde con la dimensión del menor espacio afín E tal que el poliedro $P \subseteq E$. A este espacio afín E se le denomina *envolvente afín* del poliedro P y se denota por *aff(P)=E*. Se cumple que *dim(P)=dim(aff(P))=dim(E)*.

Podemos estudiar la envolvente afín de un poliedro P haciendo uso del sistema $Ax \leq b$ que define nuestro poliedro, siendo $a^i x \leq b_i \ \ i \in \{1, ..., m\}$ las desigualdades que aparecen en el sistema.

Definición 1.9. Una desigualdad $a^i x \leq b_i$ es una igualdad implícita de $Ax \leq b$ si $a^i x = b_i$ para toda solución de $Ax \leq b$.

Observación 1.10. *Utilizaremos la notación $A^= x \leq b^=$ para el sistema reducido que incluye todas las igualdades implícitas del sistema $Ax \leq b$.*

El sistema que incluye todas las igualdades implícitas nos permite definir la envolvente afín de P, resultado que vemos en la siguiente proposición:

Proposición 1.2. *Sea $P := \{x \in \mathbb{R}^n : Ax \leq b\}$ un poliedro no vacío. Entonces:*

$$aff(P) = \{x \in \mathbb{R}^n : A^= x = b^=\}$$

Además $dim(P) = n - rank(A^=)$.

Tampoco incluiremos la demostración de esta proposición, pues no juega un papel importante en el contenido de nuestro trabajo, pero la podemos encontrar en la sección 3.7 del libro [3].

1.3. Optimalidad

Dado el problema de programación lineal entera

$$max\{cx : Ax \leq b;\, x \in \mathbb{Z}_+^n\} \qquad (IP)$$

queremos saber cómo es posible probar que un punto x^* proporciona una solución óptima, es decir, probar que el valor máximo Z_{IP} se alcanza en x^* ($Z_{IP} = cx^*$). Esta necesidad de disponer de criterios de optimalidad que determinen si una solución es o no óptima nos lleva a la búsqueda de acotaciones inferiores $\underline{Z} \leq Z_{IP}$ y superiores $\overline{Z} \geq Z_{IP}$ tales que lleguemos a

$$\underline{Z} = Z_{IP} = \overline{Z} \tag{1.5}$$

Observación 1.11. *Por comodidad en la notación, utilizaremos $Z_{IP} = Z$ de aquí en adelante siempre que hablemos del valor máximo de la función objetivo en un problema de programación entera.*

En la práctica los algoritmos actuales trabajan en la búsqueda de una secuencia decreciente de acotaciones superiores:

$$\overline{Z_1} > \overline{Z_2} > ... > \overline{Z_m} \geq Z$$

y una creciente de acotaciones inferiores:

$$\underline{Z_1} < \underline{Z_2} < ... < \underline{Z_n} \leq Z$$

El algoritmo para cuando $\overline{Z_m} - \underline{Z_n} < \epsilon$ para un $\epsilon > 0$ fijado. Por tanto, debemos buscar caminos para obtener estas acotaciones:

- Las acotaciones inferiores se pueden obtener utilizando el valor que alcanza la función objetivo en cualquier solución factible $\hat{x}$ de nuestro problema (IP), ya que este valor va a ser igual o menor que Z, al cual habíamos denotado como el valor máximo que alcanza nuestra función objetivo. Por ello, podemos tomar como acotación inferior $\underline{Z} = c\hat{x} \leq Z$.

- Las acotaciones superiores requieren un diferente estudio. La mejor vía para obtener estas acotaciones es utilizando la *relajación*, es decir, reemplazando el problema inicial de programación entera (IP) por un problema más sencillo cuyo valor óptimo sea mayor o igual que Z. Para conseguir esta *relajación* del problema original se nos presentan dos opciones:

  1. Ampliar el conjunto de soluciones factibles, es decir, relajar las restricciones que se imponen a las variables.
  2. Reemplazar la función objetivo a maximizar (o minimizar) por otra que tome un valor igual o mayor (igual o menor) en todas las soluciones factibles originales.

  Además del estudio de un problema relajado como vía para obtener acotaciones superiores, también podemos estudiar el problema dual de un problema dado.

Tanto la relajación como la dualidad son las dos vías principales para determinar acotaciones superiores de Z, y por tanto estos dos conceptos son los que vamos a estudiar en los dos siguientes apartados.

1.4. Relajación

En este apartado presentaremos la noción general de *relajación* de un problema, centrándonos después en el concepto de relajación cuando trabajamos con un problema de programación entera. En esta primera definición f y c hacen referencia a las funciones objetivo de ambos problemas, en nuestro caso funciones lineares; así como T y X hacen referencia a los poliedros donde debemos maximizar el valor de esas funciones.

Observación 1.12. *Haciendo uso de la noción de poliedro podemos reescribir nuestro problema original*

$$max\{cx : Ax \leq b, x \in \mathbb{R}^n_+\}$$

de la siguiente forma:

$$max\{cx : x \in P \cap \mathbb{R}^n_+\}$$

Definición 1.13. El problema

$$max\{f(x) : x \in T \subseteq \mathbb{R}^n\} \quad (RP)$$

es una relajación del problema original

$$max\{c(x) : x \in P \subseteq \mathbb{R}^n\} \quad (LP)$$

si se cumplen las siguientes condiciones:

$$P \subseteq T \ y \ f(x) \geq c(x) \, \forall x \in P$$

Llevar a cabo una relajación de un problema dado nos da la posibilidad de obtener acotaciones superiores para ese problema original, como podemos observar en la siguiente proposición:

Proposición 1.3. *Si el problema (RP) es una relajación del problema original (LP), entonces $Z_{RP} \geq Z_{LP}$.*

Demostración. Si x^* es una solución óptima para el problema *(LP)*, entonces

$$x^* \in P \subseteq T \quad y \quad Z_{LP} = c(x^*) \leq f(x^*)$$

Puesto que $x^* \in T$, entonces es una solución factible para el problema RP, y por tanto es una acotación inferior de Z_{RP}, es decir:

$$Z_{LP} \leq f(x^*) \leq Z_{RP}$$

□

Proposición 1.4. *i) Si una relajación (RP) de un problema es no factible, el problema original (LP) tampoco lo es.*

ii) Sea x^* una solución óptima para una relajación del problema original. Si $x^* \in X$ y $f(x^*) = c(x^*)$, entonces x^* es una solución óptima del problema original (LP).

Demostración. i) Puesto que la relajación del problema es no factible, entonces $T = \emptyset$, y debido a que $P \subset T$, $P = \emptyset$, por lo que el problema original no es factible.

ii) Puesto que $x^* \in P$, el valor máximo que alcanza la función objetivo en P, denotado por Z_{LP}, cumple la siguiente desigualdad

$$Z_{LP} \geq c(x^*) = f(x^*) = Z_{RP} \tag{1.6}$$

Utilizando ahora la Proposición 1.3 sabemos que $Z_{LP} \leq Z_{RP}$, y por tanto en la expresión (1.6) todo son igualdades, y obtenemos que $c(x^*) = Z_{RP} = Z_{LP}$, por lo que x^* es una solución óptima del problema original. □

Partiendo de un problema de programación entera de la forma estándar

$$max\{cx : \ Ax \leq b;\ x \in \mathbb{Z}_+^n\}$$

vamos a estudiar la *relajación al problema lineal* de este problema. Denotaremos por P al poliedro definido por las restricciones $Ax \leq b$, como ya hicimos anteriormente.

Definición 1.14. Sea el problema de programación entera $max\{cx : x \in P \cap \mathbb{Z}_+^n\}$; la relajación al problema lineal es el siguiente problema

$$max\{cx : x \in P \cap \mathbb{R}_+^n\} \quad (RP)$$

Observación 1.15. *Se cumplen por tanto las condiciones de la Definición 1.13, puesto que* $(P \cap \mathbb{R}_+^n) \cap \mathbb{Z}^n \subseteq P \cap \mathbb{R}_+^n$ *y no hemos modificado la función objetivo. El nuevo problema RP es por tanto una relajación del problema inicial.*

1.5. Dualidad

Asociado a cada problema lineal existe otro problema de programación lineal denominado problema dual (D), que posee importantes propiedades y relaciones con respecto al problema lineal original, denominado como problema primal (P). Con el término *dualidad* hacemos referencia a estas parejas de problemas y a la relación entre sus soluciones. Los dos componentes de la pareja son el problema primal y el problema dual:

- Problema primal (P)

$$max\{cx : Ax \leq b, x \in \mathbb{R}_+^n\} \tag{1.7}$$

  Z representa el valor máximo que alcanza la función objetivo en la región factible.

- Problema dual (D)

$$min\{ub : uA \geq c, u \in \mathbb{R}_+^m\} \tag{1.8}$$

  W representa el valor mínimo que alcanza la función objetivo en la región factible.

Las relaciones entre estos dos problemas son muy numerosas, y podemos destacar algunas de ellas:

a. El problema dual tiene tantas variables de decisión como restricciones tiene el primal, en este caso m.

b. El problema dual tiene tantas restricciones como variables tiene el primal, en este caso n.

c. Los coeficientes de la función objetivo del problema dual son los términos independientes de las restricciones del programa primal, en este caso el vector b.

d. Los términos independientes de las restricciones del dual son los coeficientes de la función objetivo del problema primal, en este caso el vector c.

e. Si el problema primal es un problema de maximización (minimización), el dual es un problema de minimización (maximización).

f. El sentido de las desigualdades de las restricciones del problema dual es el inverso a las desigualdades del primal.

La propiedad más importante de esta pareja de problemas es que el valor de la función objetivo en cualquier solución factible del problema dual nos proporciona una acotación superior de la función objetivo del problema primal.

Teorema 1.5. ***(Dualidad débil).*** *El problema primal (P) y el problema dual (D) satisfacen la desigualdad*

$$cx^* \leq Z \leq W \leq u^*b$$

para toda x^* solución factible del problema primal y toda u^* solución factible del dual.

Demostración. Utilizando la desigualdad en (1.8) $u^*A \geq c$ con u^* solución factible de (D), y multiplicando por $x^* \in \mathbb{R}^n_+$, solución factible de (P), tenemos la desigualdad

$$cx^* \leq u^*Ax^*$$

Utilizando ahora que $Ax^* \leq b$ para todo x^* solución factible de (P), tenemos que

$$cx^* \leq u^*Ax^* \leq u^*b$$

Por tanto hemos llegado a que $cx^* \leq u^*b$ para toda u^* solución factible de (D) y para toda x^* solución factible de (P), lo que implica que

$$W = min\{ub : uA \geq c, u \in \mathbb{R}^m_+\} \geq cx^*$$

para toda solución factible x^* de (P). Por tanto en particular se cumple que

$$W \geq Z = max\{cx : Ax \leq b, x \in \mathbb{R}^n_+\}$$

De esta forma llegamos a la expresión pedida:

$$cx^* \leq Z \leq W \leq u^*b \tag{1.9}$$

□

Observación 1.16. *Utilizaremos el concepto pareja dual débil para referirnos a esta pareja de problemas.*

Observación 1.17. *La ventaja de utilizar un problema dual frente a una relajación es que cualquier solución factible del problema dual proporciona una acotación superior de Z, mientras que en el caso de la relajación de un problema debemos obtener la solución óptima para obtener una acotación superior, como vimos en la Proposición 1.3 ($Z_{RP} \geq Z_{LP}$).*

Veamos ahora que la relajación de un problema de programación entera a uno de programación lineal nos conduce también a una pareja dual débil, esto es:

Proposición 1.6. *El problema de programación entera*

$$max\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\} \quad (IP) \tag{1.10}$$

y el problema dual de la relajación al problema lineal de (IP)

$$min\{ub : uA \geq c, u \in \mathbb{R}_+^m\} \quad (LD) \tag{1.11}$$

satisfacen la desigualdad

$$cx \leq Z \leq W_{LD} \leq ub \tag{1.12}$$

para todo x solución factible de (IP) y u solución factible de (LD).

Demostración: debemos probar la desigualdad (1.12), donde Z hace referencia al valor máximo de cx alcanzado en (IP) y W_{LD} al valor mínimo alcanzado por ub en (LD). En primer lugar sabemos que el problema

$$max\{cx : Ax \leq b, x \in \mathbb{R}_+^n\} \quad (LP)$$

es la relajación al problema lineal de (IP), y por tanto se cumple que $Z \leq Z_{LP}$. Utilizando que $Z_{LP} \leq W_{LD}$ puesto que LD es el problema dual de (LP), llegamos a

$$cx \leq Z \leq Z_{LP} \leq W_{LD} \leq ub$$

para todo x solución factible de (IP) y u solución factible de (LD), por lo que hemos probado la desigualdad pedida.

□

Análogamente a lo visto en el caso de la relajación (Proposición 1.4), la dualidad también nos permite probar la optimalidad de una solución.

Proposición 1.7. *Sea (IP) un problema de programación entera y (LD) el problema dual asociado a la relajación lineal de (IP).*

1. *Si (LD) no está acotado, entonces (IP) es un problema no factible.*
2. *Si $x^* \in X$ y $u^* \in U$ satisfacen que $cx^* = u^*b$, entonces x^* es una solución óptima del problema (IP) y u^* es una solución óptima de (LD).*

Demostración. i) Si el problema (LD) no está acotado sabemos que

$$W_{LD} = min\{ub : uA \geq c, u \in \mathbb{R}_+^m\} \to -\infty$$

Utilizando la desigualdad (1.12) de la Proposición 1.6 sabemos que $Z \leq W_{LD}$, y por tanto $Z \to -\infty$, lo que implica que el problema (IP) es un problema no factible.

ii) Si en la desigualdad (1.12) sustituimos x y u por los correspondientes $x^* \in X$ y $u^* \in U$ tales que $cx^* = u^*b$ llegamos a:

$$cx^* = Z = W_{LD} = u^*b$$

Es decir, las desigualdades se convierten en igualdades y vemos que x^* es solución optima de (IP) puesto que $cx^* = Z$ y u^* es solución óptima de (LD) puesto que $W_{LD} = u^*b$.

□

Estos resultados vistos sobre la dualidad y la relajación nos serán de gran ayuda a la hora de aplicar el método de *Branch and Bound* pues nos permitirán comparar soluciones de subproblemas generados a partir del problema inicial y descartar algunos de ellos.

1.6. Método simplex

El método simplex es el procedimiento más utilizado para resolver problemas de programación lineal. Aunque es un procedimiento algebraico, sus conceptos fundamentales tienen un carácter geométrico, por lo que en primer lugar definiremos algún término relacionado con este carácter geométrico del método simplex.

Definición 1.18. (*Frontera de restricción)* La frontera de restricción es el hiperplano que satisface la ecuación obtenida al sustituir el signo $\leq$ (o $\geq$), de la desigualdad escogida entre todas las que definen el poliedro, por un signo $=$.

Definición 1.19. La frontera de la región factible consiste en todas aquellas soluciones factibles que satisfacen una o más de las ecuaciones que definen las fronteras de restricción.

Observación 1.20. *En el caso de un problema de programación lineal con n variables de decisión, cada una de sus soluciones en los vértices se encuentra en la intersección de n fronteras de restricción.*

Hablaremos de *solución factible en un vértice* cuando una solución del problema que estamos considerando se encuentra en un vértice de la región factible. Utilizaremos la notación FEV para referirnos a este tipo de soluciones, las más importantes dentro del método simplex.

Definición 1.21. (*Solución adyacente*) Dos soluciones FEV son adyacentes entre sí cuando comparten $n-1$ fronteras de restricción. Ambas soluciones están conectadas por un segmento de recta que se encuentra en estas fronteras de restricción compartidas, el cual recibe el nombre de arista de la región factible.

A continuación presentaremos una proposición que hace referencia a la localización de la solución óptima de un problema en un vértice de la región factible.

Proposición 1.8. *Si el problema tiene sólo una solución óptima, ésta debe ser una solución factible en un vértice.*

Demostración. Suponemos que la solución óptima x^* no es una solución FEV por lo que debe existir otras dos soluciones factibles tales que el segmento de recta que las une contiene a x^*. Sean x^1 y x^2 las otras dos soluciones factibles y Z^1 y Z^2 los valores respectivos de la función objetivo. Tomando el segmento de recta que los une, sabemos que $\exists \alpha \in \{0,1\}$ tal que

$$x^* = \alpha x^1 + (1-\alpha)x^2$$

Por tanto esta relación se mantiene con el valor de la función objetivo pues es una función lineal, y por tanto

$$Z^* = \alpha Z^1 + (1-\alpha)Z^2$$

Utilizando que $\alpha + (1-\alpha) = 1$, al comparar Z^*, Z^1 y Z^2 llegamos a tres posibilidades:

- $Z^* = Z^1 = Z^2$ implica que x^1 y x^2 también son óptimas, lo que contradice la unicidad de la solución óptima.
- $Z^1 < Z^* < Z^2$ (o $Z^2 < Z^* < Z^1$) contradicen que x^* sea óptima.

En ambos casos llegamos a un absurdo, y por tanto si existe una solución óptima es una solución FEV. □

El método simplex es un método iterativo que toma una solución FEV, comprueba si es óptima, y si no lo es realiza una nueva iteración para encontrar una mejor solución FEV. La prueba de optimalidad consiste en que si una solución factible en un vértice x^* no tiene soluciones FEV adyacentes en las cuales el valor que toma la función objetivo es mayor que el valor alcanzado en x^*, entonces x^* debe ser una solución óptima del problema de programación lineal.

Siempre que es posible, en el paso inicial del método símplex se elige el origen (todas las variables de decisión iguales a cero) como la solución FEV inicial. Cada vez que el método símplex realiza una iteración para moverse de la solución factible en el vértice actual a una mejor, siempre escoge una solución FEV adyacente. De esta manera la trayectoria que sigue hasta obtener una solución óptima se realiza a lo largo de las aristas de la región factible.

El procedimiento algebraico detrás del método se basa en la resolución de sistemas de ecuaciones. El primer paso para obtener estos sistemas de ecuaciones es convertir las restricciones que son desigualdades en restricciones de igualdad equivalentes, y esto se consigue introduciendo variables de holgura en estas restricciones. De esta manera el problema original se puede sustituir por un problema equivalente *aumentado* con las nuevas variables introducidas.

Definición 1.22. Una solución aumentada es una solución de las variables originales (las variables de decisión) aumentada con los valores correspondientes de las variables de holgura.

Definición 1.23. Una solución básica es una solución en un vértice aumentada.

Definición 1.24. Una solución básica factible (BF) es una solución FEV aumentada.

La diferencia entre el número de variables que aparecen en el sistema y el número de ecuaciones nos proporciona los grados de libertad (d) a la hora de resolver el sistema. Por tanto, podemos elegir d variables cualesquiera e igualarlas a cualquier valor arbitrario para resolver el sistema de ecuaciones en términos de las variables restantes. El método símplex utiliza el cero para este valor arbitrario. Las d variables que se igualan a cero se denominan *variables no básicas* y las otras variables se denominan *variables básicas*. El número de variables básicas es igual al número de restricciones funcionales.

Los valores de las variables básicas se obtienen como la solución del sistema de ecuaciones en el cual las variables no básicas toman el valor 0. A este conjunto de variables básicas se le conoce como la *base*. La solución de este sistema se denomina *solución básica*. Si además las variables básicas satisfacen las restricciones de no negatividad, la solución básica es una solución BF. Dos soluciones básicas factibles son adyacentes si todas menos una de sus variables no básicas son las mismas, lo que implica que todas menos una de sus variables básicas son también las mismas.

Capítulo 2

Método *Branch and Bound*

2.1. Presentación del método

El método de *Branch and Bound*, conocido en español como el método de *ramificación y poda* (o de ramificación y acotación), se aplica para resolver problemas de optimización. Este método fue propuesto por primera vez por la matemática inglesa Ailsa Land y la australiana Alison Doig mientras realizaban una investigación en la *London School of Economics* en 1960. El nombre *Branch and Bound* apareció por primera vez en el trabajo del profesor estadounidense John Little "*An algorithm for the traveling salesman problem*", del año 1963.

La idea principal de este método es, dado un problema original de la forma

$$max\{cx : x \in S\} \ ; \ S = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$$

dividir este problema en una serie de problemas más pequeños, resolver estos problemas y juntar esta información para resolver el problema original. La forma usual de representar esta división del problema original es utilizando un árbol de soluciones. La característica de esta técnica es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no son óptimas, para «podar» esa rama del árbol y no continuar malgastando recursos y procesos en casos que se alejan de la solución óptima. Para «podar» estas ramificaciones se usan distintos criterios de finalización.

La operación de *Branch* consiste en la ramificación de las posibles soluciones. En primer lugar, podemos dividir el conjunto S en conjuntos más pequeños $S^1, S^2, .., S^k$ tales que $S = S^1 \cup S^2 \cup ... \cup S^k$, y denotar $Z^k = max\{cx : x \in S^k\}$ como el valor máximo obtenido en cada subconjunto S^i.

Observación 2.1. *Utilizando esta división de la región admisible S en otras más pequeñas $S^1, ..., S^k$ que cubren por completo la región S podemos probar que $Z = max_{1 \leq i \leq k} Z^i$:*

$$Z = max\{cx : x \in S\} = max\{cx : x \in \cup_{i=1}^k S^i\} =$$

$$= max_{1 \leq i \leq k}\{max\{cx : x \in S^i\}\} = max_{1 \leq i \leq k} Z^i$$

Por tanto, una forma teórica pero nada práctica de obtener la solución óptima en un problema de programación entera donde la región factible está acotada es dividir esta región factible inicial en cada uno de estos subconjuntos, y hacer esto recursivamente, de tal forma que dividamos en cada iteración las regiones admisibles en regiones cada vez más pequeñas. Si esto lo desarrollamos hasta el final enumeraremos todas las posibles soluciones admisibles del problema original, ya que existe un número finito de ellas. Esta división iterativa del conjunto de soluciones en subconjuntos más pequeños se puede representar en un árbol de soluciones admisibles.

Observación 2.2. *Denotaremos nuestro problema de optimización con P^0, su conjunto de posibles soluciones $S^0 = S$ y S^i al conjunto de soluciones asociado a cada nodo i. La operación de subdividir un nodo S^i da lugar a nodos hijos, de tal forma que se debe garantizar que*

$$S^i = \cup_{j \in J_i} S^j$$

donde J_i es el conjunto de subíndices j tales que S^j es un nodo hijo de S_i. Con esta unión hacemos referencia a que las soluciones presentes en el nodo padre deben estar presentes en al menos uno de sus nodos hijo, es decir, a la hora del desarrollo del árbol debemos tener en cuenta que no se pueden perder soluciones.

Una condición que se puede pedir a los subespacios en los que dividimos un subespacio dado es que sean disjuntos, es decir, que $S^j \cap S^k = \emptyset, \forall j, k \in J_i$. Esta operación de subdivisión sucesiva acabará cuando cada nodo contenga una sola solución, es decir, tal que $\mid S^f \mid = 1$. Esta fase de construcción de nodos hijos utilizando particiones de un nodo padre se llama *Branch* (*ramificación*), pues se ha ramificado el problema inicial en subproblemas.

Ejemplo 2.1. *Sea un problema de optimización combinatoria (conjunto de soluciones posibles discreto) con n variables binarias $x_i \in \{0, 1\}$ $i = 1, ..., n$. En este caso, dado un problema P y el correspondiente conjunto de soluciones admisibles E, podemos obtener dos sub-problemas y por tanto dos subconjuntos de E, fijando una de las variables binarias x_i con valor 0 y la otra con valor 1.*

Utilizando esta regla de Branch binario, en el cual cada nodo está subdividido en dos nodos hijo, obtenemos un árbol de soluciones de $n+1$ niveles, en el cual el nivel h contiene todas las soluciones admisibles con las variables $x_1, ..., x_h$ fijadas. Por tanto en el nivel $n+1$ aparecen todas las posibles soluciones, siendo el cardinal de las posibles soluciones 2^n y el número de niveles $n+1$, incluyendo el nivel 0 donde aparece la raíz. En la Figura 1 aparece un árbol de este tipo con $n = 3$.

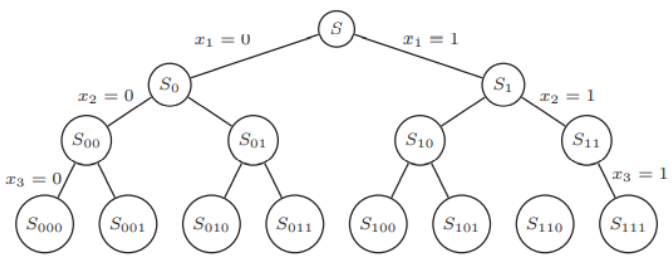

Figura 2.1: Árbol de soluciones aplicando un *Branch* binario con 3 variables

Sin embargo, una enumeración completa de todas las soluciones es imposible para la mayoría de problemas pues el número de nodos obtenidos con la operación de *Branch* crece exponencialmente. Es por esto que el desarrollo completo de un árbol de soluciones de tal forma que obtengamos todas las soluciones no es un método resolutivo, y por ello debemos buscar un método que nos permite explorar solo alguna de las áreas de la región admisible. Para esto vamos a hacer uso de los Z^k, que recordamos que representaban $Z^k = max\{cx : x \in S^k\}$, siendo S^k cada uno de los subconjuntos donde hemos ido dividiendo la región admisible inicial. Las acotaciones inferiores y superiores de estos valores Z^k nos van a permitir descartar alguna de las posibles soluciones que vayan apareciendo en el desarrollo del árbol.

Proposición 2.1. *Sea $S = S^1 \cup S^2 \cup ... \cup S^n$ una descomposición de S en subconjuntos más pequeños, y sea $Z^k = max\{cx : x \in S^k\}$ para k=1,...,n, con $\overline{Z}^k$ una acotación superior de Z^k y $\underline{Z}^k$ una acotación inferior de Z^k. Entonces $\overline{Z} = max_k \overline{Z}^k$ es una acotación superior de Z y $\underline{Z} = max_k \underline{Z}^k$ es una acotación inferior de Z.*

Demostración. Sabemos que $Z = max_{1 \leq k \leq n} Z^k$, por tanto tomamos k^* tal que $Z = Z^{k^*}$. Sea ahora $\overline{Z}^{k^*}$ la acotación superior de Z^{k^*}, sabemos que $Z = Z^{k^*} \leq \overline{Z}^{k^*}$, por lo que también es una acotación superior de Z. Tomando ahora $max_k \overline{Z}^k$ existen dos opciones:

1. El máximo se alcanza en $\overline{Z}^{k^*}$, la cual ya hemos visto que es una acotación superior.
2. El máximo se alcanza en $\overline{Z}^j$ con $k^* \neq j$. En este caso tenemos $Z \leq \overline{Z}^{k^*} \leq \overline{Z}^j$, por lo que $\overline{Z}^j$ también es una acotación superior de Z.

En ambos casos hemos visto que $\overline{Z} = \text{máx}_k \overline{Z}^k$ representa una acotación superior de Z. El segundo caso $\underline{Z} = max_k \underline{Z}^k$ se razona de una manera muy parecida, teniendo en cuenta que si el máximo de las acotaciones inferiores no se alcanza en la acotación inferior $\underline{Z}^{k^*}$, entonces existe $\underline{Z}^j$ donde se alcanza el máximo tal que:

$$\underline{Z}^{k^*} \leq \underline{Z}^j \leq Z^j \leq Z^{k^*} = Z$$

Por tanto, $\underline{Z} = max_k \underline{Z}^k = \underline{Z}^j$ es una acotación inferior de Z. □

Estas acotaciones inferiores y superiores las podemos obtener de la misma manera que ya hemos visto: las acotaciones inferiores vienen dadas por los valores de la función objetivo en las soluciones factibles y las acotaciones superiores haciendo uso de la relajación o la dualidad. Escribiremos $\overline{Z}^k = -\infty$ cuando $S_k = \emptyset$. Estudiaremos ahora en qué casos no es necesario dividir el subconjunto en S^i, y por tanto podemos *podar* esta rama.

Proposición 2.2. *(Criterios de poda) El árbol de soluciones puede ser podado en el nodo correspondiente al subconjunto S^i si alguna de estas tres condiciones se cumple:*

1. *Optimalidad: hemos encontrado una solución óptima entera al problema*

$$max\{\, cx : x \in S^i\} \quad (P^i)$$

.

2. *Acotación: $\overline{Z}^i \leq \underline{Z}$, lo que implica que $Z^i \leq Z$.*
3. *Solución no factible: $S^i = \emptyset$.*

Cada uno de estos subproblemas P^i ($max\{\, cx : x \in S^i\}$) son problemas de programación entera, y por tanto podemos aplicar las técnicas de relajación o dualidad ya vistas. Denotaremos por RP^i una relajación de P^i, aunque en la práctica siempre utilizaremos la relajación al problema lineal del problema P^i vista en 1.14:

$$(P^i) \quad Z^i = max\,\{\, cx : x \in S^i\}$$

$$(RP^i) \quad Z^i_R = max\,\{\, f(x) : x \in S^i_R\}$$

Esta relajación cumple que $S^i \subset S^i_R$ y $f(x) \geq cx$ para todo $x \in S^i$. Entonces podemos reformular la proposición anterior en los siguientes términos:

Proposición 2.3. *El árbol de soluciones puede ser podado en el nodo correspondiente al subconjunto S^i si alguna de estas tres condiciones se cumple.*

1. *Optimalidad: Una solución óptima x^* de RP^i satisface que: $x^* \in S^i$ y $Z^i_R = cx^*$.*
2. *Acotación: $Z^i_R \leq \underline{Z}$, donde $\underline{Z}$ es el valor de alguna solución factible del problema de programación entera.*
3. *Solución no factible: RP^i es un problema no factible.*

Demostración:

1. Sabemos que x^* es una solución óptima del problema relajado (RP^i), es decir $f(x^*) = Z^i_R$. Puesto que $x^* \in S^i$ y $Z^i_R = cx^*$, se cumplen las condiciones del segundo apartado de la Proposición 1.4, y por tanto podemos afirmar que x^* es también es una solución óptima del problema original de programación entera P^i, y por tanto podemos *"podar"* el nodo por optimalidad: el problema P^i ha sido resuelto.
2. La solución óptima del problema original P^i en el nodo i satisface la desigualdad $Z^i \leq Z^i_R$ (Proposición 1.3), por lo que Z^i_R es una acotación superior de Z^i. Utilizando la desigualdad del enunciado sabemos que $\overline{Z}^i = Z^i_R \leq \underline{Z}$, lo que implica que en particular $\overline{Z}^i \leq \underline{Z}$, y podemos podar el nodo por acotación.
3. Por último, utilizando el primer apartado de la Proposición 1.4 vemos que si una relajación del problema original no es factible, el problema original tampoco lo es, y por tanto podemos *"podar"* el nodo utilizando el tercer criterio.

□

Al igual que podíamos resolver los problemas de programación entera utilizando la técnica de la relajación, también podemos utilizar la dualidad. Para ello denotamos en primer lugar DP^i como un problema dual débil de P^i, el problema de programación entera asociado al nodo i. La proposición ahora la podemos formular de la siguiente manera:

Proposición 2.4. *El árbol de soluciones puede ser podado en el nodo correspondiente al subconjunto S^i si alguna de estas dos condiciones se cumple:*

1. *La función objetivo del problema DP^i no está acotada inferiormente.*
2. *DP^i tiene una solución factible en u^* de valor igual o menor que $\underline{Z}$*

Demostración:

1. Utilizando el primer apartado de la Proposición 1.7 sabemos que si DP^i no está acotado inferiormente, el problema original de programación lineal P^i es un problema no factible, y por tanto $S^i = \emptyset$.

2. Utilizando que DP^i es un problema dual débil de P^i, sabemos que para cualquier solución factible del problema dual, el valor de esta solución es mayor o igual que la solución óptima del problema original P^i. Por tanto, si tomamos u^* tenemos

$$Z^i \leq u^* b \leq \underline{Z} \leq Z$$

   Llegamos a la desigualdad $Z^i \leq Z$, la cual ya hemos visto que nos permite *podar* el nodo por *acotación*.

□

Observación 2.3. *Comparando ambos métodos (la relajación y la dualidad) podemos ver que el problema relajado RP^i debe ser resuelto por optimalidad para poder aplicar alguna de las condiciones que nos permiten descartar nodos, mientras que utilizando el problema dual podemos descartar nodos utilizando soluciones factibles aunque no sean óptimas.*

2.2. Algoritmo del método

En esta sección viene presentado un algoritmo general de *Branch and Bound* para resolver problemas de programación entera. Partimos del problema:

$$max\{cx : x \in S\} \ ; \ S = \{x \in \mathbb{Z}_+^n : Ax \leq b\} \quad (P)$$

Sus características más importantes son las siguientes:

- Durante el algoritmo aplicaremos a P y a cada uno de sus subproblemas P^i la relajación lineal consistente en eliminar todas las restricciones enteras y resolver el problema de programación lineal asociado.
- En la descripción del algoritmo, $\mathcal{L}$ almacena esta colección de problemas de programación entera $\{P^i\}$, cada uno de los cuales es de la forma $max\{cx : x \in S^i\}$, siendo Z^i el valor objetivo a maximizar, y con $P^0 = P$ y $S^0 = S$.
- Asociado a cada problema P^i de $\mathcal{L}$ tenemos una acotación superior $\overline{Z}^i \geq Z^i$, las cuales se almacenan para poder aplicar el criterio de poda por *acotación*.
- En el paso 2 del método hablaremos de *métodos heurísticos* para buscar una primera solución factible de nuestro problema original.

La definición de este tipo de métodos es la siguiente:

Definición 2.4. Un método heurístico es un procedimiento para resolver un problema complejo de optimización mediante una aproximación intuitiva, en la que se utiliza la estructura del problema para obtener una solución de manera eficiente.

El uso de estos métodos nos permite obtener soluciones factibles de una manera rápida, pero la obtención de una solución no factible no implica que esta solución sea cercana a una solución óptima del problema, por lo que este paso heurístico está diseñado para encontrar una solución de partida. Profundizaremos en este tipo de métodos en el capítulo 4, cuando hablemos del algoritmo de *Branch and Cut*.

El algoritmo es el siguiente:

- Paso 1 *(Inicialización)*: $\mathcal{L} = \{P\}$. La región $S^0 = S$, es decir, la región de soluciones factibles en el nodo 0 es la original, la cota inferior con la que comenzamos es $\underline{Z} = -\infty$ y la cota superior es $\overline{Z} = +\infty$.

- Paso 2 *(Métodos heurísticos)*: utilizando métodos heurísticos obtenemos una primera solución x^H tal que la función objetivo toma el valor $z^H = cx^H$, el cual podemos tomar como una primera acotación inferior $\underline{Z} = z^H$. La solución óptima la almacenamos por ahora como $x^* = x^H$.

- Paso 3 *(Test de finalización)*: si $\mathcal{L} = \{\emptyset\}$, es decir, si no hay ningún nodo activo, entonces la solución x^* que cumple la igualdad $\underline{Z} = cx^*$ es óptima y el algoritmo se da por finalizado.

- Paso 4 *(Elección del nodo y relajación lineal del problema)*: si hay algún nodo activo, elegimos uno de ellos y resolvemos la relajación lineal del problema seleccionado P^i. De esta manera obtenemos Z_R^i, que es el valor de la función objetivo del problema relajado; y x_R^i, la solución obtenida mediante la relajación lineal.

- Paso 5 *(Poda)*:

  1. Si el problema P^i no es factible, podemos descartar este nodo y volver al paso 3.
  2. Si el valor de $Z_R^i \leq \underline{Z}$, entonces podemos descartar este nodo por acotación y volver al paso 3.
  3. Si la solución $x_R^i \in S^i$, esto es, si tiene todas las variables enteras, entonces esta solución x_R^i ocupa el lugar de $x^* = x_R^i$, y modificamos la acotación inferior: $\underline{Z} = Z_R^i$. Utilizando esta nueva acotación inferior eliminamos los problemas P^i de $\mathcal{L}$ tales que $\overline{Z}^i \leq \underline{Z}$. Descartamos este nodo por optimalidad y volvemos al paso 3.

- Paso 6 *(División)*: si ninguno de los criterios se han cumplido, generamos dos subproblemas P^{i_1} y P^{i_2} con región factible S^{i_1} y S^{i_2} respectivamente tales que $S^{i_1} \cup S^{i_2} = S^i$. A estos problemas les asignamos la cota superior $\overline{Z}^{i_1} = \overline{Z}^{i_2} = Z_R^i$. Añadimos estos dos problemas al conjunto $\mathcal{L}$, y volvemos al paso 3.

El método de *Branch and Bound* es por tanto un procedimiento iterativo de búsqueda en un árbol de soluciones que utiliza herramientas como la relajación lineal de los subproblemas generados y los criterios vistos en la proposición 2.3 para acotar esta búsqueda.

Revisando el algoritmo podemos ver que en cada iteración debemos resolver un problema de programación lineal que se obtiene eliminando las restricciones de integridad de las variables de decisión (paso 4) y utilizar los distintos criterios de finalización para *podar* las ramificaciones (paso 5). En el tercer punto de este quinto paso se indica que si la

solución óptima del problema de programación lineal P^i es entera, entonces es una solución factible del problema original P. Sin embargo, si la solución óptima de este problema P^i no es entera, se debe ramificar en una de las variables que no sea entera para obtener los problemas de programación lineal P^{i_1} y P^{i_2}. El algoritmo continua hasta que todos los subproblemas generados están resueltos, es decir, hasta que $\mathcal{L} = \{\emptyset\}$.

Una vez que el algoritmo está presentado, debemos dar respuesta a numerosas preguntas que pueden aparecer a la hora de intentar entender correctamente este algoritmo. Por ello vamos a tratar las siguientes cuestiones:

1. ¿Qué criterio utilizamos para elegir uno de los nodos activos en la lista de problemas $\mathcal{L}$ en el paso 4 del algoritmo?

2. ¿Cómo efectuamos la división del problema P^i en dos subproblemas P^{i_1} y P^{i_2} en el paso 6 del algoritmo?

3. Al efectuar esta división, ¿qué criterio seguimos para elegir la variable con la cuál vamos a efectuar la división?

En los siguientes apartados daremos respuesta a estas preguntas estudiando varias de las estrategias utilizadas en el algoritmo de *Branch and Bound* para tomar las decisiones acerca de la división en subproblemas, la elección de la variable a la hora de realizar la división o la elección del nodo.

2.2.1. División del problema P^i

Partiendo de nuestro problema original:

$$max\{cx : x \in S\};\ S = \{x \in \mathbb{Z}_+^n : Ax \leq b\} \quad (P) \tag{2.1}$$

Efectuaremos la división de este problema añadiendo restricciones lineales. Para ello, partiendo del conjunto S de soluciones iniciales, el primer paso consiste en dividir este conjunto en dos subconjuntos S_1 y S_2 tales que $S = S_1 \cup S_2$, los cuales están definidos de la siguiente forma:

$$S_1 = S \cap \{x \in \mathbb{R}_+^n\,;\, dx \leq d_0\} \ \ ; \ \ S_2 = S \cap \{x \in \mathbb{R}_+^n\,;\, dx \geq d_0 + 1\}$$

En esta formulación de los conjuntos S_1 y S_2 utilizamos $d \in \mathbb{Z}^n$ y $d_0 \in \mathbb{Z}$. La elección de ambos se puede hacer considerando que satisfagan la siguiente desigualdad:

$$d_0 < dx^0 < d_0 + 1 \tag{2.2}$$

siendo x^0 la solución óptima del problema (LP) obtenido mediante la relajación al problema lineal de nuestro problema original (P), y formulado de la siguiente forma:

$$max\{cx : x \in S_{LP}\};\ S_{LP} = \{x \in \mathbb{R}_+^n : Ax \leq b\} \quad (LP) \tag{2.3}$$

con solución óptima $Z_{LP}^0 = cx^0$. Eligiendo d y d_0 de esta forma obtenemos los subconjuntos

$$S_{1,LP} = S_{LP} \cap \{x \in \mathbb{R}_+^n\,;\, dx \leq d_0\} \ \ ; \ \ S_{2,LP} = S_{LP} \cap \{x \in \mathbb{R}_+^n\,;\, dx \geq d_0 + 1\}$$

con la propiedad de que $x^0 \notin S_{1,LP} \cup S_{2,LP}$ debido a la desigualdad 2.2, y por tanto podemos obtener soluciones a los nuevos problemas LP^i con $i = 1, 2$ tales que:

$$Z^i_{LP} = max\{cx : x \in S^i_{LP}\} < Z^0_{LP}$$

Esta desigualdad se cumple puesto que x^0, donde se alcanza la solución óptima Z^0_{LP}, no pertenece a ninguno de los dos subconjuntos $S_{i,LP}$ con $i \in \{1, 2\}$. De esta forma podemos obtener una acotación superior del problema original más *ajustada* al valor real de ésta.

Para la elección de (d, d_0) podemos utilizar $d = e_j$ para un $j \in \{1, ..., n\}$. Con esta elección obtenemos que la solución x^0 no será factible en los dos nuevos problemas relajados si se cumple que $x^0_j \notin \mathbb{Z}^1$ y $d_0 = \lfloor x^0_j \rfloor$, puesto que en este caso

$$d_0 = \lfloor x^0_j \rfloor < x^0_j < \lfloor x^0_j \rfloor + 1$$

Una ventaja de efectuar esta división es que añadimos restricciones sencillas a nuestro problema original. Esta manera de elegir el valor de (d, d_0) se le conoce como *variable dicotómica*. Aunque existen otros métodos de división de un problema dado, nos centraremos en este a la hora de presentar el siguiente teorema, que hace referencia al tamaño del árbol de soluciones resultante de efectuar las sucesivas divisiones:

Proposición 2.5. *Si $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ está acotado, entonces el árbol de soluciones desarrollado con las variables dicotómicas será finito siempre que en cada nodo i que requiere una división, se elija una dicotomía de la forma*

$$x_j \leq \lfloor x^i_j \rfloor \ ; \ x_j \geq \lfloor x^i_j \rfloor + 1$$

siendo x^i_j la componente j-ésima de la solución al problema lineal LP^i y tal que x^i_j no sea entero. En particular, si $\omega_j = \lceil max\{x_j : x \in P\} \rceil$, entonces ninguna rama del árbol puede contener más de $\sum_{j=1}^n \omega_j$ aristas.

Demostración. Una vez que hemos añadido la restricción $x_j \leq d$ para algún $d \in \{0, ...\omega_j - 1\}$, las únicas otras restricciones relacionadas con la variable x_j que pueden aparecer en una *rama* de nuestro árbol de soluciones que vaya desde la raíz hasta una *hoja* del árbol son las siguientes:

$$x_j \leq d' \ , \ d' \in \{0, ..., d-1\} \ y \ x_j \geq \overline{d} \ , \ \overline{d} \in \{1, ..., d\}$$

Por tanto podemos ver de esta forma que el mayor número de restricciones relacionadas con la variable x_j aparecerán en los tres posibles casos siguientes:

1. Añadiendo las restricciones $x_j \leq d$ para todo $d \in \{0, ..., \omega_j - 1\}$
2. Añadiendo las restricciones $x_j \geq d$ para todo $d \in \{1, ..., \omega_j\}$
3. Añadiendo las restricciones $x_j \geq d$ para todo $d \in \{1, ..., \alpha\}$ y $x_j \leq d$ para todo $d \in \{\alpha, ..., \omega_j - 1\}$, con $\alpha \in \{1, ..., \omega_j - 1\}$.

En cualquiera de los tres casos aparecen ω_j restricciones en x_j y por tanto $\sum_{j=1}^n \omega_j$ en total en cualquier rama. □

Gracias a esta proposición podemos ver que el árbol de soluciones es finito cuando el poliedro P es acotado. Sin embargo, su tamaño depende en gran medida de las acotaciones obtenidas mediante las relajaciones lineales, puesto que una buena acotación inferior del problema (P) cercana a la solución óptima nos permitirá utilizar el criterio de acotación para podar diversos nodos. Por ello, estas acotaciones constituyen el factor más importante en cuanto a la eficiencia de un algoritmo de *Branch and Bound*. Sin embargo, a la hora de desarrollar un árbol existen otras estrategias que se deben tener en cuenta, como la elección del siguiente nodo o cuál de las variables que toman un valor no entero debe ser elegida para llevar a cabo la división dicotómica comentada en esta sección.

2.2.2. Elección de la variable

En este apartado vamos a comentar qué factores se deben tener en cuenta a la hora de elegir la variable x_j con la cuál vamos a efectuar la división. Para ello en primer lugar vamos a suponer que nos encontramos en un nodo activo i con el problema P^i correspondiente a este nodo y su relajación lineal asociada LP^i. Considerando a x^i como la solución óptima de este problema lineal, lo primero que debemos hacer es restringir el conjunto de variables a considerar a aquellas x^i_j tales que $x^i_j \notin \mathbb{Z}^1$. Denotaremos por $N^i = \{j \in \{1, ..., n\};\ x^i_j \notin \mathbb{Z}^1\}$ el conjunto de variables que cumplen esta condición.

Observación 2.5. *Utilizaremos la siguiente notación para los dos subproblemas de LP_i a considerar:*

- *(LP^{-i}_j): problema resultante de añadir a LP^i la restricción $x_j \leq \lfloor x^i_j \rfloor$*
- *(LP^{+i}_j): problema resultante de añadir a LP^i la restricción $x_j \geq \lfloor x^i_j \rfloor + 1$.*

A la hora de determinar que variable x_j vamos a elegir, podemos encontrar dos métodos para efectuar esta elección:

- Elección de la variable por especificaciones del usuario, es decir, existe un orden de importancia o relevancia de las variables que viene dado como parte del algoritmo. De esta forma, se elige una variable x_j que tenga una relevancia mayor que otras dentro de este orden establecido. Por ejemplo, en un problema de programación entera binaria, la variable de decisión correspondiente a llevar a cabo o no un determinado proyecto tiene mayor importancia que aquellas que puedan corresponder a decisiones relacionadas con el proyecto en sí mismo.
- Elección de la variable en función de lo que decrece el valor de la función objetivo del problema relajado LP^i, en el momento que se fuerza a la variable x_j a tomar un valor entero.

Esta segunda opción es más compleja y por ello es necesario introducir nuevos conceptos que nos permitan medir este *decrecimiento*.

Observación 2.6. *Denotaremos a estos decrecimientos con D^{-i}_j y D^{+i}_j. El primero de ellos (D^{-i}_j) hace referencia al decrecimiento de la función objetivo entre (LP^i) y (LP^{-i}_j), mientras que D^{+i}_j mide el decrecimiento entre (LP^i) y (LP^{+i}_j).*

Una buena variable para ramificar es aquella x_j cuyos decrecimientos asociados sean grandes en comparación con el resto, puesto esto nos permite conseguir una cota superior en el nodo $\overline{Z}^i$ más ajustada, lo cual es útil para que el algoritmo funcione más rápido.

La estrategia que consiste en calcular D_j^{-i} y D_j^{+i} explícitamente para cada j se conoce como *strong branching*, pero implica resolver los problemas de programación lineal (LP_j^{-i}) y (LP_j^{+i}) para cada j. Se ha demostrado con experimentos que esta estrategia reduce considerablemente el árbol de soluciones respecto a otras estrategias más sencillas; sin embargo, el coste de resolver todos estos problemas de programación lineal resulta ser demasiado alto. Por ello, presentaremos los pseudocostes, los cuales nos permiten obtener estimaciones de estos decrecimientos.

Consideraremos el problema P^i y su relajación al problema lineal LP^i, con x^i la solución óptima de este problema lineal ($Z_{LP}^i = cx^i$) y x_j^i el valor que toma la variable x_j en esta solución. Denotaremos con f_j^i la parte decimal de x_j^i, que se calcula de la siguiente forma:

$$f_j^i = x_j^i - \lfloor x_j^i \rfloor$$

Los *pseudocostes* P_j^- y P_j^+ son una estimación del coste por unidad que puede tener en el valor de la función objetivo el forzar a la variable x_j a tomar un valor entero.

- Si forzamos $x_j \to \lfloor x_j^i \rfloor$, el pseudocoste asociado viene dado por P_j^-.
- Si forzamos $x_j \to \lfloor x_j^i \rfloor + 1$, el pseudocoste asociado viene dado por P_j^+.

Para calcular estos pseudocostes podemos hacer uso de las siguientes estimaciones:

$$P_j^- = \frac{Z_{LP}^i - Z_j^-}{f_j^i} \;\; ; \;\; P_j^+ = \frac{Z_{LP}^i - Z_j^+}{1 - f_j^i}$$

En las estimaciones de los pseudocostes el valor Z_j^- denota el valor obtenido al resolver el problema lineal LP^i con el valor de $x_j = \lfloor x_j^i \rfloor$. El valor Z_j^+ está definido de la misma manera pero con $x_j = \lceil x_j^i \rceil$.

Se ha observado en la práctica que estos pseudocostes tienden a permanecer constantes a lo largo del árbol de soluciones, por lo que no es necesario calcularlos en cada nodo del árbol. Una opción es obtener una estimación inicial de ellos que nos permita utilizarla a lo largo de todos los nodos. Para obtener esta estimación podemos calcular los pseudocostes asociados a la variable x_j (P_j^- y P_j^+) la primera vez que esta variable toma un valor no entero. Podemos repetir este cálculo las r primeras veces (con $r \geq 2$) para obtener una mejor estimación.

Una vez que hemos definido estos pseudocostes y hemos proporcionado una manera de estimarlos, podemos obtener una estimación del decrecimiento que va a sufrir la función objetivo al forzar la variable x_j a tomar un valor entero. Estas estimaciones toman el nombre de *degradación* y se calculan de la siguiente forma:

- $\hat{D}_j^{-i} = f_j^i P_j^-$ es el coste estimado en la función objetivo de forzar a la variable x_j a tomar el valor $\lfloor x_j^i \rfloor$.

- $\hat{D}_j^{+i} = (1 - f_j^i)P_j^+$ es el coste estimado en la función objetivo de forzar a la variable x_j a tomar el valor $\lceil x_j^i \rceil$.

Es decir, con estos dos valores estimamos un descenso de $\hat{D}_j^{-i}$ al elegir el nodo hijo izquierdo del nodo i y un descenso de $\hat{D}_j^{+i}$ al elegir el nodo hijo derecho. Una vez que hemos calculado los valores de estas *degradaciones* para todas las variables x_j tales que $j \in N^i$, debemos establecer un criterio que nos permita decidir qué variable elegir.

- El primer criterio para la elección de la variable consiste en elegir la variable j que cumpla lo siguiente:

$$max_{j \in N^i} min \{\hat{D}_j^{-i}, \hat{D}_j^{+i}\} \tag{2.4}$$

  Utilizando este criterio elegimos a la variable x_j tal que el valor más pequeño entre $\hat{D}_j^{-i}$ y $\hat{D}_j^{+i}$ sea el mayor entre todas las asociadas a las variables x_j con $j \in N^i$. La idea es que eligiendo la variable según este criterio estamos escogiendo la variable que es más importante para obtener una solución entera, ya que es la que repercute en mayor medida sobre el valor de la función objetivo. Cuando elegimos la variable de ramificación utilizando este criterio, es recomendable que el siguiente subproblema a considerar sea el que tiene una degradación menor, es decir, elegiríamos el nodo izquierdo si y sólo si $\hat{D}_j^{-i} \leq \hat{D}_j^{+i}$.

- El segundo criterio elige la variable x_j en función de:

$$max_{j \in N^i} max \{\hat{D}_j^{-i}, \hat{D}_j^{+i}\}$$

  De esta forma, una de las ramificaciones puede resultar ser *podada* por acotación puesto que hemos elegido la mayor degradación posible, lo que implica un gran decrecimiento en el valor que toma nuestra función objetivo.

- *Regla de máxima inviabilidad*: elige la variable con el valor más alejado de un entero, es decir:

$$max_{j \in N^i} min \{f_j, 1 - f_j\}$$

- *Regla de mínima inviabilidad*: elige la variable con el valor más cercano a un entero, es decir:

$$min_{j \in N^i} min \{f_j, 1 - f_j\}$$

2.2.3. Elección del nodo

Dada una lista $\mathcal{L}$ de subproblemas activos, o equivalentemente un conjunto de nodos activos en el árbol de soluciones, la pregunta a responder es cuál de esos nodos debe ser el siguiente a estudiar. Para responder a esta pregunta debemos tener en cuenta la búsqueda de dos objetivos: encontrar una solución factible que nos permita incrementar la acotación inferior $\underline{Z}$ y probar la optimalidad de la solución factible actual, haciendo decrecer la acotación superior $\overline{Z}$ tan rápido como sea posible.

Existen dos enfoques distintos a la hora de decidir los criterios a seguir para la elección del siguiente nodo:

- Determinar antes de empezar a trabajar con el algoritmo cuáles van a ser las reglas a seguir para la elección de los nodos, es decir, el orden en el que vamos a desarrollar el árbol.
- Seguir unas reglas adaptadas a la situación en el momento de la elección de cada uno de los nodos activos, que utilizan la información que estos proporcionan, como pueden ser las acotaciones que presentan.

En el primer caso, una de las reglas más usadas se conoce en inglés como *depth-first search plus backtracking*. En este tipo de estrategia, si el nodo actual no ha sido *podado*, el siguiente nodo a considerar es uno de sus nodos *hijo*. El término *backtracking* hace referencia a que una vez que el nodo es podado, debemos volver hacia atrás en el árbol con dirección hacia la raíz (el problema original) hasta encontrar un nodo, si existe, que tenga un nodo *hijo* que todavía no ha sido considerado.

Por tanto este tipo de estrategia entra dentro del conjunto de reglas que se pueden establecer antes de comenzar el algoritmo, fijando una regla para elegir las variables a ramificar y considerando que el nodo *hijo* izquierdo se debe estudiar antes que el derecho. Un ejemplo de este tipo de estrategia a la hora de elegir un nodo activo lo podemos encontrar en la *Figura 2*, en la cual los nodos están numerados en el orden en el que han sido considerados, y aquellos que están subrayados es porque han sido podados.

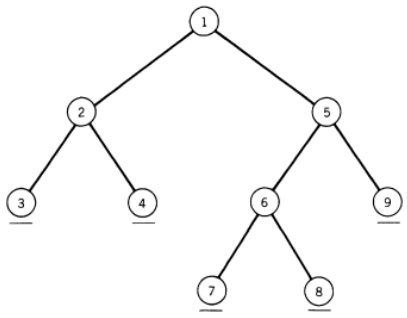

Figura 2.2: Árbol de soluciones desarrollado utilizando la estrategia de *depth-first search plus backtracking*

La principal ventaja de esta estrategia consiste en que en la mayor parte de los casos las soluciones factibles al problema de optimización se encuentran *alejadas* de la raíz del árbol, y es por ello que esta estrategia nos permite alejarnos rápidamente de la raíz hasta nodos que nos aporten soluciones factibles. Estas soluciones factibles nos dan acotaciones inferiores del problema original $\underline{Z}$, y esto es importante pues la obtención de una buena cota inferior nos permite descartar o *podar* nodos utilizando el criterio de acotación ($Z_R^i \leq \underline{Z}$)

A la hora de aplicar una estrategia de *depth-first search* es recomendable que la cota inferior $\overline{Z}$ de la que disponemos en ese momento sea una buena acotación inferior, puesto que si no lo es con esta estrategia podemos estudiar numerosos nodos consecutivos que tengan un valor objetivo alejado del valor óptimo que estamos buscando. Esto ocurre puesto que los problemas lineales que aparecen en una búsqueda de este tipo en el que el nodo siguiente es uno de los nodos hijo del nodo anterior, suelen proporcionar soluciones similares, ya que se añade una única restricción en cada paso.

Otra estrategia para la elección del nodo del primer tipo, es decir, fijando las reglas que se van a seguir para la elección desde un primer momento, es la conocida en inglés como *breadth-first search*. Para explicar esta estrategia debemos definir el nivel de un nodo como el número de *ramas* entre un nodo y la raíz del árbol. Por tanto en este caso todos los nodos que se encuentran en un mismo nivel son considerados antes que cualquiera que esté en el siguiente nivel. Sin embargo, este método de elección del nodo no es práctico para resolver problemas de programación entera usando la relajación lineal, puesto que las soluciones factibles que nos permiten obtener acotaciones inferiores suelen encontrarse en nodos alejados de la raíz del árbol, por lo que estudiar todos los nodos de un mismo nivel nos ralentiza la obtención de estas soluciones factibles, y por tanto la posibilidad de podar nodos utilizando el criterio de acotación visto en la proposición 2.3.

La opción contemplada en la mayoría de códigos comerciales es la estrategia de *depth-first search*, es decir, si el nodo no ha sido podado, entonces el siguiente a considerar es uno de sus hijos. Sin embargo, cuando un nodo es podado, el siguiente a considerar no suele ser determinado por la estrategia de *backtracking*, si no hay que otros numerosos criterios para la elección de un nodo activo.

- El primer criterio para la elección de un nodo es elegir aquel que tenga la mayor acotación superior, es decir, debemos elegir el nodo s tal que este nodo cumpla que $\overline{Z}^s = max_t \overline{Z}^t$. Utilizando esta regla denominada *regla de la mejor cota* nunca ramificaremos un nodo cuya acotación superior $\overline{Z}^t$ sea menor que el valor óptimo Z (utilizando la Proposición 2.1). Además, puesto que este nodo posee la mejor acotación superior, es también una cota superior para nuestro problema, lo que implica que no vamos a encontrar ninguna solución factible con un valor objetivo mejor que esta cota, y por tanto no va a poder ser podado por acotación en ningún caso. En definitiva, este nodo va a tener que ser estudiado tarde o temprano.

- El segundo criterio consiste en escoger un nodo el cuál sea más probable que contenga una solución óptima. Esto se debe a que una vez que hemos encontrado una solución óptima, aunque no podamos probar que lo es, habremos obtenido el mayor valor posible para la acotación inferior de Z, es decir, el mayor valor de $\underline{Z}$, y por tanto esto nos sirve de ayuda a la hora de aplicar el criterio de *poda* por acotación. Veremos un procedimiento que estima este valor óptimo Z^i. Por tanto este criterio de *mejor estimación* consiste en elegir un nodo i de la lista de problemas $\mathcal{L}$ tal que maximice $\hat{Z}^i$, siendo este $\hat{Z}^i$ una estimación del valor Z^i y tal que cumple que $\hat{Z}^i \leq \overline{Z}^i$.

Una vez que hemos visto estos criterios para elegir el nodo podemos analizar y estudiar como hallar la estimación $\hat{Z}^i$. Consideramos x^* la solución óptima al problema lineal LP^i y $Z^i_{LP} = cx^*$ el valor óptimo que alcanza la función objetivo. Utilizando las *degradaciones* vistas en el apartado anterior podemos calcular la estimación $\hat{Z}^i$ en función de si elegimos el nodo hijo izquierdo o el derecho. En el caso del izquierdo tendríamos la estimación:

$$\hat{Z}^i = Z^i_{LP} - D^{-i}_j - \sum_{k \in N^i \setminus \{j\}} min\{D^{-i}_k, D^{+i}_k\} \tag{2.5}$$

En el caso de elegir el nodo derecho la estimación sería

$$\hat{Z}^i = Z^i_{LP} - D^{+i}_j - \sum_{k \in N^i \setminus \{j\}} min\{D^{-i}_k, D^{+i}_k\} \tag{2.6}$$

Por tanto, siguiendo el segundo criterio debemos elegir el nodo que maximice esta estimación $\hat{Z}^i$. En general, la estrategia a utilizar puede variar en función del tipo de problema que se esté intentando resolver, por lo que los programas informáticos dedicados a la programación entera suelen disponer de diferentes estrategias. Estas estrategias se suele entremezclar a la hora de resolver un problema, empezando habitualmente por una estrategia de *depth-first search* para encontrar una solución factible, y utilizando el criterio de mejor estimación una vez que se ha encontrado esta solución para seleccionar nodos que nos conduzcan a soluciones óptimas.

Capítulo 3

Método de planos de corte

El algoritmo de planos de corte es un método de optimización que iterativamente refina una región factible utilizando desigualdades lineales, llamadas cortes. Este algoritmo fue propuesto por Ralph Gomory en la década de 1950 como un método para resolver problemas de programación entera y de programación entera mixta. La idea de este algoritmo es la siguiente:

- Utilizando la teoría estudiada acerca de la programación lineal, sabemos que podemos encontrar un punto extremo o vértice de la región factible en el cual la solución es óptima.
- Comprobamos si este punto es una solución entera, y en el caso de que no lo sea buscamos una desigualdad lineal que separe a esta solución óptima del problema relajado de la componente convexa del conjunto S de posibles soluciones de nuestro problema de programación entera. La búsqueda de esta desigualdad es a lo que llamamos *problema de separación*, y a la desigualdad es a lo que llamamos *corte*.
- Añadimos esta desigualdad al problema relajado y volvemos a repetir el proceso hasta que lleguemos a una solución entera óptima.

3.1. Contenido teórico del método

El método de planos de corte es interesante desde el punto de vista teórico pues ofrece una caracterización de la *envolvente entera* de un poliedro P. Presentaremos dos definiciones previas:

Definición 3.1. *(Combinación convexa)* Dado un conjunto $P \subseteq \mathbb{R}^n$, un punto $x \in \mathbb{R}^n$ es una *combinación convexa* de puntos de P si existe un conjunto finito de puntos $\{x^i\}_{i=1}^t$ en P y un conjunto $\{\lambda_i\}_{i=1}^t$ con $\lambda_i \in \mathbb{R}, \lambda_i \geq 0 \ \forall i$ y $\sum_{i=1}^t \lambda_i = 1$, tal que $x = \sum_{i=1}^t \lambda_i x^i$.

Definición 3.2. *(Envolvente convexa)* La *envolvente convexa* de P, denotada por conv(P), es el conjunto de todos los puntos que son combinaciones convexas de puntos de P:

$$conv(P) = \{x \in \mathbb{R}^n : x = \sum_{i=1}^{m} \lambda_i x_i \ , \ x_i \in P \ , \ \lambda_i \in \mathbb{R}, \lambda_i \geq 0 \ \forall i, \sum_{i=1}^{m} \lambda_i = 1\}$$

Definición 3.3. *(Envolvente entera)* La envolvente entera de P, denotada por P_I, se define como la envolvente convexa del conjunto $S = P \cap \mathbb{Z}^n$.

Nuestro objetivo en las siguientes páginas es probar la equivalencia entre el problema de programación entera (IP) con región factible el poliedro P y el problema lineal (LP) con la misma función objetivo pero con región factible $P_I = conv(P \cap \mathbb{Z}^n)$. Los dos problemas a estudiar son:

$$max\{cx : Ax \leq b,\, x \in \mathbb{Z}^n,\, x \geq 0\} \quad (IP) \tag{3.1}$$

$$max\{cx : x \in P_I,\, x \in \mathbb{R}^n,\, x \geq 0\} \quad (LP) \tag{3.2}$$

Partiendo del problema de programación entera (IP) con $P = \{x : Ax \leq b, x \in \mathbb{R}^n_+\}$ y (A, b) una matriz entera de tamaño $m \times (n+1)$, estudiaremos si el conjunto $P_I = conv(S)$ es un poliedro, con $S = \{x : Ax \leq b, x \in \mathbb{Z}^n_+\}$. Para ello tendremos en cuenta en primer lugar si P es un poliedro racional acotado o no.

En el caso que P está acotado, entonces el conjunto S es el vacío o un número finito de puntos $\{x_1, ..., x_m\}$. En este segundo caso sabemos que podemos definir el conjunto $conv(S)$ como:

$$conv(S) = \{x \in \mathbb{R}^n : x = \sum_{i=1}^{m} \lambda_i x^i \ ,\ \lambda_i \in \mathbb{R}, \lambda_i \geq 0 \ \forall i, \sum_{i=1}^{m} \lambda_i = 1\}$$

Esta descripción del conjunto $conv(S)$ es equivalante a la siguiente:

$$conv(S) = \{x \in \mathbb{R}^n : x = \lambda A \ ,\ \lambda \in \mathbb{R}^m_+, \sum_{i=1}^{m} \lambda_i = 1\} \tag{3.3}$$

En este caso la matriz A es una matriz racional de tamaño $m \times n$ cuya fila i son las coordenadas del punto x^i. Por tanto de esta forma estamos representando la combinación lineal de estos puntos como combinación lineal de las filas de una matriz. Esta descripción nos permite aplicar el siguiente teorema y afirmar que $conv(S)$ es un poliedro.

Teorema 3.1. *(Teorema de Weyl) Si A es una matriz racional $m_1 \times n$, B es una matriz racional $m_2 \times n$, y*

$$Q = \{x \in \mathbb{R}^n : x = yA + zB, \sum_{k=1}^{m_1} y_k = 1, y \in \mathbb{R}^{m_1}_+, z \in \mathbb{R}^{m_2}_+\}$$

entonces Q es un poliedro racional.

Podemos observar que la descripción de $conv(S)$ hecha en (3.3) coincide con la del teorema de Weyl siendo A la matriz cuyas filas son las coordenadas de los puntos $\{x^1, ..., x^m\}$, la matriz B es la matriz nula y el vector y se corresponde con λ, por lo que $P_I = conv(S)$ es un poliedro racional.

Si P no está acotado entonces S contiene un número infinito de puntos, y en este caso veremos que $P_I = conv(S)$ es un poliedro racional que está generado por un número finito de puntos de S y un número finito de rayos enteros de P. Para ello tomaremos un conjunto finito $Q \subset S$ y veremos que cualquier punto de S puede ser generado tomando un punto en Q más una combinación entera no negativa de rayos extremos de P.

Proposición 3.2. *Si* $P = \{x : Ax \leq b, x \in \mathbb{R}^n_+\} \neq \emptyset$ *y* $S = P \cap \mathbb{Z}^n$, *donde* (A, b) *es una matriz entera de tamaño* $m \times (n+1)$, *entonces existe un conjunto finito* $\{q^l\}_{l \in L}$ *de puntos de* S *y un conjunto finito* $\{r^j\}_{j \in J}$ *de rayos de* P *tal que*

$$S = \{x \in \mathbb{R}^n_+ : x = \sum_{l \in L} \alpha_l q^l + \sum_{j \in J} \beta_j r^j, \sum_{l \in L} \alpha_l = 1, \alpha_l \geq 0 \, \forall l \in L, \beta_j \geq 0 \, \forall j \in J\}$$

Demostración: Sean $\{x^k \in \mathbb{R}^n_+ : k \in K\}$ el conjunto finito de puntos extremos de P y sea $\{r^j \in \mathbb{R}^n_+ : j \in J\}$ el conjunto finito de rayos extremos de P. Utilizando el teorema de Minkowski (teorema 1.1) podemos representar P de la siguiente forma:

$$P = \{x \in \mathbb{R}^n_+ : x = \sum_{k \in K} \lambda_k x^k + \sum_{j \in J} \mu_j r^j, \sum_{k \in K} \lambda_k = 1, \lambda_k \geq 0 \, \forall k \in K, \mu_j \geq 0 \, \forall j \in J\}$$

Puesto que P es un poliedro racional, estos rayos extremos $\{r^j\}_{j \in J}$ los podemos considerar vectores enteros. Definimos el conjunto Q de la siguiente forma:

$$Q = \{x \in \mathbb{Z}^n_+ : x = \sum_{k \in K} \lambda_k x^k + \sum_{j \in J} \mu_j r^j, \sum_{k \in K} \lambda_k = 1, \lambda_k \geq 0 \, \forall k \in K, 0 \leq \mu_j \leq 1 \, \forall j \in J\}$$

Este conjunto Q es un conjunto finito ($Q = \{q^l \in \mathbb{Z}^n_+ : l \in L\}$) y $Q \subseteq S$. Podemos comprobar fácilmente que $x^i \in S$ si y solo si $x^i \in \mathbb{Z}^n_+$ y

$$x^i = (\sum_{k \in K} \lambda^i_k x^k + \sum_{j \in J} (\mu^i_j - \lfloor \mu^i_j \rfloor) r^j) + \sum_{j \in J} \lfloor \mu^i_j \rfloor r^j \quad con \quad \sum_{k \in K} \lambda_k = 1, \lambda_k \geq 0 \, , \mu_j \geq 0$$

En este expresión, el primer término es un punto de Q, por lo que existe $l(i) \in L$ tal que

$$x^i = q^{l(i)} + \sum_{j \in J} \beta^i_j r^j \quad con \quad \beta^i_j = \lfloor \mu^i_j \rfloor \quad \forall j \in J \tag{3.4}$$

Por tanto hemos probado que podemos escribir S como una combinación lineal de puntos de Q y rayos extremos del poliedro P. □

Utilizando el anterior teorema podemos probar el siguiente resultado, que nos muestra que el conjunto $P_I = conv(S)$ es un poliedro racional:

Proposición 3.3. *Si* $P = \{x : Ax \leq b, x \in \mathbb{R}^n_+\} \neq \emptyset$ *y* $S = P \cap \mathbb{Z}^n$, *donde* (A, b) *es una matriz entera de tamaño* $m \times (n+1)$, *entonces* $conv(S)$ *es un poliedro racional.*

Demostración: Puesto que cualquier punto $x^i \in S$ puede ser escrito en la forma de la ecuación (3.4), cualquier combinación convexa de puntos $\{x^i \in S, i \in I\}$ puede ser escrita como

$$x = \sum_{i \in I} \gamma_i x^i = \sum_{i \in I} \gamma_i (q^{l(i)} + \sum_{j \in J} \beta^i_j r^j)$$

Puesto que para cada x^i con $i \in I$ existe un l tal que $l(i)=l$, como hemos visto en la ecuación (3.4), podemos reescribir el primer sumando de la última expresión de la siguiente forma:

$$x = \sum_{l \in L} (\sum_{i \in I : l(i) = l} \gamma_i) q^l + \sum_{j \in J} (\sum_{i \in I} \gamma_i \beta^i_j) r^j = \sum_{l \in L} \alpha_l q^l + \sum_{j \in J} \beta_j r^j \tag{3.5}$$

donde $\alpha_l = \sum_{i \in I: l(i)=l} \gamma_i \geq 0$ para $l \in L$ y $\beta_j = \sum_{i \in I} \gamma_i \beta_j^i \geq 0$ para $j \in J$. También podemos comprobar que $\sum_{l \in L} \alpha_l = \sum_{i \in I} \gamma_i = 1$ (puesto que habíamos tomado una combinación convexa de elementos de S). De esta forma llegamos al conjunto *conv(S)* definido de la siguiente forma:

$$conv(S) = \{x \in \mathbb{R}_n^+ : x = \sum_{l \in L} \alpha_l q^l + \sum_{j \in J} \beta_j r^j \, , \, \sum_{l \in L} \alpha_l = 1, \, \alpha_l, \beta_j \geq 0 \; \forall l \in L, \forall j \in J\}$$

En esta expresión $q^l, r^j \in \mathbb{Z}_+^n \; \forall l \in L, \forall j \in J$, por tanto podemos reescribir estas combinaciones lineales como producto de los vectores $\alpha \in \mathbb{R}_+^{|L|}, \beta \in \mathbb{R}_+^{|J|}$ por las matrices A y B cuyas filas son las coordenadas de los puntos q^l y de los rayos r^j respectivamente. Por tanto aplicando el teorema de Weyl (teorema 3.1) hemos probado que *conv(S)* es un poliedro racional. □

Gracias a este resultado el objetivo es reformular nuestro problema original (IP)

$$max\{cx : x \in S\} \; (IP)$$

de tal forma que sea equivalente al problema de programación lineal (LP) formulado en (3.2). De esta forma si encontramos una solución óptima (LP) con el algoritmo simplex, esta se encontrará en un vértice de $P_I = conv(S)$, y puesto que en este vértice todas las variables toman un valor entero, la solución encontrada será una solución factible del problema (IP). Sin embargo, esto no es suficiente para probar que ambos problemas son equivalentes.

La equivalencia de ambos problemas la formalizaremos en el siguiente teorema, pero antes de ello probaremos dos afirmaciones que nos resultarán útiles en la demostración:

Lema 3.4. *La envolvente entera de P está contenida en P: conv(S) ⊆ P*

Demostración. Un punto $x \in conv(S)$ lo podemos escribir, utilizando la expresión de x vista en la ecuación (23), como

$$x = \sum_{l \in L} \alpha_l q^l + \sum_{j \in J} \beta_j r^j$$

Multiplicando esta expresión por A obtenemos:

$$Ax = \sum_{l \in L} \alpha_l A q^l + \sum_{j \in J} \beta_j A r^j \leq \sum_{l \in L} \alpha_l b = b$$

Llegamos a la desigualdad $Ax \leq b$ teniendo en cuenta que los puntos q^l son puntos de P y que r^j son rayos de P, por lo que $Ar^j \leq 0$. Hemos probado por tanto que $conv(S) \subseteq P$. □

Lema 3.5. *Un punto extremo de conv(S) es un punto de S*

Demostración. Si $x \in conv(S)$ es un punto extremo de $conv(S)$, entonces no se puede escribir como combinación convexa de puntos de $conv(S)$, y por tanto tampoco de S. Esto implica $x \in S$, puesto que los puntos de $conv(S)$ son por definición combinaciones convexas de puntos de S. □

Teorema 3.6. *Dado* $S = P \cap \mathbb{Z}^n \neq \emptyset$, $P = \{x : Ax \leq b, x \in \mathbb{R}^n_+\}$ *y cualquier* $c \in \mathbb{R}^n$, *entonces:*

a. *El valor de la función objetivo de (IP) no está acotado superiormente si y solo si el valor de la función objetivo de (LP) no está acotado superiormente.*

b. *Si (LP) tiene un valor óptimo acotado, entonces admite una solución óptima que es una solución óptima de (IP).*

c. *Si* x^0 *es una solución óptima de (IP), entonces* x^0 *es una solución óptima de (IP).*

Demostración: Sea z^I y z^L los valores óptimos de los problemas *(IP)* y *(LP)* respectivamente, escribiendo $z^I = \infty$ o $z^P = \infty$ si el valor de la función objetivo no está acotado superiormente. Puesto que $S \subseteq conv(S)$, esto implica que $z^I \leq z^P$. Probemos los tres apartados:

a. La desigualdad $z^I \leq z^P$ implica que si $z^I = \infty$ entonces $z^P = \infty$. Si suponemos ahora que $z^P = \infty$, entonces existe un punto extremo $x^0 \in conv(S)$ con coordenadas enteras y un rayo $r \in \mathbb{Z}^n_+$ tal que $cr > 0$ y tal que $x^0 + \lambda r \in conv(S)\ \forall \lambda \geq 0$, es decir, *conv(S)* es un conjunto no acotado, puesto que si lo fuera el valor objetivo de la función estaría acotado superiormente. Por tanto, si tomamos $\lambda \in \mathbb{Z}_+$ el punto $x^0 + \lambda r \in P \cap \mathbb{Z}^n_+ = S$, ya que es un punto con coordenadas enteras que pertenece a P. Por tanto si evaluamos $c(x^0 + \lambda r)$ y hacemos tender $\lambda \to \infty$, el valor de la función objetivo también tiende a infinito, por lo que $z^I = \infty$.

b. Utilizando que $conv(S)$ es un poliedro, si *(LP)* tiene una solución óptima, esta se alcanza en un punto extremo x^0 del poliedro. Anteriormente hemos visto que un punto extremo de $conv(S)$ es un punto extremo de S, por lo que $x^0 \in S$, lo que implica que $Z^I \geq cx^0 = z^P$, por lo que utilizando la desigualdad $z^I \leq z^P$ vemos que se da la igualdad $z^I = z^P$ y que por tanto x^0 es una solución óptima de *(IP)*.

c. Si x^0 es una solución óptima de *(IP)* entonces el valor óptimo de *(LP)* está acotado, pues en caso contrario el valor óptimo de *(IP)* no estaría acotado superiormente (apartado a). Puesto que $x^0 \in S$ y $S \subseteq conv(S)$, resulta que $x^0 \in conv(S)$. Si existiera $x^* \in conv(S)$ tal que se alcanzara un valor óptimo mayor al obtenido por x^0, entonces este x^* sería solución óptima de *(IP)* por el apartado b), lo que nos lleva a una contradicción puesto que habíamos supuesto que x^0 era una solución óptima de *(IP)*. Hemos probado por tanto que x^0 es solución óptima de *(LP)*.

□

En resumen lo que hemos probado es que podemos resolver el problema de programación entera *(IP)* resolviendo el de programación lineal *(LP)*. Sin embargo, las dificultades las encontramos en la descripción del conjunto $P_I = conv(S)$ al cual debemos restringirnos para encontrar la solución óptima de *(LP)*. Aunque existen casos en los que podemos dar una descripción explícita de $P_I = conv(S)$, en general es una tarea difícil.

La idea teórica del algoritmo de planos de corte consiste en generar una sucesión de poliedros contenidos en el poliedro original P y que aproximen de la mejor forma posible el conjunto P_I. Desarrollaremos esta idea en las siguientes páginas, pero antes de ellos definiremos dos conceptos que usaremos numerosas veces en este apartado: *desigualdad válida*,*semiespacio* y *cara de un poliedro*.

Definición 3.4. Una desigualdad $\pi x \leq \pi_0$ es una desigualdad válida para $X \subset \mathbb{R}^n$ si $\pi x \leq \pi_0 \ \forall x \in X$.

Definición 3.5. Un semiespacio es una de los dos partes en las que queda dividido un espacio afín por un hiperplano. Hablamos de semiespacio cerrado si contiene al hiperplano que delimita ambas partes.

Definición 3.6. Una cara del poliedro P es un conjunto definido de la siguiente forma

$$F := P \cap \{x \in \mathbb{R}^n \ : \ cx = \delta\}$$

donde $cx \leq \delta$ es una desigualdad válida para el poliedro P.

El concepto de desigualdad válida nos permite definir qué es un *plano de corte*, también denominado *corte*:

Definición 3.7. Un plano de corte es una desigualdad válida para el conjunto P_I que no es válida para al menos un punto del poliedro P.

Volviendo al objetivo de estudiar $P_I = conv(S)$, utilizaremos los planos de corte para encontrar la envolvente entera de nuestro poliedro P. Para ello partimos de un semiespacio H definido de la siguiente forma:

$$H = \{x : cx \leq \delta\}$$

En esta definición $cx \leq \delta$ es una desigualdad válida para nuestro poliedro P, con $c \in \mathbb{Z}^n$ un vector cuyas componentes son enteros coprimos entre sí. Definimos el conjunto H_I de la siguiente forma:

$$H_I = \{x : cx \leq \lfloor \delta \rfloor\}$$

Hemos definido la envolvente entera de nuestro semiespacio H desplazando el hiperplano que delimita nuestro semiespacio hasta que el hiperplano contiene vectores enteros. Podemos comprobar fácilmente que $\forall x \in P \cap \mathbb{Z}^n$ la desigualdad $cx \leq \lfloor \delta \rfloor$ es una desigualdad válida, puesto al pertenecer x a P se cumple $cx \leq \delta$ y puesto que tanto c como x son vectores enteros, $cx \in \mathbb{Z}$, lo que implica que se cumple la desigualdad $cx \leq \lfloor \delta \rfloor$. Por tanto esta desigualdad $cx \leq \lfloor \delta \rfloor$ es una desigualdad válida para el conjunto P_I.

Definición 3.8. Una desigualdad de la forma $cx \leq \lfloor \delta \rfloor$ es un corte de Chvátal-Gomory de P si $c \in \mathbb{Z}^n$ y $cx \leq \delta$ es una desigualdad válida para P.

Una vez que hemos presentado este tipo de desigualdades, podemos definir la clausura de Chvátal-Gomory como el siguiente conjunto:

$$P' := \cap_{cx \leq \delta \ valida \ para \ P, \ c \in \mathbb{Z}^n} \{x : \ cx \leq \lfloor \delta \rfloor\} \tag{3.6}$$

Este conjunto P' satisface las siguientes contenciones: $P_I \subseteq P' \subseteq P$.

- $P_I \subseteq P'$: puesto que todas las desigualdades que aparecen en (3.6) son desigualdades válidas para el conjunto P_I, cualquier punto $x \in P_I$ pertenece a P'.

- $P' \subseteq P$: cualquiera fila del sistema $Ax \leq b$ que define el poliedro P es una desigualdad válida para P con coeficientes enteros (recordemos que (A, b) era una matriz entera), por lo que aparece en la intersección. Si $x \in P'$, x satisface $Ax \leq b$, y por tanto $x \in P$

El siguiente paso consiste en probar que si P es un poliedro racional, como es nuestro caso, entonces P' también lo es. Si esto es cierto, podríamos iterar el proceso de tal forma que lleguemos a la cadena de contenciones:

$$P_I \subseteq ... \subseteq P^{(n)} \subseteq ... \subseteq P'' \subseteq P' \subseteq P \tag{3.7}$$

Nuestro objetivo en último lugar es probar que existe un t natural tal que $P^{(t)} = P_I$, definiendo $P^{(t)}$ de la siguiente forma:

$$P^0 = P \ ; \ P^{(t+1)} = P^{(t)'}$$

Presentaremos dos lemas con una gran importancia en las siguientes demostraciones. El primero de ellos es la versión entera del *Lema de Farkas*. Este primer lema se utiliza en la demostración del segundo lema que presentaremos, que a su vez utilizaremos en la demostración de la siguiente proposición.

Lema 3.7. *(Farkas, versión entera) Sea A una matriz racional y b un vector racional. El sistema $Ax = b$ no admite solución entera si y sólo si existe un vector $u \in \mathbb{R}^m$ tal que $uA \in \mathbb{Z}^n$,$ub \notin \mathbb{Z}$.*

Lema 3.8. *Sea $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, entonces $cx \leq \delta$ es una desigualdad válida si y solo si $\exists u \geq 0$ tal que $c = uA$ y $\delta \geq ub$.*

Teorema 3.9. *Sea P un poliedro racional, entonces P' es un poliedro racional.*

Demostración: Dado el sistema $Ax \leq b$ que define P, sabemos por el lema precedente que $cx \leq \delta$ es una desigualdad válida para P si y solo si $\exists u \geq 0$ tal que $c = uA$ y $\delta \geq ub$. Por tanto en lugar de escribir la desigualdad $cx \leq \delta$ podemos escribir $uAx \leq \delta$. Además puesto que $P' \subseteq P$, todos los puntos de P' satisfacen $Ax \leq b$, por lo que también se cumple con $u \geq 0$ que $uAx \leq ub \leq \delta$. Imponiendo la condición de que $uA \in \mathbb{Z}^n$ (ya que $c \in \mathbb{Z}^n$), entonces llegamos a $uAx \leq ub$ válida para P, y por tanto $uAx \leq \lfloor ub \rfloor$ válida $\forall x \in P \cap \mathbb{Z}^n$. En resumen, podemos reescribir P' de la siguiente forma:

$$P' = \cap_{u \geq 0,\ uA \in \mathbb{Z}^n} \{x : \ uAx \leq \lfloor ub \rfloor\}$$

Si comprobamos que con un número finito de restricciones del tipo $uAx \leq \lfloor ub \rfloor$ somos capaces de describir el conjunto P', entonces habremos probado que es un poliedro racional, pues lo habremos definido como el conjunto de puntos que satisfacen un número finito de restricciones con coeficientes enteros.

Vamos a comprobar que solo necesitamos vectores u tales que todas sus variables toman valores en $[0, 1]$, para ello tomamos $u \geq 0$ tal que $uA \in \mathbb{Z}^n$ y reescribimos u como $u = u' + u''$ donde $u' = \lfloor u \rfloor$, $u' \geq 0$, $u' \in \mathbb{Z}^n$ y $u'' = u - \lfloor u \rfloor$, por lo que u'' verifica que $0 \leq u'' \leq 1$. Estos dos vectores generan las siguientes desigualdades:

- La desigualdad $u'Ax \leq u'b$ es válida para P puesto que $u' \geq 0$, y por tanto también lo es para P_I ya que $P_I \subseteq P$.

- Probaremos que la desigualdad $u''Ax \leq \lfloor u''b \rfloor$ es una desigualdad del tipo Chvátal Gomory. Es decir, debemos probar que $u''A \in \mathbb{Z}^n$ y que $u''Ax \leq u''b$ es válida para P.

  - $u''A = uA - \lfloor u \rfloor A \in \mathbb{Z}^n$ ya que $uA \in \mathbb{Z}^n$, pues es una de las condiciones que se han impuesto en la intersección, y $\lfloor u \rfloor A \in \mathbb{Z}^n$ pues $\lfloor u \rfloor \in \mathbb{Z}^n$ y A es una matriz entera.
  - $u''Ax \leq u''b$ es válida para P puesto que $u'' \geq 0$, y por tanto es una combinación lineal con coeficientes no negativos de las filas de A.

Por tanto, ambas desigualdades son desigualdades válidas para P_I. Si sumamos ambas llegamos a:

$$u'Ax + u''Ax = uAx \leq \lfloor ub \rfloor$$

Para el lado derecho de la desigualdad hemos tenido en cuenta la suma del lado derecho de ambas desigualdades :

$$u'b + \lfloor u''b \rfloor = u'b + \lfloor ub - \lfloor u \rfloor b \rfloor = u'b + \lfloor ub \rfloor - \lfloor u \rfloor b = \lfloor ub \rfloor \quad (u' = \lfloor u \rfloor)$$

Por tanto la desigualdad $uAx \leq \lfloor ub \rfloor$ es la desigualdad de Chvátal-Gomory dada por u, la cual hemos visto que podemos escribir como la suma de:

- $u'Ax \leq u'b$, desigualdad válida para P y que por tanto está implícita en el sistema de desigualdades $Ax \leq b$ que define P. Todas estas desigualdades las podemos obtener con un vector $u = e_i = (0, ..., 1, ..., 0)$, el cual verifica $0 \leq u_i \leq 1 \ \forall i \in \{1, ..., n\}$.
- $u''Ax \leq \lfloor u''b \rfloor$, desigualdad de Chvatal Gomory con $0 \leq u''_i \leq 1 \ \forall i \in \{1, ..., n\}$.

Hemos comprobado que podemos modificar la definición de P' de la siguiente forma:

$$P' = \cap_{0 \leq u \leq 1, \ uA \in \mathbb{Z}^n} \{x : \ uAx \leq \lfloor ub \rfloor\}$$

Debemos comprobar finalmente que en esta intersección aparece un número finito de desigualdes. Para ello veamos que el cardinal del siguiente conjunto es finito:

$$\{u \in \mathbb{R}^n : \ 0 \leq u_i \leq 1 \ \forall i \in \{1, ..., n\}, \ uA \in \mathbb{Z}^n\}$$

Podemos comprobar esto teniendo en cuenta que el conjunto $T = \{u \in \mathbb{R}^n : 0 \leq u_i \leq 1 \ \forall i \in \{1, ..., n\}$ es un conjunto acotado. Por tanto al multiplicar todos estos puntos por la matriz A, el conjunto $T_A = \{uA : 0 \leq u_i \leq 1 \ \forall i \in \{1, ..., n\}$ sigue siendo un conjunto acotado, lo que implica que existe un número finito de puntos con coordenadas enteras en su interior.

Hemos comprobado que con un número finito de restricciones con coeficientes enteros del tipo $uAx \leq \lfloor ub \rfloor$ somos capaces de describir el conjunto P', por lo que hemos probado que P' es un poliedro racional. □

Una vez que hemos demostrado que P' es un poliedro racional, debemos probar que existe t tal que $P^{(t)} = P_I$.

Definición 3.9. Se denomina *rango de Chvátal* del poliedro P al menor $t \in \mathbb{N}$ tal que $P^{(t)} = P_I$.

Antes de probar que todo poliedro racional tiene un rango de Chvátal finito presentaremos tres lemas necesarios para la demostración del teorema.

Lema 3.10. *Sea $P \subseteq \mathbb{R}^n$ un poliedro racional no vacío tal que aff(P)$\cap\mathbb{Z}^n \neq \emptyset$. Si $P_I = \emptyset$, entonces $dim(rec(P)) < dim(P)$.*

Lema 3.11. *Sea $P \subseteq \mathbb{R}^n$ un poliedro racional no vacío tal que aff(P)$\cap\mathbb{Z}^n \neq \emptyset$. Entonces $P_I = \{x : Ax \leq b\}\cap$aff(P) con b vector entero y A una matriz entera tal que para toda fila a_i de A se cumple que:*

- *a_i no es ortogonal a aff(P).*
- *Existe $d_i \in \mathbb{R}$ tal que $a_i x \leq d_i$ es válida para el poliedro P.*

Lema 3.12. *Sea P un poliedro racional y F una cara no vacía de P. Entonces $F^{(s)} = P^{(s)} \cap F$ para todo $s \in \mathbb{Z}_+$.*

Teorema 3.13. *Para cada poliedro racional P, existe un número natural t tal que $P^{(t)} = P_I$.*

Demostración. Realizaremos esta demostración por inducción sobre la dimensión del poliedro P. Si $dim(P) = -1$ ($P = \emptyset$) o $dim(P) = 0$ (P es un punto) la demostración es trivial. Suponemos $d = dim(P)$ y en primer lugar consideramos el caso en el que aff(P)$\cap\mathbb{Z}^n = \emptyset$. Utilizando el teorema 1.2 sabemos que aff(P) se puede expresar como el conjunto de puntos solución del sistema $A^{=}x = b^{=}$, donde tomamos solo las igualdades implícitas del sistema $Ax \leq b$ que define nuestro poliedro P.

Si aff(P)$\cap\mathbb{Z}^n = \emptyset$ entonces el sistema $A^{=}x = b^{=}$ no admite solución entera, por lo que aplicando el lema de Farkas en su versión entera (lema 3.7) sabemos que existe $u \in \mathbb{R}^m$ tal que $a = uA \in \mathbb{Z}^n$ y tal que $ub = d \notin \mathbb{Z}$. Por tanto tenemos un sistema equivalente al que define la envolvente afín de P, y se da la contención $P \subseteq \{x \in \mathbb{R}^n : ax = d\}$. Si tomamos ahora las desigualdades $ax \leq d$ (válida para P), y la desigualdad $-ax \leq d$ (válida para P), podemos definir el conjunto P':

$$P' = \{x : ax \leq \lfloor d \rfloor, -ax \leq \lfloor -d \rfloor = -\lceil d \rceil\} = \{x : ax \leq \lfloor d \rfloor, ax \geq \lceil d \rceil\} = \emptyset = P_I$$

Por lo tanto hemos probado que existe t finito con $P^{(t)} = P_I$.

Estudiamos ahora el caso en el que aff(P)$\cap\mathbb{Z}^n \neq \emptyset$. Utilizando el lema 3.11 sabemos que podemos definir el conjunto P_I de la siguiente forma: $P_I = \{x : Mx \leq n\}\cap$aff(P) con n un vector entero y M una matriz entera tal que para cada fila m_i de M, m_i no sea ortogonal a aff(P) y $m_i x \leq d_i$ sea válida para P con $d_i \in \mathbb{R}^n$.

Ahora debemos probar que para cada fila m_i de M, existe un entero positivo t_i tal que la desigualdad $m_i x \leq n_i$ sea válida para $P^{(t_i)}$. Si probamos esto entonces llegaremos a que tomando $t^* = max_i\{t_i\}$, entonces $P^{(t^*)} \subseteq P_I$, puesto que todas las desigualdades que definen P_I son válidas para $P^{(t^*)}$. Utilizando ahora que $P_I \subseteq P^{(t^*)}$ (por la cadena de contenciones (3.7)) llegamos a la igualdad $P_I = P^{(t^*)}$, y por tanto al objetivo del teorema.

Supongamos que esto no ocurre, es decir fijada una fila m_i no existe un entero positivo t_i tal que la desigualdad $m_i x \leq n_i$ sea válida para $P^{(t_i)}$. Puesto que $m_i x \leq d_i$ sí que es válida para $P^0 = P$, existen enteros $d > n_i$ y $r \geq 0$ tal que para todo $s \geq r$ la desigualdad $m_i x \leq d$ es válida para $P^{(s)}$ pero $m_i x \leq d-1$ no lo es. Podemos definir una cara de $P^{(r)}$ tal que:

$$F := P^{(r)} \cap \{x : m_i x = d\}$$

F es una cara puesto que $m_i x \leq d$ es válida para $P^{(r)}$. Sin embargo, $F_I = conv(F \cap \mathbb{Z}^n) = \emptyset$, ya que si $\exists x \in F \cap \mathbb{Z}^n$ entonces:

- $x \in F$, por lo que $m_i x = d$
- $x \in P_I$, por lo que en particular $m_i x \leq n_i < d$, lo que es absurdo por el resultado inmediatamente anterior.

Sabemos que $F \subseteq P$, por lo que $dim(F) \leq dim(P)$; sin embargo, no se puede dar la igualdad puesto que si se diera la igualdad entonces compartirían la misma envolvente afín, por lo que el vector m_i sería ortogonal a *aff(F)=aff(P)*, pero lo hemos elegido tal que m_i no es ortogonal a *aff(P)*. Por tanto aplicando inducción sobre la dimensión de F ($dim(F) < d$) y teniendo en cuenta que $F_I = \emptyset$, sabemos que $\exists h$ tal que $F^{(h)} = \emptyset$.

Utilizando el lema 3.12 sabemos que $F^{(h)} = P^{(r+h)} \cap F$, pero este conjunto es vacío, por lo que $m_i x < d$ para todo $x \in P^{(r+h)}$, lo que implica que en particular al estudiar $(P^{(r+h)})' = P^{(r+h+1)}$ obtenemos la desigualdad $m_i x \leq \lfloor d \rfloor = d-1$ válida para $P^{(r+h+1)}$, lo que contradice la elección de d y r. □

Desde un punto de vista teórico el método desarrollado por Gomory consistía en resolver problemas de programación lineal sobre $P, P', P'', ...$ hasta llegar al poliedro racional $P^t = P_I$, en el cual encontraremos una solución entera óptima. Este método no es nada aconsejable en la práctica, pues requiere generar una gran cantidad de planos de corte para definir cada P^i, y repetir este proceso hasta llegar al poliedro que coincida con la envolvente entera de P.

Es por ello que en los siguientes apartados presentaremos el algoritmo que se utiliza en los problemas prácticos que se resuelven aplicando cortes, así como volveremos a estudiar las desigualdades de Chvátal-Gomory debido a su gran importancia a la hora de buscar planos de corte.

3.2. Algoritmo del método

En este apartado presentaremos un algoritmo que se puede poner en práctica para resolver un problema de programación entera utilizando los planos de corte. Para ello en primer lugar tenemos el conjunto de soluciones enteras $X = P \cap \mathbb{Z}^n$ y una familia $\mathcal{F}$ de desigualdades válidas para el conjunto X:

$$\pi x \leq \pi_0 \ ; \ (\pi, \pi_0) \in \mathcal{F}$$

Por tanto, partimos de nuestro problema (IP) con la notación usual:

$$max\{cx : x \in X\} \, ; \, X = P \cap \mathbb{Z}^n \, ; \, P = \{x : Ax \leq b, x \in \mathbb{R}^n_+\} \ (IP)$$

A continuación presentaremos un algoritmo básico que utiliza los planos de corte para resolver este problema.

- Paso 1 (*Inicialización*): $t = 0$ y $P^0 = P$
- Paso 2 (*Iteración* t): resuelve el problema lineal

  $$Z^t = max\{cx : x \in P^t\}$$

  denotando a x^t como la solución óptima.
- Paso 3: si $x^t \in \mathbb{Z}^n$, x^t es una solución óptima para nuestro problema IP y el algoritmo termina.
- Paso 4: si $x^t \notin \mathbb{Z}^n$, entonces se resuelve el *problema de separación* para la solución x^t y la familia $\mathcal{F}$ (búsqueda de una desigualdad en $\mathcal{F}$ que no satisfaga la solución x^t).
- Paso 5: si encontramos una desigualdad $(\pi^t, \pi_0^t) \in \mathcal{F}$ tal que $\pi^t x^t > \pi_0^t$, entonces podemos definir el conjunto

  $$P^{t+1} = P^t \cap \{x : \pi^t x^t \leq \pi_0^t\}$$

  La solución x^t queda fuera de este conjunto, por lo tanto volvemos al paso 2 aumentando t en una unidad $(t = t + 1)$.
- Paso 6: Si no hemos encontrado una desigualdad $(\pi^t, \pi_0^t) \in \mathcal{F}$ tal que $\pi^t x^t > \pi_0^t$, el algoritmo finaliza.

Si llegamos al paso 6 no hemos obtenido una solución entera para nuestro problema IP, sin embargo si que hemos conseguido una formulación mejorada de nuestro problema, pues hemos llegado al conjunto:

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

En este caso podemos poner en práctica otros algoritmos como el de *Branch and Cut* para llegar a la solución de este problema.

3.3. Procedimiento de Chvátal-Gomory

En esta sección estudiaremos el procedimiento de Chvátal-Gomory para construir desigualdades válidas para el conjunto $X = P \cap \mathbb{Z}^n$, siendo $P = \{x : Ax \leq b, x \in \mathbb{R}_+^n\}$, con A una matriz $m \times n$ con columnas $\{a_1, ..., a_n\}$, $b \in \mathbb{R}^m$ y $u \in \mathbb{R}_+^m$.

- En primer lugar la desigualdad

  $$u \sum_{j=1}^{n} a_j x_j = \sum_{j=1}^{n} u a_j x_j \leq ub$$

  es válida para P puesto que $u \geq 0$ y $\sum_{j=1}^{n} a_j x_j \leq b$, ya que este sumatorio es una forma de expresar el producto Ax, el cual sabemos que para todo $x \in P$ satisface que $Ax \leq b$.

- La desigualdad

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq ub$$

es válida para P teniendo en cuenta que:

  - $x_j \geq 0 \ \forall j \in \{1, ..., n\}$ $(x \in \mathbb{R}^n_+)$
  - $\lfloor ua_j \rfloor \leq ua_j \forall j \in \{1, ..., n\}$ y hemos verificado que la desigualdad anterior se cumplía.

- La desigualdad

$$\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor$$

es válida para X puesto que $x \in \mathbb{Z}^n$, por lo que $\sum_{j=1}^{n} \lfloor ua_j \rfloor x_j \in \mathbb{Z}$.

Este procedimiento nos permite generar todas las desigualdades válidas para un problema de programación entera, resultado que recoge el siguiente teorema:

Teorema 3.14. *Toda desigualdad válida para X se puede obtener aplicando el procedimiento de Chvátal-Gomory un número finito de veces.*

La prueba de este teorema es larga y viene recogida en el tercer apartado del capítulo 8 dedicado a los algoritmos de planos de corte del libro [2]. Esta demostración consta de 5 pasos diferenciados y en ella se toma una desigualdad válida $\pi x \leq \pi_0$ para el conjunto $X = P \cup \mathbb{Z}^n$ con $P = \{x \in \mathbb{R}^n : Ax \leq b, 0 \leq x \leq 1\}$. Se prueba que esta desigualdad es una desigualdad de Chvàtal-Gomory, es decir, que se puede obtener aplicando este procedimiento un número finito de veces.

3.3.1. Algoritmo de planos de corte de Gomory

En este apartado pondremos en práctica el algoritmo de los planos de corte con el corte fraccional de Gomory. Para ello consideraremos el problema *(IP)* con la notación usual $(max\{cx : Ax \leq b, x \in \mathbb{Z}^n_+\})$ y resolveremos el problema lineal relajado *(LP)* para encontrar una solución óptima y consecuentemente una base óptima (conjunto de las variables básicas). Una vez que hemos obtenido esa base podemos elegir una variable básica que no sea entera y generar una desigualdad del tipo Chvàtal-Gomory en la restricción asociada a esta variable básica, de tal forma que esta nueva desigualdad no la cumpla la solución óptima encontrada previamente.

Antes de explicar este algoritmo definiremos qué es una base para el sistema $Ax = b$, mismo sistema que el utilizado para el problema (IP) que ha sido ampliado añadiendo las variables de holgura necesarias para convertir las desigualdades en igualdades. Recordamos que A es una matriz $m \times n$ con $m < n$.

Definición 3.10. Una base es una selección de columnas de la matriz A, es decir, $B \subset \{a_1, ...a_n\}$, tal que $\mid B \mid = m$ y tal que la matriz A_B de tamaño $m \times m$ obtenida al seleccionar las m columnas que indica B es una matriz invertible.

De esta manera podemos escribir el sistema $Ax = b$ como $A_B x_B + A_N x_N = b$, y utilizando que la matriz A_B es invertible llegamos a

$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b \tag{3.8}$$

La solución básica asociada a este problema es la solución obtenida sustituyendo el vector $x_N = 0$, por lo que $x_B = A_B^{-1} b$. Al resolver un problema de programación lineal con el método simplex, llegamos a una solución básica óptima, a una base B y a un sistema de la forma $x_B + A_B^{-1} A_N x_N = A_B^{-1} b$.

Volviendo a nuestro problema original (IP), si aplicamos el método simplex al problema lineal asociado (LP) obtenemos una solución básica óptima x^* y una base B. Si $x^* \in \mathbb{Z}^n$, entonces hemos encontrado la solución óptima para el problema de programación entera. Si la solución básica óptima obtenida x^* no es entera, entonces existe una $u \in B$ tal que $x_u = \beta_u \notin \mathbb{Z}$ (puesto que $x_N = 0$ en 3.8). Utilizando la base obtenida llegamos al sistema de la ecuación (3.8), que podemos reescribir de tal forma que:

$$max\ \alpha_{00} + \sum_{j \in N} \alpha_{0j} x_j\ ;\ sujeto\ a\quad x_u + \sum_{j \in N} \alpha_{uj} x_j = \beta_u\ \ para\ \ u \in B\,(base)$$

Observación 3.11. *N hace referencia al conjunto de variables no básicas, las cuales toman el valor* 0.

En esta expresión los coeficientes α_{0j} son ≤ 0, puesto que si fueran mayores que 0 podríamos encontrar una solución óptima asignando valores mayores que 0 a las variables no básicas; además, los coeficientes β_u son ≥ 0 pues estamos buscando soluciones positivas. Tomando la fila u tal que $\beta_u \notin \mathbb{Z}$, llegamos a la ecuación:

$$x_u + \sum_{j \in N} \alpha_{uj} x_j = \beta_u \tag{3.9}$$

Efectuando el procedimiento de Chvatal-Gomory llegamos a la desigualdad:

$$x_u + \sum_{j \in N} \lfloor \alpha_{uj} \rfloor x_j \leq \lfloor \beta_u \rfloor \tag{3.10}$$

Restando esta desigualdad a la igualdad (3.9) llegamos a la desigualdad:

$$\sum_{j \in N} (\alpha_{uj} - \lfloor \alpha_{uj} \rfloor) x_j \geq \beta_u - \lfloor \beta_u \rfloor\ \ reescrita\ \ como\ \ \sum_{j \in N} f_{uj} x_j \geq f_{u0} \tag{3.11}$$

donde $f_{uj} = \alpha_{uj} - \lfloor \alpha_{uj} \rfloor$ para $j \in N$ cumplen $0 \leq f_{uj} < 1$ y $f_{u0} = \beta_u - \lfloor \beta_u \rfloor$. Este último valor cumple que $0 < f_{u0} < 1$ puesto que habíamos elegido la fila u tal que $\beta_u \notin \mathbb{Z}$. Puesto que en nuestra solución óptima las variables x_j con $j \in N$ toman el valor 0, entonces la desigualdad no se cumple para esta solución óptima x^*, ya que sustituyendo en la ecuación (3.11) llegamos a $0 \geq f_{u0} > 0$, es decir, $0 > 0$. De esta forma hemos obtenido una desigualdad

$$\sum_{j \in N} f_{uj} x_j \geq f_{u0}$$

que se puede añadir a nuestro problema original y volver a calcular una solución óptima con esta nueva restricción.

3.3.2. Cortes de Gomory en Programación Entera Mixta

Hasta ahora hemos tratado problemas donde todas las variables debían tomar valores enteros, sin embargo existe una gran cantidad de problemas donde existen variables discretas y variables continuas. Es por ello que en este apartado presentaremos los planos de corte estudiados por Gomory para este tipo de problemas. Antes de presentar este tipo de planos de corte comentaremos un método que utilizaremos para generar estos cortes. El método se denomina *redondeo entero mixto*, denominado como *M.I.R* por su traducción al inglés). En este método partimos de una desigualdad con dos variables: una variable discreta $z \in \mathbb{Z}$ y una variable continua positiva $s \in \mathbb{R}_+$, tal que verifiquen $z \leq \beta + s$ con $\beta \notin \mathbb{Z}$ y $f = \beta - \lfloor \beta \rfloor > 0$.

Proposición 3.15. *(Desigualdad M.I.R) La desigualdad*

$$z \leq \lfloor \beta \rfloor + \frac{s}{1-f} \quad (M.I.R)$$

es una desigualdad válida $\forall (z, s)$ *que satisfacen* $z \leq \beta + s$.

Demostración. En primer lugar suponemos que $z \leq \lfloor \beta \rfloor$ y puesto que $0 \leq s$, entonces $0 \leq \frac{s}{1-f}$, por lo que se verifica que $z \leq \lfloor \beta \rfloor + \frac{s}{1-f}$. Supongamos ahora que $z \geq \lfloor \beta \rfloor + 1$. A continuación vamos a sumar las dos desigualdades siguientes:

$$(-z \leq -\lfloor \beta \rfloor - 1)(\frac{f}{1-f}) \quad ; \quad (z \leq \beta + s)(\frac{1}{1-f})$$

$$-\frac{zf}{1-f} + \frac{z}{1-f} \leq \frac{-f\lfloor \beta \rfloor - f}{1-f} + \frac{\beta + s}{1-f} \; ; \; \frac{(1-f)z}{(1-f)} \leq \frac{-f\lfloor \beta \rfloor - f + f + \lfloor \beta \rfloor}{1-f} + \frac{s}{1-f}$$

$$z \leq \frac{(1-f)\lfloor \beta \rfloor}{1-f} + \frac{s}{1-f} \; ; \; z \leq \lfloor \beta \rfloor + \frac{s}{1-f}$$

De esta forma hemos llegado a la desigualdad pedida, y hemos comprobado que es válida $\forall (z, s)$ que satisfacen $z \leq \beta + s$. □

Una vez que hemos estudiado este método, consideraremos el siguiente problema:

$$max\{cx : x \in \mathbb{R}^n,\, Ax \leq b,\, x \geq 0,\, x_i \in \mathbb{Z}\, \forall i \in I \subseteq \{1, ..., n\}\} \quad (MILP)$$

En este caso A es una matriz racional de tamaño $m \times n$, mientras que b y c son vectores tales que $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$. I representa el conjunto de índices i tales que $x_i \in \mathbb{Z}$, mientras que C representa el conjunto de variables continuas. Al igual que en el caso de programación entera pura, debemos resolver la relajación lineal del problema utilizando el método simplex, lo cual nos lleva al sistema:

$$max\ \alpha_{00} + \sum_{j \in N} \alpha_{0j} x_j \,;\ sujeto\ a \quad x_u + \sum_{j \in N} \alpha_{uj} x_j = \beta_u \ \ para\ \ u \in B\, (base)$$

Sea la solución óptima de la relajación lineal $\overline{x}$, si $\exists l \in B \cap I$ tal que $\overline{x}^l \notin \mathbb{Z}$, entonces vamos a estudiar como generar un corte de Gomory para la igualdad:

$$x_l + \sum_{j \in N} \alpha_j x_j = \beta$$

En primer lugar sabemos que $b \notin \mathbb{Z}$ porque $\overline{x}^l \notin \mathbb{Z}$ y $x_j = 0 \; \forall j \in NB$. Podemos definir $f = \beta - \lfloor \beta \rfloor$ y $f_j = \alpha_j - \lfloor \alpha_j \rfloor$. Vamos a transformar la igualdad en una desigualdad teniendo en cuenta que x es un vector positivo.

$$x_l + \sum_{j \in I \cap N, f_j \leq f} \lfloor \alpha_j \rfloor x_j + \sum_{j \in I \cap N, f_j > f} (\lfloor \alpha_j \rfloor + 1 - (1 - f_j)) x_j + \sum_{j \in C \cap N, \alpha_j < 0} \alpha_j x_j \leq \beta \quad (3.12)$$

Podemos descartar de esta desigualdad que $\lfloor \alpha_j \rfloor + 1 - (1 - f_j) = \alpha_j$ y que hemos omitido las variables x_j con coeficiente $\alpha_j \geq 0$, llegando de esta forma a la desigualdad. El siguiente paso consiste en convertir esta desigualdad en una del tipo $z \leq \beta + s$ con $z \in \mathbb{Z}$,$b \notin \mathbb{Z}$ y $s \geq 0$.

$$x_l + \sum_{j \in I \cap N, f_j \leq f} \lfloor \alpha_j \rfloor x_j + \sum_{j \in I \cap N, f_j > f} (\lfloor \alpha_j \rfloor + 1) x_j \leq \beta + \sum_{j \in I \cap N, f_j > f} (1 - f_j) x_j - \sum_{j \in C \cap N, \alpha_j < 0} \alpha_j x_j$$

De esta forma la parte izquierda de la desigualdad toma un valor entero, mientras que

$$s = \sum_{j \in I \cap NB, f_j > f} (1 - f_j) x_j - \sum_{j \in C \cap NB, \alpha_j > 0} \alpha_j x_j \geq 0$$

Por tanto, podemos aplicar el método de redondeo entero mixto *(MIR)* visto previamente para obtener la desigualdad:

$$x_l + \sum_{j \in I \cap N, f_j \leq f} \lfloor \alpha_j \rfloor x_j + \sum_{j \in I \cap N, f_j > f} (\lfloor \alpha_j \rfloor + 1) x_j \leq \lfloor \beta \rfloor + \sum_{j \in I \cap N, f_j > f} \frac{(1 - f_j)}{(1 - f)} x_j - \sum_{j \in C \cap N, \alpha_j < 0} \frac{\alpha_j}{(1 - f)} x_j$$

Teniendo en cuenta ahora que si $f_j \leq f$ entonces el coeficiente asociado a x_j es $\lfloor \alpha_j \rfloor$, y en el caso de $f_j > f$ entonces es

$$\lfloor \alpha_j \rfloor + 1 - \frac{(1 - f_j)}{(1 - f)} = \lfloor \alpha_j \rfloor + \frac{(f_j - f)}{(1 - f)}$$

Tomando por tanto el coeficiente $\lfloor \alpha_j \rfloor + \frac{(f_j - f)^+}{(1 - f)} \; \forall j \in I \cap N$ llegamos al corte de Gomory:

$$x_l + \sum_{j \in I \cap N} (\lfloor \alpha_j \rfloor + \frac{(f_j - f)^+}{(1 - f)}) x_j + \sum_{j \in C \cap N, \alpha_j < 0} \frac{\alpha_j}{(1 - f)} x_j \leq \lfloor \beta \rfloor$$

La solución óptima $\overline{x}$ no satisface esta desigualdad, puesto que $x_j = 0 \, \forall j \in N$ y $x_l = \beta \lfloor \beta \rfloor$. Podemos comparar en el caso que tengamos un problema de programación entera pura si este corte de Gomory es más restrictivo que el obtenido con el corte fraccional de Gomory visto en el apartado anterior y recogido en la desigualdad 3.10. Las dos desigualdades obtenidas con estos procedimientos son las siguientes:

$$x_l + \sum_{j \in N} \lfloor \alpha_j \rfloor x_j \leq \lfloor \beta \rfloor \; (C.F.G) \; ; \; x_l + \sum_{j \in N} (\lfloor \alpha_j \rfloor + \frac{(f_j - f)^+}{(1 - f)}) x_j \leq \lfloor \beta \rfloor \; (C.E.M.G)$$

Puesto que $\frac{(f_j - f)^+}{(1 - f)}$ siempre toma un valor positivo, toda solución que satisface la segunda desigualdad satisface la primera; es decir, el corte obtenido mediante este procedimiento para problemas de programación entera mixta también es aplicable para problemas de programación entera pura pues obtiene una desigualdad mejor. Por tanto, en todos los programas que resuelven problemas de optimización viene implementado este segundo método.

Capítulo 4

Método de *Branch and Cut*

La búsqueda de soluciones de un problema de programación entera puede llegar a requerir la combinación de diferentes ideas y métodos. Un ejemplo de esta combinación de procedimientos es el método de *Branch and Cut* (*ramificación y corte*), desarrollado en los años 90 del siglo pasado y el cual combina el método de *Branch and Bound* y los planos de corte para obtener una solución óptima al problema presentado.

El algoritmo general de *Branch and Cut* que presentaremos en el siguiente apartado es similar al de *Branch and Bound*, puesto que se encarga de dividir el problema de programación entera original en subproblemas que se representan en un árbol de soluciones y resolver en cada nodo la relajación lineal del problema correspondiente para posteriormente utilizar tres criterios que determinen si se poda el nodo o se divide en dos subproblemas.

Aunque la principal modificación que se realiza en el algoritmo de *Branch and Cut* respecto al de *Branch and Bound* consiste en generar planos de corte en todos (o en algunos) de los nodos del árbol de soluciones de forma que se obtengan mejores aproximaciones a la solución de cada subproblema, existe también un enfoque diferente a la hora de poner el algoritmo en práctica. Esto se debe a que se intenta profundizar el estudio del problema asociado a un nodo en concreto si este nodo puede conducir a la obtención de una mejor acotación superior para nuestro problema, estudio que puede conllevar el uso de preprocesamiento en el nodo o la utilización de métodos heurísticos. Es decir, el objetivo no es sólo estudiar la relajación del problema asociado en un nodo para poder utilizar los criterios de acotación, si no utilizar diferentes herramientas que nos conduzcan a obtener la mejor acotación posible.

Puesto que el método de *Branch and Cut* se suele aplicar con problemas de programación entera mixta, de aquí en adelante consideraremos el siguiente problema:

$$max\{cx : x \in \mathbb{R}^n,\, Ax \leq b,\, x \geq 0,\, x_j \in \mathbb{Z} \forall j \in J \subset \{1, ..., n\}\} \quad (MILP)$$

En este caso A es una matriz racional de tamaño $m \times n$, mientras que b y c son vectores tales que $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$. Denotamos con Z_I el valor máximo que alcanza la función objetivo de nuestro problema *(MILP)*, y utilizaremos la siguiente notación para cada subproblema P^i:

$$max\{cx : x \in X^i\} \quad (P^i)$$

4.1. Algoritmo del método

En este apartado presentaremos un algoritmo general del método de *Branch and Cut*, recordando previamente que $\mathcal{L}$ representaba la lista de nodos activos N^i y sus problemas correspondientes P^i que todavía no han sido resueltos. Cada problema P^i está relacionado con el problema lineal LP^i, el cual se obtiene eliminando las restricciones de integralidad de las variables correspondientes.

- Paso 1 (*Inicialización*): añadimos el nodo N^0 correspondiente al problema *(MILP)* a la lista $\mathcal{L}$ de nodos activos ($\mathcal{L} = \{N^0\}$). Denotamos $x^* = \emptyset$ y $\underline{Z} = -\infty$ como primera cota inferior. En este paso también podemos utilizar métodos heurísticos para obtener una solución factible x^* y una cota inferior $\underline{Z}$ de mejor calidad.

- Paso 2 *(Test de finalización):* si $\mathcal{L} = \emptyset$, entonces la solución x^* es óptima y vamos al paso 7. En el caso que $x^* = \emptyset$, el problema *(MILP)* es un problema no factible o no acotado.

- Paso 3 *(Elección del nodo):* seleccionamos un nodo N^i de $\mathcal{L}$ y lo eliminamos de la lista. Inicializamos $k = 1$ y denotamos al problema asociado al nodo elegido como $P^{i,k}$

- Paso 4 *(Relajación lineal):* resolvemos la relajación lineal ($LP^{i,k}$) del problema seleccionado $P^{i,k}$. Si el problema no está acotado, *(MILP)* tampoco lo está, y el algoritmo termina. Si $LP^{i,k}$ no es factible, volvemos al paso 3. En caso contrario, denotamos a la solución óptima de $LP^{i,k}$ con $x^{i,k}$ y al valor alcanzado por la función objetivo con $Z^{i,k}$.

  - Si $Z^{i,k} \leq \underline{Z}$, entonces volvemos al punto 3.
  - Si $x^{i,k}$ es una solución factible de *(MILP)*, entonces $\underline{Z} = Z^{i,k}$, $x^* = x^{i,k}$. Eliminamos de la lista $\mathcal{L}$ cualquier nodo N^j cuya acotación $\overline{Z}^j$ sea conocida y cumpla que $z\overline{Z}^j \leq \underline{Z}$. Volvemos al paso 3.

  Si $x^{i,k}$ no es factible para *(MILP)*, vamos al siguiente paso.

- Paso 5 *(Elección de planos de corte)*: en este paso debemos decidir si buscamos planos de corte para la solución $x^{i,k}$. Si los buscamos y los encontramos, añadimos estos cortes a la relajación lineal del problema y volvemos al paso 4, con $k = k + 1$. En caso contrario, vamos al siguiente paso.

- Paso 6 (*Ramificación):* ramificamos el problema $LP^{i,k}$ en subproblemas LP^{i+1}, LP^{i+2}, tales que la unión de sus regiones factibles no contenga la solución actual $x^{i,k}$ pero contenga todas las soluciones factibles de P^i. Añadimos estos nuevos nodos a la lista $\mathcal{L}$ y volvemos al paso 3.

- Paso 7 *(Solución óptima)*: devolvemos x^* como solución óptima, y $Z = cx^*$ como valor que toma la función objetivo.

Revisando el algoritmo podemos observar que los dos pasos principales de la estrategia consisten en resolver la relajación lineal del problema en cada nodo del árbol y decidir, en caso de que la solución encontrada no sea factible para nuestro problema, si se buscan planos de corte para esta solución.

En caso afirmativo, debemos añadirlos a la relajación y resolver el problema con la nueva restricción. Comparando este algoritmo con el presentado en el apartado correspondiente al de *Branch and Bound* podemos observar que la principal diferencia se encuentra en la toma de esta decisión, ya que el resto del algoritmo presenta una estructura muy similar.

Existen diversas implementaciones del método de *Branch and Cut*, y en algunas de ellas se fija un número determinado de rondas de búsqueda de planos de corte en cada nodo. Este número puede variar en función de si el nodo es la raíz del árbol, en el cual el número de rondas sería mayor; o si es uno de los nodos restantes. En estos nodos que van apareciendo a medida que desarrollamos el árbol de soluciones puede llegar un momento en el cual la solución a una relajación añadiendo un nuevo plano diste muy poco de la solución obtenida en anteriores relajaciones, ya que existe un gran número de variables que toman un valor ya fijado. En este caso es recomendable dejar de trabajar en la búsqueda de planos de corte en este nodo en concreto y ramificar el problema.

Existe una variante de este algoritmo llamada *Cut and Branch*, en la cual los planos de corte se añaden solo en el nodo raíz del árbol. Esto implica un gran esfuerzo inicial al generar diversos planos de corte, pero también presenta beneficios, puesto que estos cortes generados son válidos en todos los nodos restantes del árbol, ya que son válidos en la raíz, y por tanto no es necesario determinar nuevos planos de corte en cada nodo.

La búsqueda de planos de corte en nodos del árbol diferentes al nodo raíz presenta una consecuencia clara, y es que estos planos de corte pueden no ser válidos para otros subproblemas en los nodos restantes del árbol de soluciones. En el caso de que esto ocurra, hablamos de un *plano de corte local*, el cual solo es válido para el subproblema correspondiente y sus descendientes. Esto implica que en cada nodo del árbol estamos añadiendo diferentes versiones con diferentes planos de corte del problema original.

Debemos tener en cuenta que a la hora de tomar decisiones debemos favorecer aquellas que nos permitan podar el árbol de soluciones lo más rápido posible, lo que implica buscar estrategias que obtengan buenas acotaciones superiores e inferiores del valor óptimo del problema estudiado *(MILP)* para poder utilizar el criterio de acotación. Para obtener estas cotas, tanto los métodos heurísticos como los planos de corte representan una buena herramienta. También es importante aplicar un preprocesamiento del problema que nos permita eliminar restricciones innecesarias, fijar variables o simplificar el problema. Los dos siguientes apartados tratarán de este preprocesamiento y de diferentes métodos heurísticos utilizados en este tipo de problemas, ya que las particularidades del método de *Branch and Bound* y de los planos de corte ya han sido tratadas en sus correspondientes capítulos.

4.2. Preprocesamiento

Antes de resolver cualquier problema de programación lineal es recomendable comprobar que la formulación planteada es tan fuerte como sea posible utilizando la información de la que disponemos. Todos los paquetes comerciales que utilizan el método de *Branch and Cut* llevan a cabo este procedimiento, que realiza diferente operaciones, entre las que se encuentran:

- Comprobar si existen restricciones redundantes, es decir, restricciones que se pueden eliminar sin modificar la región factible de soluciones; y en el caso de existir suprimirlas de la formulación.
- Comprobar si se pueden mejorar las acotaciones dadas de las variables, y en caso afirmativo modificarlas.
- Comprobar si utilizando las restricciones dadas se puede fijar el valor de alguna variable, y en caso afirmativo fijarlo.
- En el caso de problemas con variables binarias, estudiar implicaciones lógicas que relacionen dos variables y utilizarlas para fijar el valor de alguna de ellas.

Si estas comprobaciones dan lugar a la eliminación de restricciones o la mejora de acotaciones, se ha conseguido un sistema más sencillo y que por tanto se puede resolver mucho más rápido. Este procedimiento es de vital importancia en el caso de aplicar un algoritmo de *Branch and Cut*, ya que una mejora en la formulación puede ahorrar una gran cantidad de nodos a estudiar y por tanto de problemas a resolver.

Centrándonos en el estudio de las diferentes restricciones, podemos utilizar las acotaciones relacionadas con cada variable x_i y los correspondientes coeficientes que aparecen en la parte izquierda de la ecuación para comparar la cota obtenida con la parte derecha. Las posibles situaciones que se pueden encontrar al realizar esta comparación vienen recogidas en la siguiente proposición:

Proposición 4.1. *Consideramos el conjunto*

$$S = \{x : a_0x_0 + \sum_{j=1}^{n} a_jx_j \leq b, l_j \leq x_j \leq k_j, \forall j \in \{0, 1, ..., n\}$$

.

- *Cota de la variable x_0: si $a_0 > 0$, entonces:*

$$x_0 \leq \frac{(b - \sum_{j\geq 1, a_j>0} a_jl_j - \sum_{j\geq 1, a_j<0} a_jk_j)}{a_0}$$

  En el caso $a_0 < 0$ se cambia el $\leq$ por el $\geq$.

- *Restricción redundante: la restricción $a_0x_0 + \sum_{j=1}^{n} a_jx_j \leq b$ es redundante si:*

$$\sum_{j,a_j>0} a_jk_j + \sum_{j:a_j<0} a_jl_j \leq b$$

- *Restricción no factible: el conjunto $S = \emptyset$ si:*

$$\sum_{j,a_j>0} a_jl_j + \sum_{j:a_j<0} a_jk_j > b$$

Demostración. La prueba de la primera desigualdad parte de la desigualdad inicial que define el conjunto S y aisla la parte izquierda (a_0x_0) llegando a la desigualdad:

$$a_0x_0 \leq b - \sum_{j=1}^{n} a_jx_j = b - \sum_{j\geq 1, a_j>0} a_jx_j - \sum_{j\geq 1, a_j<0} a_jx_j$$

Utilizando ahora que $l_j \leq x_j \leq k_j, \forall j \in \{0, 1, ..., n\}$ obtenemos:

$$\sum_{j \geq 1, a_j > 0} a_j x_j \geq \sum_{j \geq 1, a_j > 0} a_j l_j \; ; \; \sum_{j \geq 1, a_j < 0} a_j x_j \geq \sum_{j \geq 1, a_j < 0} a_j k_j$$

Invirtiendo estas desigualdades multiplicando por -1 a ambos lados y dividiendo entre el coeficiente a_0 llegamos a la desigualdad pedida.

Las demostraciones de los dos apartados restantes se hacen de una manera muy similar utilizando las desigualdades $l_j \leq x_j \leq k_j$ y el signo de los coeficientes a_j.En el segundo caso llegamos a una restricción redundante pues las desigualdades $l_j \leq x_j \leq k_j$ implican que la suma $\sum_{j=0}^{n} a_j x_j$ tome un valor menor que b. En el tercer caso ocurre al contrario, el conjunto S es vacío pues el mínimo valor que toma la suma $\sum_{j=0}^{n} a_j x_j$ es mayor que la cota b. □

Este preprocesamiento también se puede aplicar para fijar el valor de alguna de las variables si se dan las condiciones adecuadas. Estas condiciones vienen recogidas en la siguiente proposición:

Proposición 4.2. *Dado un problema de maximización*

$$max\{cx \; : \; Ax \leq b, l_j \leq x_j \leq k_j \, \forall j \in \{1, ..., n\}\}$$

- *Si* $a_{ij} \geq 0 \, \forall i = 1, ..., m$ *y* $c_j < 0$, *entonces* $x_j = l_j$.
- *Si* $a_{ij} \leq 0 \, \forall i = 1, ..., m$ *y* $c_j > 0$, *entonces* $x_j = k_j$.

Demostración. Estudiaremos el primer caso, ya que el segundo caso se demuestra de forma análoga. Utilizando que el coeficiente a_{ij} que acompaña a la variable x_j en cada una de las m restricciones es positivo, sabemos que $a_{ij} l_j \leq a_{ij} x_j \; \forall i = 1, ..., m$, por lo que:

$$\sum_{k \in \{1,...,n\} \setminus j}^{n} a_{ik} x_k + a_{ij} l_j \leq \sum_{k=1}^{n} a_{ik} x_k \leq b_i$$

Por lo tanto si x_j toma el valor l_j se cumplen todas las restricciones que aparecen en $Ax \leq b$. Utilizando ahora que $c_j < 0$ sabemos que $c_j x_j \leq c_j l_j$, por lo que a la hora de maximizar cx tenemos la desigualdad:

$$\sum_{k=1}^{n} c_k x_k \leq \sum_{k \in \{1,...,n\} \setminus j}^{n} c_k x_k + c_j l_j$$

Por tanto para maximizar la suma $cx = \sum_{k=1}^{n} c_k x_k$, el valor de x_j debe ser l_j. □

Si nos centramos ahora en la programación lineal entera, se puede utilizar el redondeo para mejorar las acotaciones. Es decir, las acotaciones $l_j \leq x_j \leq k_j \, \forall j \in \{1, ..., n\}$ se pueden sustituir por $\lceil l_j \rceil \leq x_j \leq \lfloor k_j \rfloor \, \forall j \in \{1, ..., n\}$.

Por último, centraremos nuestro estudio en el preprocesamiento de problemas que utilizan variables binarias. En este caso, es recomendable revisar las diferentes restricciones pues nos pueden conducir a restricciones que impliquen solo a dos variables e incluso a fijar el valor de algunas de ellas.

Ejemplo 4.1. *Uno de los problemas más conocidos utilizando este tipo de variables binarias es el Problema de la mochila. Partiendo de una mochila que soporta un peso P y de n objetos con un peso asociado a cada uno de p_i y un valor asociado de b_i, el problema consiste en decidir qué objetos se meten en la mochila de tal forma que se maximice la suma del valor de los diferentes objetos que contiene la mochila sin superar el peso máximo que puede soportar la misma. La solución al problema vendrá dado por una secuencia de variables binarias $x_1, x_2, ..., x_n$ donde el valor de x_i indica si el objeto i es introducido ($x_i = 1$) o si no lo es.*

Utilizando el planteamiento del problema de la mochila, hablaremos de conjunto *knapsack* (traducción inglesa de *mochila*) para referirnos a un conjunto de variables binarias del tipo:

$$X = \{x \in \{0,1\}^n : \sum_{j=1}^{n} p_j x_j \leq P \, , \, p_j \geq 0 \; \forall j\} \tag{4.1}$$

Proposición 4.3. *Dado un conjunto X de la forma 4.1 (conjunto knapsack), sabemos que:*

- $X = \{0,1\}^n$ *si* $\sum_{j=1}^{n} p_j \leq P$.
- *Si* $p_k > P$, *entonces* $x_k = 0$ *en todas las soluciones factibles.*
- *Si* $p_j + p_k > P$ *con* $j \neq k$, *entonces* $x_j + x_k \leq 1$ *en todas las soluciones factibles.*

Demostración. Las demostraciones de las diferentes afirmaciones son sencillas y se pueden resumir brevemente:

- Si $\sum_{j=1}^{n} p_j \leq P$, entonces $\sum_{j=1}^{n} p_j x_j \leq \sum_{j=1}^{n} p_j \leq P$ para todo $x \in \{0,1\}^n$. Por lo tanto
$$X = \{x \in \{0,1\}^n : \sum_{j=1}^{n} p_j x_j \leq P \, , \, p_j \geq 0 \; \forall j\} = \{0,1\}^n$$
- Si $p_k > P$ entonces $\sum_{j=1}^{n} p_j x_j > P$ si $x_k = 1$, por lo tanto para toda solución factible $x_k = 0$.
- Si $p_j + p_k > P$, entonces $\sum_{j=1}^{n} p_j x_j > P$ si $x_j = x_k = 1$; por lo tanto si $x_j = 1$ entonces $x_k = 0$ y viceversa, lo que se traduce en la restricción $x_j + x_k \leq 1$.

□

Si generalizamos ahora los conjuntos de este tipo incluyendo también coeficientes negativos llegamos a la restricción $\sum_{j=1}^{n} a'_j z_j \leq b', z \in \{0,1\}^n$. La inclusión de las llamadas *variables complementarias* nos permiten reconducir esta desigualdad al caso anterior. Para ello definimos la variable complementaria de z_j como $\overline{z}_j = 1 - z_j$, y reescribimos la desigualdad de la siguiente manera:

$$\sum_{j=1}^{n} a_j x_j \leq b, \; x \in \{0,1\}^n$$

En esta desigualdad $a_j = \mid a'_j \mid$ y $x_j = z_j$ si $a'_j > 0$, mientras que $x_j = \overline{z}_j = 1 - z_j$ si $a'_j < 0$. El término independiente b se define como $b = b' - \sum_{j:a'_j<0} a'_j$. De esta forma con la inclusión de las variables complementarias podemos combinar parejas de desigualdades donde aparacen dos variables de tal forma que nos permitan fijar determinadas variables:

Proposición 4.4. *Dado un conjunto X de la forma $X = \{x \in \{0,1\}^n : \sum_{j=1}^{n} a_j x_j \leq b\}$ con $a_j \geq 0\ \forall j$:*

- *Si $x_j + x_k \leq 1$ y $x_j + \overline{x}_k \leq 1$, entonces $x_j = 0$.*
- *Si $x_j + x_k \leq 1$ y $\overline{x}_j + \overline{x}_k \leq 1$, entonces $x_j + x_k = 1$.*
- *Si $x_j + \overline{x}_k \leq 1$ y $\overline{x}_j + \overline{x}_k \leq 1$, entonces $x_k = 1$.*

Demostración. La demostración de las tres afirmaciones se realiza de forma similar, por lo que solamente demostraremos la primera. En este caso tenemos:

$$x_j + x_k \leq 1\ ;\ \ x_j + 1 - x_k \leq 1\ \Rightarrow\ x_j + x_k \leq 1\ ;\ x_j - x_k \leq 0\ \Rightarrow\ 2x_j \leq 1\ \Rightarrow\ x_j = 0$$

□

4.3. Métodos heurísticos

Un método heurístico es un procedimiento para resolver un problema complejo de optimización mediante una aproximación intuitiva a la solución. La inclusión al inicio del algoritmo de este tipo de métodos nos permiten encontrar soluciones factibles, y por tanto obtener mejores acotaciones inferiores de nuestro problema original ($\underline{z}$). Presentaremos dos métodos heurísticos diferentes: la estrategia de *diving* (buceo), y la estrategia de *local branching* (ramificación local). Cada uno de estos procedimientos se puede aplicar en cualquier nodo del árbol de soluciones generado con el método de *Branch and Cut* y se puede implementar de diferentes formas.

- *Estrategia de buceo*: en esta estrategia se escoge una variable x_j que toma un valor fraccional en la solución actual y se añade la restricción $x_j \leq \lfloor x_j \rfloor$ ó $x_j \geq \lceil x_j \rceil$, posteriormente se resuelve el problema lineal con esta nueva restricción y se repite el proceso hasta que se encuentra una solución del problema o se llega a una solución no factible. Para elegir la variable x_j se pueden seguir diferentes reglas, como elegir la variable x_j con el menor valor fraccional. Es decir, si denotamos a $\overline{f_j} = \lfloor x_j \rfloor - x_j$, escogeremos la variable j tal que se alcanza $min_j\{min(\overline{f_j}, 1 - \overline{f_j})\}$ entre las variables $x_j \notin \mathbb{Z}$, añadiendo la restricción $x_j \leq \lfloor x_j \rfloor$ si $\overline{f_j} < 1/2$ y $x_j \geq \lceil x_j \rceil$ en caso contrario.

  También se pueden seguir otras reglas más sofisticadas que utilizan la información que aportan los pseudocostes para elegir el problema *hijo* que presenta mayor estimación de la función objetivo del problema (IP). Este procedimiento heurístico se puede repetir desde diferentes nodos N_i para mejorar las posibilidades de obtener una buena solución.

- *Estrategia de ramificación local:* una vez que hemos obtenido una solución factible x^*, este método se aplica para encontrar una mejor solución en un entorno de la solución. Para ello, y asumiendo que las variables x_j toman valores $\{0,1\}$, definimos el entorno de x^*:

$$\sum_{j=1}^{p} \mid x_j - x_j^* \mid \leq k\ ;\ \ k \in \mathbb{Z}$$

  Una vez que tenemos esta restricción, la podemos añadir a nuestra formulación y volver a resolver el problema, aunque antes de añadirla debemos linearizarla:

$$\sum_{j:\, x_j^* = 0} x_j + \sum_{j:\, x_j^* = 1} (1 - x_j) \leq k\ ;\ \ k \in \mathbb{Z}$$

Esta restricción nos permite restringir la búsqueda de la solución a un entorno de x^* y no a toda la región factible. Si el procedimiento encuentra una mejor solución que x^*, entonces definimos un nuevo entorno de esta solución y volvemos a aplicar el procedimiento hasta que no encontremos una mejor solución.

Este método heurístico admite diferentes variaciones, y uno de esos ejemplos es el RINS, cuyas siglas en inglés hacen referencia a la búsqueda en un entorno inducido por la solución de una relajación. La idea de este método es la utilización de las dos soluciones que aparecen cuando aplicamos un algoritmo de *Branch and Cut*. En primer lugar disponemos de una solución factible x^*, la cual cumple las restricciones de integralidad pero puede estar alejada de ser una solución óptima para nuestra función objetivo. Por otra parte disponemos de la solución $\overline{x}$ a la relajación lineal del problema correspondiente al nodo en el que nos encontramos, la cual no cumple las condiciones de integralidad pero cuyo valor objetivo es mayor que el asociado a x^*, ya que si no lo es podemos podar el nodo por acotación. Por tanto cada una de las dos soluciones mencionadas anteriormente cumple una sola condición de las dos que buscamos: que sea una solución óptima de nuestro problema y que tome valores enteros.

Mientras que alguna de las variables de las soluciones anteriores deben tomar diferentes valores, pues son soluciones distintas, otras variables pueden coincidir. Este método utiliza las soluciones x^* y $\overline{x}$, fijando el valor de las variables que toman el mismo valor en ambas soluciones y resolviendo el problema de programación entera en las variables restantes, trabajando en un entorno de ambas soluciones. El objetivo es encontrar una solución factible con un valor de la función objetivo mejorado.

Feasibility pump

En los ejemplos anteriores hemos resuelto el problema inicial con algunas variaciones como añadir restricciones o fijar el valor de ciertas variables. Sin embargo, existen problemas con una gran complejidad en los cuales la resolución de estos problemas supone un gran coste computacional. Otro factor a tener en cuenta es que ambas estrategias parten de una solución factible, por lo que antes de aplicarlas debemos disponer de una. Debido a ambas razones presentaremos a continuación un procedimiento heurístico llamado *Feasibility pump* que permite obtener una solución factible utilizando dos secuencias de puntos construidas de tal forma que converjan a una solución factible de nuestro problema.

Consideramos el problema:

$$max\{cx : x \in \mathbb{R}^n,\ Ax \leq b,\ x \geq 0,\ x_j \in \mathbb{Z}\ \forall j \in \mathcal{J} \subset \{1, ..., n\}\} \quad (MILP)$$

Observación 4.1. *Denotaremos por $\overline{x}$ el redondeo a un punto $x \in P = \{x : Ax \leq b\}$, en el cual cada variable toma el valor $\overline{x}_j = [x_j]$, $j \in \mathcal{J}$, donde $[\cdot]$ representa el redondeo al entero más cercano.*

El método denominado *Feasibility pump* parte de un punto $\hat{x}$ en el cuál las variables discretas toman valores enteros, y busca el punto más cercano $x^* \in P$ utilizando la norma L^1: $\Delta(x, \hat{x}) = \sum_{j \in \mathcal{J}} |x_j - \hat{x}_j|$. Este punto de mínima distancia se determina resolviendo el problema de programación lineal $min\{\Delta(x, \hat{x}) : Ax \leq b\}$.

Sea x^* la solución a este problema, si $\Delta(x^*, \hat{x}) = 0$, entonces hemos obtenido una solución factible de nuestro problema *(MILP)*, que es x^*. En caso contrario, podemos aplicar el redondeo al punto $x^* \in P$, y volver a repetir el procedimiento buscando la solución factible más cercana al nuevo punto entero $\hat{x}$ (redondeo entero de x^*). Geométricamente, este método genera dos trayectorias de puntos: $x_i^* \in P$, que satisfacen las restricciones lineales pero no las enteras; y $\hat{x}_i$, que satisfacen tomar valores enteros en las variables necesarias pero no las restricciones lineales.

4.4. Ejemplo de aplicación del *Branch and Cut*

En este apartado presentaremos un ejemplo sencillo que nos permite mostrar cómo se trabaja con un algoritmo de *Branch and Cut*. Partiendo de una solución óptima del problema de programación lineal resultante de eliminar las restricciones enteras, ramificaremos el problema inicial en dos subproblemas, para posteriormente resolver estos dos subproblemas y aplicar los cortes de Gomory vistos en el anterior capítulo. El problema a considerar es el siguiente:

$$max \;\; 3x_1 + 2x_2 \;\; sujeto \;\; a$$

$$-x_1 + 3x_2 \leq 7 \;;\; 2x_1 + 5x_2 \leq 12 \;\;;\;\; 5x_1 + 3x_2 \leq 17 \;\;;\;\; x_1, x_2 \in \mathbb{Z}$$

Utilizando las variables de holgura y eliminando las restricciones enteras podemos reescribar el problema relajado en la forma adecuada para poder aplicar el método símplex.

$$\begin{aligned} Z - 3x_1 - 2x_2 &= 0 \\ -x_1 + 3x_2 + s_1 &= 7 \\ 2x_1 + 5x_2 + s_2 &= 12 \\ 5x_1 + 3x_2 + s_3 &= 17 \end{aligned}$$

Una vez que hemos reescrito el problema de esta forma podemos aplicar el método símplex, el cual proporciona la siguiente solución óptima, que no es una solución entera:

$$x_1 = \frac{49}{19} = 2 + \frac{11}{19} \;\;;\;\; x_2 = \frac{26}{19} = 1 + \frac{7}{19} \;\;;\;\; Z = \frac{199}{19} = 10 + \frac{9}{19} \tag{4.2}$$

Ramificaremos el problema original en dos subproblemas, tomando como variable para efectuar la división la variable x_1 y añadiendo a la formulación las desigualdades $x_1 \leq 2$ y $x_1 \geq 3$, respectivamente. El árbol de soluciones presenta el siguiente aspecto:

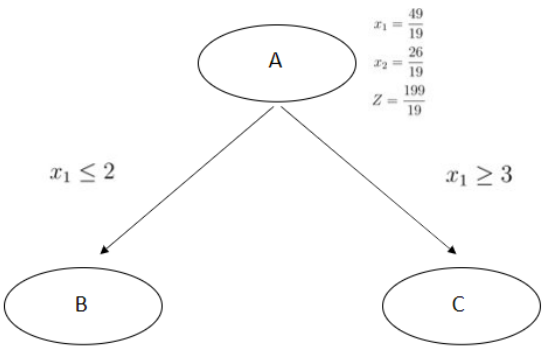

Figura 4.1: Árbol de soluciones generado al dividir el problema inicial

Siguiendo el algoritmo previamente expuesto del *Branch and Cut*, y puesto que esta solución no es una solución entera, podíamos haber buscado planos de corte y añadirles a la formulación o bien ramificar este problema en dos subproblemas. En nuestro caso hemos tomado la segunda opción.

Estudiaremos en primer lugar el subproblema asociado al nodo B. En este caso debemos añadir al problema inicial la restricción $x_1 \leq 2$, la cual podemos reescribir utilizando la variable de holgura de la siguiente forma: $x_1 + s_4 = 2$. Añadiendo esta nueva restricción y esta nueva variable de holgura llegamos a la siguiente tabla del método simplex:

	Z	x_1	x_2	s_1	s_2	s_3	s_4	b
Z	1	0	0	0	2/5	0	11/5	46/5
s_1	0	0	0	1	-3/5	0	11/5	21/5
x_2	**0**	**0**	**1**	**0**	**1/5**	**0**	**-2/5**	**8/5**
x_1	0	1	0	0	0	0	1	2
s_3	0	0	0	0	-3/5	1	-11/5	11/5

Tabla 4.1: Tabla símplex asociada al nodo B

Atendiendo a esta tabla la solución es la siguiente:

$$x_1 = 2 \ ; \ x_2 = \frac{8}{5} \ ; \ Z = \frac{46}{5} \tag{4.3}$$

La solución no es, por tanto, una solución entera, lo que implica que debemos tomar de nuevo la decisión de buscar un plano de corte o subdividir el problema. En este caso, decidimos utilizar los cortes de Gomory desarrollados en el apartado 3.3. Partiendo de la igualdad que aparece en la cuarta fila de la tabla 4.1, la correspondiente a la variable x_2, y aplicando el procedimiento de Gomory llegamos a la desigualdad:

$$x_2 + \lfloor\frac{1}{5}\rfloor s_2 - \lfloor\frac{2}{5}\rfloor s_4 = \lfloor\frac{8}{5}\rfloor \ \rightarrow \ x_2 - s_4 \leq 1$$

Añadiendo esta nueva restricción a la formulación de nuestro problema llegamos a la solución:

$$x_1 = 2 \ ; \ x_2 = 1 \ ; \ Z = 8 \tag{4.4}$$

Puesto que hemos llegado a una solución entera, podemos podar este nodo por optimalidad y estudiar el nodo restante, el nodo C. En este caso debemos añadir la restricción $x_1 \geq 3$ a la formulación de nuestro problema, convertida en $x_1 - s_4 = 3$ utilizando la variable de holgura. Aplicando el método símplex llegamos a la tabla:

	Z	x_1	x_2	s_1	s_2	s_3	s_4	b
Z	1	0	0	0	0	2/3	1/3	31/3
s_1	0	0	0	1	0	-1	-6	8
x_2	**0**	**0**	**1**	**0**	**0**	**1/3**	**5/3**	**2/3**
x_1	0	1	0	0	0	0	-1	3
s_2	0	0	0	0	1	-1/3	-17/3	8/3

Tabla 4.2: Tabla símplex asociada al nodo C

Atendiendo a esta tabla la solución es la siguiente:

$$x_1 = 3 \ ; \ \ x_2 = \frac{2}{3} \ ; \ \ Z = \frac{31}{3} \tag{4.5}$$

La solución no es entera, y en este caso de nuevo decidiremos utilizar los cortes de Gomory. Partiendo de la igualdad que aparece en la fila correspondiente a la variable x_2, y aplicando el procedimiento de Gomory llegamos a la desigualdad:

$$x_2 + \lfloor \frac{1}{3} \rfloor s_3 + \lfloor \frac{5}{3} \rfloor s_4 = \lfloor \frac{2}{3} \rfloor \ \ \rightarrow \ \ x_2 + s_4 \leq 0$$

Añadiendo esta nueva restricción a la formulación de nuestro problema llegamos a la solución:

$$x_1 = 3 \ ; \ \ x_2 = 0 \ ; \ \ Z = 9 \tag{4.6}$$

Esta solución es una solución entera, y por tanto podemos podar este nodo por optimalidad. Una vez que no existen nodos que no hayan sido estudiados, podemos verificar que la solución asociada al nodo C presenta un valor de la función objetivo mayor, por lo que la solución $x_1 = 3$ y $x_2 = 0$ es la solución óptima. En el siguiente árbol de soluciones podemos comprobar el desarrollo entero de la resolución del problema:

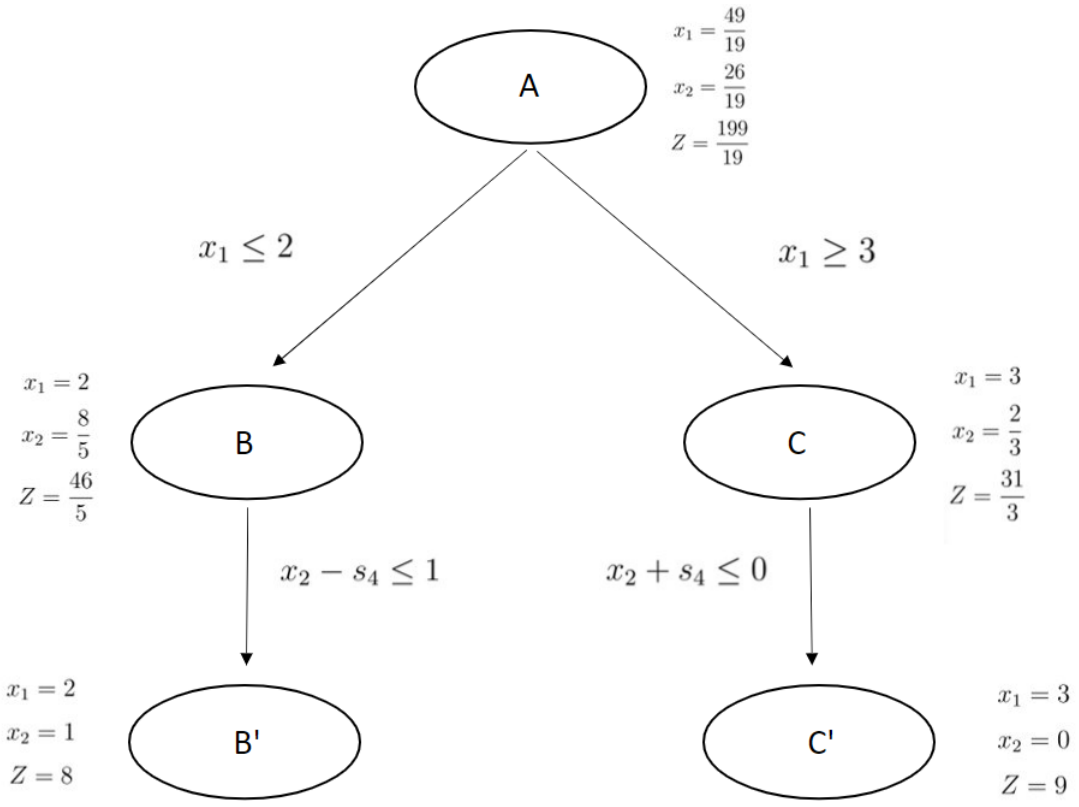

Figura 4.2: Árbol de soluciones generado al aplicar el método de Branch and Cut

Capítulo 5

Caso práctico: *Team Orienteering Problem*

5.1. Presentación del problema

El problema que vamos a estudiar en este apartado se conoce como el *Team Orienteering Problem,(TOP)*, y su origen se encuentra en un problema más sencillo, conocido como el *Single Competitor Orienteering Problem,(OP)* cuyo planteamiento está relacionado con el deporte de la orientación. Este deporte consiste en obtener la mayor puntuación entre todos los competidores al realizar un recorrido desde un punto de salida a un punto de llegada pasando por diferentes puntos de control en un tiempo máximo; y teniendo en cuenta que no todos los puntos de control nos aportan la misma puntuación, así como que una vez que has pasado por un punto de control ya has obtenido esa puntuación y no la puedes volver a conseguir. El problema *(OP)* se encarga de estudiar el recorrido que debe hacer el corredor de tal forma que maximice la puntuación obtenida y que no exceda el tiempo límite.

El *Team Orienteering Problem,(TOP)*, propuesto por primera vez en el artículo [9], consiste en una versión de este problema *(OP)* en el cual no debemos determinar la ruta de un solo corredor si no la de varios corredores que componen un equipo. La suma de cada una de las puntuaciones que consiga cada miembro del equipo es la puntuación total del equipo, por lo que la puntuación a maximizar es la suma de las diferentes puntuaciones obtenidas en cada ruta. El planteamiento es por tanto muy similar al anterior, teniendo en cuenta que dos participantes de un mismo equipo no deben pasar por el mismo punto 2 veces ya que la puntuación asignada a ese punto solo la obtienen la primera vez que pasaron. La solución del problema debe determinar el conjunto de puntos que se van a visitar, el corredor que va a visitar cada punto y el orden en el que cada corredor visita los puntos asignados.

Partiendo de este planteamiento podemos trasladar este problema a diversas situaciones como la distribución de combustible a un conjunto de vehículos o la búsqueda de jugadores de fútbol entre las universidades de un país. En el primer caso la puntuación otorgada a cada vehículo es la urgencia de combustible que presenta cada vehículo, y en el segundo a cada universidad se le asocia un determinado potencial asociado a la calidad de sus jugadores. Éstos son solo dos ejemplos de la gran variedad de problemas que se pueden plantear como un problema de este tipo.

5.2. Primer algoritmo para el *(TOP)*

Utilizando el artículo [10] escrito por Duc-Cuong Dang, Racha El-Hajj y Aziz Moukrim, presentaremos una aplicación del *Team Orienteering Problem* diferente a las vistas anteriormente. El planteamiento de esta nueva versión consiste en una flota de m vehículos, un conjunto de n clientes y unos beneficios asociados a cada cliente i de P_i. El objetivo es determinar un recorrido para cada vehículo de tal forma que se maximice el beneficio obtenido por la flota (F), con la restricción de que cada vehículo no puede superar un tiempo máximo de viaje T, siendo este tiempo máximo el mismo para todos los vehículos.

Utilizaremos el grafo $G = (V, E)$ para modelar el problema. En este grafo los vértices son el conjunto $V = \{1, ..., n\} \cup \{d, a\}$, donde se encuentran los n clientes y los puntos de salida (d) y llegada (a). Utilizaremos la siguiente notación:

$$V^- = V \setminus \{a, d\} \ ; \ V^d = V \setminus \{a\} \ ; \ V^a = V \setminus \{d\}$$

El conjunto de arcos es el conjunto $E = \{(i, j) : i, j \in V\}$. Cada vértice $i \in V^-$ está asociado a un beneficio P_i, así como cada arco $(i, j) \in E$ tiene asociado un coste de tiempo $c_{i,j}$, con excepción de $c_{i,d} = c_{a,i} = \infty \, , \forall i \in V^-$, puesto que ninguna ruta puede pasar dos veces por el punto de salida o de llegada. Estos costes satisfacen la desigualdad triangular, es decir, para llegar de i a j el menor coste se alcanza tomando el arco (i, j).

Formularemos el problema tomando las siguientes variables binarias:

- $x_{ijr} = 1$ si el vehículo $r \in F$ utiliza el camino (i, j) para visitar al cliente j desde el cliente i, $x_{ijr} = 0$ en otro caso.
- $y_{ir} = 1$ si el cliente i es atendido por el vehículo r, $y_{ir} = 0$ en otro caso.

La función objetivo busca maximizar los beneficios obtenidos al visitar a cada cliente, por lo que es la siguiente:

$$max \sum_{i \in V^-} \sum_{r \in F} p_i \, y_{ir}$$

Sujeta a las siguientes restricciones:

$$\sum_{r \in F} y_{ir} \leq 1 \ , \ \forall i \in V^- \quad (1)$$

$$\sum_{j \in V^a} x_{djr} = \sum_{j \in V^d} x_{jar} = 1 \ , \forall r \in F \quad (2)$$

$$\sum_{i \in V^d \setminus \{k\}} x_{ikr} = \sum_{j \in V^a \setminus \{k\}} x_{kjr} = y_{kr} \ , \forall k \in V^- \, \forall r \in F \quad (3)$$

$$\sum_{i \in V^d} \sum_{j \in V^a \setminus \{i\}} c_{ij} x_{ijr} \leq T \ , \forall r \in F \quad (4)$$

$$\sum_{(i,j) \in U \times U} x_{ijr} \leq |U| - 1 \ \ \forall U \subseteq V^-, \, |U| \geq 2, \forall r \in F \quad (5)$$

$$x_{ijr} \in \{0, 1\} \ \ \forall i \in V, \forall j \in V, \forall r \in F \ ; \ y_{ir} \in \{0, 1\} \ \forall i \in V^-, \forall r \in F \quad (6)$$

La restricción (1) garantiza que cada cliente es visitado una vez como máximo. La restricción (2) garantiza que todos los vehículos salen del punto de partida d y llegan al punto final a. La restricción (3) garantiza que todas las rutas están bien definidas: si el vehículo r visita el cliente k, entonces existe un único vértice $i \in V^d$ de donde proviene el vehículo r y un único vértice $j \in V^a$ al que se dirige.

La restricción número (4) establece que el tiempo empleado en la ruta de cada vehículo no puede superar el tiempo máximo T. La restricción (5) está relacionada con los ciclos, los cuales no se pueden producir ya que en este caso el vehículo r estaría visitando dos veces al mismo cliente. Por último aparecen las restricciones por las cuales las variables de decisión deben tomar valores 0 o 1. Partiendo de esta formulación podemos añadir restricciones y añadir planos de corte que nos permitan obtener la solución de nuestro problema en un menor tiempo. Presentaremos en primer lugar dos tipos de restricciones que nos ayudan a fortalecer la formulación inicial:

Symetric Breaking

Si tenemos en cuenta la formulación de nuestro problema, no existe restricción alguna referida a qué vehículo en particular debe realizar una ruta; es decir, el valor de la función objetivo $\sum_{i \in V^-} \sum_{r \in F} p_i \, y_{ir}$ es el mismo una vez que se han determinado las m rutas óptimas si intercambiamos los vehículos que realizan cada ruta. Por ello, pueden existir numerosas soluciones tales que la función objetivo alcanza en todas ellas el mismo valor. Para eliminar esta simetría en las soluciones podemos ordenar las rutas en orden decreciente en relación al beneficio que obtienen, llegando a la siguiente restricción:

$$\sum_{i \in V^-} y_{i,r+1} P_i - \sum_{i \in V^-} y_{i,r} P_i \leq 0 \, , \, \forall r \in F \setminus \{m\} \quad (7)$$

Acotaciones en los beneficios y en el número de clientes

Continuando con la idea de añadir restricciones a nuestro problema que nos permitan acotar en gran medida la región factible de soluciones, en este apartado proponemos 4 restricciones relacionadas tanto con acotaciones en los beneficios como con acotaciones en el número de clientes.

Observación 5.1. *Dado un caso X del TOP, entre los 387 posibles propuestos en el artículo [9], con m vehículos disponibles, denotaremos con X^n el caso X en el cual hemos reducido el número de vehículos a n, con $n \leq m$.*

Las restricciones referidas a los beneficios en las acotaciones son las siguientes:

$$\sum_{r \in H} \sum_{i \in V^-} y_{i,r} P_i \leq UB(X^{|H|}), \, \forall H \subset F \quad (8)$$

$$\sum_{r \in H} \sum_{i \in V^-} y_{i,r} P_i + UB(X^{m-|H|}) \geq LB(X), \, \forall H \subseteq F \quad (9)$$

El valor de la cota $UB(X^{|H|})$ que aparece en la primera ecuación se calcula utilizando un razonamiento similar a la *programación dinámica*. La programación dinámica es un método que consiste en simplificar un problema de programación matemática complejo en subproblemas más simples, de manera recursiva, de forma que, resolviendo estos últimos, podamos hallar una solución óptima para el problema original. Aplicando esta idea

calcularemos en primer lugar una cota superior de los beneficios que se pueden alcanzar en el caso en el que solo disponemos de un vehículo, es decir, con $\mid H \mid = 1$. Para obtener cotas de una manera rápida podemos establecer una condición de parada en la resolución de estos problemas, como puede ser el tiempo de computación o el número de nodos estudiados en un árbol de soluciones generado por un algoritmo de *Branch and Bound*. Una vez que se ha obtenido la cota superior para cada caso X concreto y con el número de vehículos disponible igual a 1, podemos usar estas acotaciones para resolver los casos en los que el número de vehículos aumenta.

En la segunda ecuación aparecen los valores $LB(X)$, es decir, acotaciones inferiores en los beneficios de cada caso X. Para calcular estas acotaciones inferiores podemos estudiar una solución factible de nuestro problema TOP, la cual se puede obtener aplicando métodos heurísticos. Existen numerosos métodos heurísticos que permiten obtener soluciones factibles para nuestro problema, y explicaremos uno de ellos brevemente en el apartado correspondiente a este tipo de métodos para resolver este problema.

Además de las acotaciones en los beneficios también utilizaremos las acotaciones en el número de clientes. Para ello modificaremos el problema X por el problema X_I, en el cual cada cliente representa un beneficio unitario, por lo que la suma de los beneficios obtenidos por visitar a cada cliente representa el número de clientes que han sido visitados. Las acotaciones utilizadas son las siguientes:

$$\sum_{r \in H} \sum_{i \in V^-} y_{i,r} \leq UB(X_I^{|H|}), \forall H \subset F \quad (10)$$

$$\sum_{i \in V^-} y_{i,r} \geq LB(\overline{X}_I^1), \forall r \in F \quad (11)$$

El valor de la cota superior $UB(X_I^{|H|})$ se calcula de una manera similar al explicado anteriormente para la acotación en los beneficios, mientras que $LB(\overline{X}_I^1)$ denota una acotación inferior obtenida de resolver el problema X_I^1 que pretende minimizar la misma función objetivo que en el problema original buscábamos maximizar. Además en este nuevo problema añadimos las restricciones relacionadas con las acotaciones en los beneficios vistas previamente, con $\mid H \mid = 1$.

Restricciones de Eliminación de Ciclos

Las restricciones del tipo (5) relacionadas con la presencia de ciclos en las rutas crecen de manera exponencial con el número de vértices del problema, pues se toman subconjuntos de V^-. Por ello, en la práctica no es útil hacer uso de estas restricciones. Esta familia de restricciones es reemplazada por otra familia de restricciones llamada *Restricciones de Eliminación de Ciclos Generalizadas*.

Esta familia de restricciones es más fuerte y utiliza en su definición los siguientes conjuntos relacionados con un conjunto de vértices S:

- $\delta(S)$ se define como el conjunto de arcos que unen vértices de S con vértices de $V \setminus S$.
- $\gamma(S)$ se define como el conjunto de arcos que unen vértices de S.

La siguiente restricción asegura que cada cliente i visitado por el coche r está conectado con los puntos de salida y llegada:

$$\sum_{(u,v)\in\delta(S)} x_{uvr} \geq 2y_{ir}\,,\ \forall S \subset V\,,\ \{d,a\} \subseteq S, \forall i \in V \setminus S, \forall r \in F \quad (12)$$

Observación 5.2. *En el caso que $y_{i,r} = 0$ la desigualdad es cierta pues $x_{u,v,r} \geq 0$. En el caso que $y_{i,r} = 1$, puesto que $i \notin S$ y $\{d,a\} \subseteq S$, para llegar al vértice i hay que tomar al menos dos veces un camino $(u,v) \in \delta(S)$: una de ellas la primera vez que desde el punto de salida visitamos un cliente $j \notin S$ y la segunda el camino que tomamos desde el último cliente $k \notin S$ para volver al conjunto S, pues el punto de llegada $a \in S$. De esta forma, aseguramos que cada cliente visitado está conectado con los puntos de salida y llegada y por tanto que no existen ciclos.*

Las siguientes dos restricciones son equivalentes a la restricción (12):

$$\sum_{(u,v)\in\gamma(S)} x_{uvr} \leq \sum_{i\in S} y_{ir} - y_{jr}\,,\ \forall S \subset V\,,\ \{d,a\} \subseteq S, \forall j \in V \setminus S, \forall r \in F \quad (13)$$

$$\sum_{(u,v)\in\gamma(U)} x_{uvr} \leq \sum_{i\in U} y_{ir} - y_{jr}\,,\ \forall U \subseteq V^-\,,\ \forall j \in U\,,\ \forall r \in F \quad (14)$$

Incompatibilidades y *clique cuts*

Comenzaremos tomando un subconjunto $S \subset V^-$ y denotando por *Minlength(S)* el tiempo que se tarda en recorrer el camino más rápido desde el depósito **d** hasta el **a** pasando por todos los vértices contenidos en S. De manera análoga podemos tomar un subconjunto de arcos $R \subset E$ y denotar por *Minlength(R)* el tiempo que se tarda en recorrer la ruta más rápida que contiene todos los arcos de R. Utilizando ambas ideas presentaremos el *grafo de incompatibilidades* entre vértices y entre arcos.

- *Grafo de incompatibilidades entre vértices*: $G^{Inc}_{V^-} = (V^-, E^{Inc}_{V^-})$, donde el conjunto $E^{Inc}_{V^-}$ se define de la siguiente forma:

$$E^{Inc}_{V^-} = \{(i,j) \in E,\, i,j \in V^-\,;\, Minlength(\{i,j\}) > T\}$$

Observación 5.3. *Este grafo contiene todos los vértices de V^- (todos los clientes de nuestro problema) y todos los arcos que unen dos puntos los cuales no pueden ser visitados en una misma ruta, pues la ruta más rápida que pasa por esos dos puntos y va desde **d** hasta **a** supone un mayor tiempo que el tiempo máximo T.*

- *Grafo de incompatibilidades entre arcos*: $G^{Inc}_{E} = (E, E^{Inc}_{E})$ donde el conjunto E^{Inc}_{E} se define de la siguiente forma:

$$E^{Inc}_{E} = \{[i,j)];\, i = (u,v) \in E, j = (w,s) \in E\,;\, Minlength(\{(u,v),(w,s)\}) > T\}$$

Observación 5.4. *Este grafo contiene todos los arcos de E y están unidos por un nuevo arco de G^{Inc}_{E} aquellos tales que ambos no pueden ser visitados en una misma ruta, pues la ruta más rápida que pasa por ambos arcos y va desde **d** hasta **a** supone un mayor tiempo que el tiempo máximo T.*

Una vez que hemos definido estos grafos de incompatibilidades podemos estudiar dos tipos de restricciones que utilizaremos como planos de corte en el algoritmo de *Branch and Cut*. Para ello definiremos antes el siguiente concepto:

Definición 5.5. *(Clique)* Dado un grafo $G = (V, E)$, decimos que $K = (\overline{K}, \overline{E})$ es un *clique* de este grafo si $\overline{K}$ es un subconjunto de vértices de V tal que todo vértice $i, j \in \overline{K}$, $i \neq j$ está unido por un arco $(\exists (i, j) \in \overline{E})$.

Observación 5.6. *Un clique K es por tanto un subgrafo del grafo original G en el que cada vértice está conectado a todos los demás vértices del subgrafo. Hablaremos de clique maximal para referirnos a un clique que no puede ser extendido añadiendo ningún vértice.*

Podemos definir por tanto un clique K en el grafo $G^{Inc}_{V^-}$ y un clique Q en el grafo G^{Inc}_{E}. Utilizando ambos subconjuntos se pueden estudiar las siguientes restricciones:

$$\sum_{i \in K} y_{ir} \leq 1\,,\ \forall r \in F \quad (15) \quad ; \quad \sum_{(u,v) \in Q} x_{uvr} \leq 1\,,\ \forall r \in F \quad (16)$$

En el caso de la restricción (1), si tomamos K un clique de $G^{Inc}_{V^-}$, entonces no pueden existir dos vértices i, j tales que pertenezcan al clique, pues en eso caso ambos están conectados por un arco $(i, j) \in E^{Inc}_{V^-}$, el cual por definición del grafo de incompatibilidades entre vértices no puede aparecer en una ruta. La segunda restricción se puede explicar de una manera muy similar, ya que si existen más de dos arcos en el clique, entonces existe un *arco* de G^{Inc}_{E} entre ellos, lo que implica que ambos no pueden ser visitados en una misma ruta.

Los grafos $G^{Inc}_{V^-}$ y G^{Inc}_{E} pueden ser estudiados antes de iniciar el algoritmo, y se pueden guardar para cada caso en función del número de nodos y el número de vehículos. Una vez que se han determinado estos grafos podemos proceder a una descomposición de ambos grafos en cliques maximales, utilizando una estrategia denominada *greedy decomposition*. Una descomposición de este tipo se obtiene determinando cliques maximales de un grafo y eliminando sucesivamente los vértices y los arcos pertenecientes a estos cliques hasta que el grafo sea vacío.

Ejemplo 5.1. *El siguiente grafo G tiene vértices $\{A, B, C, D, E, F\}$ y se pueden encontrar tres cliques maximales: $\{A, B, F\}$,$\{C, D, E\}$ y $\{B, C, E, F\}$.*

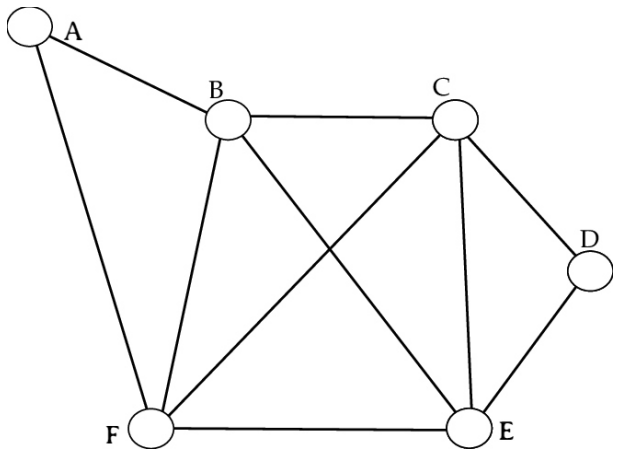

Figura 5.1: Grafo de ejemplo para el estudio de cliques maximales

5.2.1. Implementación del método y análisis de los resultados

Una vez presentada la formulación clásica del problema, las restricciones que se pueden añadir para mejorar esta formulación inicial y los planos de corte que utilizaremos, en esta sección comentaremos como aplicamos el método de *Branch and Cut*, prestando atención a todos aquellos factores a tener en cuenta a la hora de aplicar este método que comentamos en el Capítulo 4.

Estrategia de ramificación

La función objetivo maximiza los beneficios obtenidos acudiendo a cada cliente, por lo que la elección de los clientes que aporten un mayor beneficio resulta muy importante a la hora de maximizar este beneficio. Por tanto, la estrategia de ramificación priorizará las variables y_{ir} en detrimento de las variables $x_{i,j,r}$, las cuales guardan una mayor relación con el orden en el que se visitan los clientes.

Preprocesamiento

En primer lugar debemos determinar aquellos clientes que sean *inaccesibles*, es decir, aquellos clientes tales que la ruta desde el punto d hasta el punto a pasando solo por el vértice i^* (asociado al cliente en cuestión) suponga un mayor tiempo que el tiempo máximo T. En este caso podemos fijar todas las variables $y_{i^*,r} = 0 \; \forall r \in F$, $x_{j,i^*,r} = x_{i^*,k,r} = 0 \; \forall j,k \in V \setminus \{i^*\}$, $\forall r \in F$, pues ninguna ruta puede incluir a ese cliente. Al igual que con los clientes, también podemos determinar aquellos arcos que sean inaccesibles; es decir aquellos arcos $(i,j) \in E$ tales que $c_{i,j} > T$, y fijar todas las variables $x_{i,j,r} = x_{j,i,r} = 0 \; \forall r \in F$ y para todo $(i,j) \in E$ tales que sean inaccesibles.

Métodos heurísticos

Antes de comenzar a ejecutar el método utilizamos un método heurístico recogido en el artículo [9] para obtener una primera solución factible que nos permite acelerar la resolución del problema, pues podemos eliminar todas aquellas posibles rutas que generen unos beneficios inferiores. Este método heurístico consta de dos pasos: un primer paso de inicialización y un segundo paso de mejora.

El primer paso de inicialización comienza construyendo una elipse sobre el conjunto de posibles clientes utilizando el punto de salida y el de llegada como focos de la elipse y el tiempo máximo T^{max} como longitud del eje mayor de la elipse. Todos aquellos puntos que se encuentren fuera de la elipse no los consideraremos, ya que una ruta que contenga alguno de esos puntos no satisfará la restricción del tiempo máximo. Una vez que hemos determinado los N puntos que se encuentran dentro de esta elipse debemos escoger los L puntos que se encuentren más alejados del punto de salida y de llegada, siendo $L = min(5, N)$. Estos L puntos son los candidatos iniciales que debemos asignar a cada una de las M rutas, con $M \in \{2,3,4\}$ como posible cantidad de vehículos disponibles.

Observación 5.7. *En el caso que $L \leq M$ entonces $L = N$, y por tanto hay como máximo tantos clientes que se pueden visitar como vehículos disponibles, por lo que cada ruta incluye a uno y solo uno de estos clientes. Puesto que este caso carece de interés, estudiaremos el caso general donde $L > M$.*

En el caso general tomamos M de estos L puntos como primer cliente a visitar en cada una de las rutas y el resto de puntos se van asignando con el criterio que cada cliente que se añada a la ruta sea el que repercute en un tiempo de viaje menor, buscando de esta manera llegar a una mayor cantidad de clientes. Si una vez que las rutas están completas (no se puede añadir ningún cliente más pues no satisface la restricción de T^{max}) quedan aún clientes que no han sido visitados se generan nuevas rutas hasta que todos los clientes pertenezcan a una de las posibles rutas, tomando como solución inicial las M rutas cuya suma total de beneficios sea mayor.

Observación 5.8. *El conjunto de las rutas que pertenecen a la solución inicial lo denominaremos como R, mientras que el conjunto de aquellas que no han sido escogidas lo llamaremos R_N.*

Una vez que tenemos una solución inicial con las M rutas que proporcionaban un mayor beneficio, intentaremos mejorar la solución inicial realizando un intercambio de dos puntos: un punto i perteneciente a una de las rutas de R_N se incluye en una de las rutas de R; y al contrario, un punto j perteneciente a una de las rutas de R se incluye en una de las rutas de R_N. Este cambio se hace simultáneamente de tal forma que el tiempo empleado de la ruta disminuya o, en el caso de crecer, crezca lo menos posible. Para estudiar este tiempo empleado en cada ruta haremos uso de la siguiente expresión, siendo $T(p)$ el tiempo empleado en la ruta p donde queremos incluir el punto i.

$$T(p) - (c_{j,j+1} + c_{j-1,j} - c_{j-1,j+1}) + min\ _{k \in p,\, k \neq 1,j}(c_{i,k} + c_{k-1,i} - c_{k-1,k}) \qquad (5.1)$$

Observación 5.9. *En la expresión $j-1$ y $j+1$ hacen referencia al punto visitado anterior y posteriormente, respectivamente, del cliente j. En la expresión aparece el mínimo entre todos los clientes k visitados en p de tal forma que se introduzca el nodo i en la ruta generando el menor impacto posible en cuanto a tiempo empleado.*

Utilizando esta expresión podemos comprobar si la introducción del nodo i en la ruta p es o no factible, teniendo en cuenta que no será factible si el valor que toma la expresión 5.1 es mayor que T. En el caso que se pueda insertar el nodo i en la ruta p pero el nodo j no se pueda insertar en ninguna de las rutas de R_N, crearemos una nueva ruta que solo incluya el punto de salida, el de llegada y el nodo j.

Una vez que hemos comprobado si podemos insertar el nodo i en la ruta p debemos estudiar el impacto que genera este intercambio en el beneficio asociado a la ruta. Si el intercambio genera un mayor beneficio, el intercambio se realiza inmediatamente. En el caso que el intercambio de ningún nodo de ninguna ruta de R_N genera un mayor beneficio, podemos considerar nodos que impliquen un decrecimiento de los beneficios en una pequeña cantidad, manteniendo el beneficio siempre por encima de un umbral.

Además de realizar un intercambio entre dos puntos, también podemos tomar un punto que se encuentre en una ruta y añadirlo a otra ruta siempre y cuando sea factible (no se supere el T) y genere un aumento de los beneficios. Estos intercambios se pueden realizar entre nodos pertenecientes a las rutas en R_N y R, e incluso entre rutas de R. El objetivo de esta estrategia es encontrar las rutas con el mayor beneficio posible. Por último, es recomendable estudiar cada una de las rutas de R con el objetivo de visitar los diferentes clientes de la forma más rápida y de esta forma reducir el tiempo empleado en cada ruta de tal manera que podamos introducir en la ruta algún cliente más. Este método nos permite obtener una primera solución factible formadas por las M rutas con mayores beneficios.

Ejecución del algoritmo

Utilizando este método heurístico podemos conseguir una primera solución factible de gran calidad que nos permita disponer de una buena cota inferior para nuestro problema. Partiendo de esta solución factible, resolvemos el problema de programación lineal relajado que incluye las restricciones $(1) - (4)$ y (6), junto a la restricción de *Symetric Breaking* (7), las acotaciones en los beneficios y en el número de clientes $(8) - (11)$; y las desigualdades (15) relacionadas con los *cliques*. Una vez que hemos obtenido esta solución, debemos comprobar que esta solución no contiene ningún subciclo. En el caso que esto ocurra, debemos añadir las *Restricciones de Eliminación de Ciclos* $(12) - (14)$ asociadas al conjunto de vértices S resultantes de la solución obtenida. Además de añadir estas restricciones, también podemos estudiar el conjunto de vértices y arcos resultantes de la solución, extrayendo los subgrafos $G_{V^-}^{Inc}$ y G_E^{Inc} y estudiando la descomposición en cliques que nos permite añadir las correspondientes restricciones (15) y (16) al nuevo modelo. En la siguiente iteración se repite el mismo proceso pero con estas nuevas restricciones añadidas.

En el artículo [9] se recogen los 387 casos propuestos para el *Team Orienteering Problem*, divididos en 7 conjuntos en función del número de clientes, variando estos desde 21 hasta 102. Fijado el número de clientes dependiendo del conjunto en el que nos encontremos, los diferentes casos varían atendiendo al número de vehículos (de 2 a 4), y al tiempo máximo de la duración de cada ruta (T). Esta información está recogida en la tabla 5.1, cuyas columnas indican el conjunto en el que nos encontramos, el número de casos de cada conjunto, el número de clientes, el número de vehículos y el rango de tiempo máximo de cada ruta.

Conjunto	Cardinal	Número de clientes	Número de Vehículos	T^{max}
1	54	32	2-4	3.8-22.5
2	33	21	2-4	1.2-42.5
3	60	33	2-4	3.8-55.0
4	60	100	2-4	3.8-40.0
5	78	66	2-4	1.2-65.0
6	42	64	2-4	5.0-200.0
7	60	102	2-4	12.5-120.0

Tabla 5.1: Benchmark del *Team Orienteering Problem*

Aplicando la formulación vista a cada una de estos casos, podemos ver en la siguiente tabla la mejora en el número de casos resueltos y en el tiempo de CPU utilizada a medida que añadimos las diferentes restricciones.

Restricciones utilizadas	Casos resueltos	Tiempo de CPU (en segundos)
Formulación inicial	177/387	198.9
+Restricciones de eliminación de ciclos	233/387	61.75
+Symmetric Breaking	243/387	14.32
+Acotaciones en beneficios/clientes	265/387	13.16
+Cliques	278/387	3.24

Tabla 5.2: Resultados de la ejecución del algoritmo

Ambas tablas las podemos encontrar en el artículo [10].

5.3. Segundo algoritmo para el *(TOP)*

En este apartado presentaremos una formulación diferente a la vista anteriormente para resolver el mismo problema, pero utilizando tanto variables discretas como variables continuas.

Observación 5.10. *Al igual que antes utilizaremos el grafo $G = (V, E)$ para modelar el problema, siendo ahora el conjunto de vértices $V = \{0, 1, ..., n, n+1\}$, donde 0 hace referencia al punto de salida y $n+1$ al de llegada. Denotaremos con $N = \{1, ..., n\}$ el conjunto de clientes.*

Los coeficientes p_i hacen referencia al beneficio asociado a cada cliente, con $p_0 = p_{n+1} = 0$; y los coeficientes $t_{i,j}$ representan el tiempo de viaje desde el vértice i hasta el vértice j. Las variables de decisión empleadas son:

- x_{ij}: son variables binarias que toman el valor 1 si usamos el camino $(i, j) \in A$ desde el cliente i hasta el j.
- y_i: variable binaria que toma el valor 1 si el cliente $i \in N$ es visitado.
- $z_{i,j}$ con $(i, j) \in A \setminus \{0, n+1\}$ es una variable continua que representa el tiempo acumulado desde el inicio hasta llegar al vértice j de un vehículo proveniente del vértice i.

Observación 5.11. *En el caso que $x_{i,j} = 1$, entonces $z_{i,j}$ denota el tiempo de llegada al vértice j; mientras que si $x_{i,j} = 0$, entonces $z_{i,j} = 0$. De esta manera, aunque las variables no distingan el vehículo que realiza cada ruta, la condición de que ningún vehículo puede pasar dos veces por el mismo vértice implica que las variables continuas $z_{i,j}$ van almacenando el tiempo empleado en cada ruta. Podemos reconstruir cada ruta partiendo del primer cliente visitado observando el valor que toman las variables $x_{i,j}$.*

En este caso, la función objetivo es:

$$max \sum_{i \in N} p_i y_i$$

Las restricciones que se deben cumplir son las siguientes:

$$\sum_{j \in N} x_{0j} = \sum_{i \in N} x_{i,n+1} = m \quad (1)$$

$$\sum_{(j,i) \in \delta^-(i)} x_{j,i} = \sum_{(i,j) \in \delta^+(i)} x_{i,j} = y_i \ , \ \forall i \in N \quad (2)$$

$$z_{0,j} = t_{0,j} x_{0,j} \ , \ \forall j \in N \quad (3)$$

$$\sum_{(i,j) \in \delta^+(i)} z_{i,j} - \sum_{(j,i) \in \delta^-(i)} z_{j,i} = \sum_{(i,j) \in \delta^+(i)} t_{i,j} x_{i,j} \ , \ \forall i \in N \quad (4)$$

$$z_{i,j} \leq T^{max}_{j,n+1} = T^{max} - t_{j,n+1} \ , \ \forall (i, j) \in A \setminus \{(0, n+1)\} \quad (5)$$

$$z_{i,j} \geq t^0_{i,j} x_{i,j} = (t_{0,i} + t_{i,j}) x_{i,j} \ , \ \forall (i, j) \in A \setminus \{(0, n+1)\} \quad (6)$$

$$x_{i,j} \in \{0, 1\} \ , \ \forall (i, j) \in A \setminus \{(0, n+1)\} \ ; \quad y_i \in \{0, 1\} \ \forall i \in N \quad (7)$$

Observación 5.12. *En las restricciones número* (2) *y* (4) *aparecen* $\delta^-(i)$ *y* $\delta^+(i)$*. El primero de ellos,* $\delta^-(i)$*, hace referencia al conjunto de arcos que llegan al vértice* i*, mientras que el segundo,* $\delta^+(i)$*, se refiere al conjunto de arcos que salen desde el punto* i*.*

Observación 5.13. *En las restricciones* (5) *y* (6) *aparecen dos nuevos coeficientes:*

$$T_{j,n+1}^{max} = T^{max} - t_{j,n+1} \quad y \quad t_{i,j}^0 = t_{0,i} + t_{i,j}$$

El primero hace referencia al tiempo máximo que se puede tardar en llegar desde el punto de salida hasta el vértice j *pasando por el cliente* i *en la visita anterior, pues se define como la diferencia entre el límite de tiempo de cada vehículo* T^{max} *y el tiempo que se tarda en llegar desde el vértice* j *hasta el depósito de llegada. El segundo hace referencia al tiempo mínimo empleado para llegar desde el punto de salida hasta el vértice* j *pasando en la visita anterior por el vértice* i*.*

Las primeras restricciones, (1), se encargan de asegurar que los m vehículos salen del depósito y llegan al punto de destino. Las restricciones número (2) aseguran que si un cliente i es visitado, entonces a ese vértice se llega y se sale de él solo una vez (por un único vehículo). Las restricciones (3) establecen que el tiempo empleado para llegar desde el punto de partida hasta el primer cliente que se visite es el tiempo equivalente a recorrer el arco $(0, j)$. La igualdad número (4) se encarga de actualizar el tiempo empleado en la ruta una vez que un cliente es visitado.

Observación 5.14. *En la igualdad número* (4)*, una vez que fijamos un cliente* i^**, existe un único cliente* j^* *tal que es el próximo en ser visitado, por tanto, el tiempo empleado en ir del cliente* i^* *al cliente* j^* *(parte derecha de la igualdad) debe ser igual a la diferencia entre el tiempo empleado para llegar desde el inicio al cliente* j^* *y el tiempo empleado para llegar desde el inicio al cliente* i^**.*

La restricción número (5) se encarga de establecer el límite de tiempo que se puede emplear en cada ruta, como comentamos en la observación 5.13, así como la número (6) establece una acotación inferior a los valores que pueden tomar las variables $z_{i,j}$ que nos permite restringir el conjunto de soluciones factibles. Por último, podemos añadir una restricción más referida a la duración total de las rutas, la cual por su definición es una desigualdad válida para todas las soluciones del problema:

$$\sum_{(i,j)\in A\setminus\{(0,n+1)\}} t_{i,j}x_{i,j} \leq mT^{max} \quad (8)$$

Restricciones de conectividad (CC's)

Una vez que hemos planteado el problema con las nuevas variables de decisión y hemos expuesto las restricciones necesarias que determinan la región factible de soluciones, estudiaremos un tipo de restricciones que añadiremos como planos de corte a lo largo del árbol de soluciones que se genera aplicando el método de *Branch and Bound*. Este tipo de restricciones se denominan *restricciones de conectividad* (*CC's* es su abreviatura en inglés) y son del siguiente tipo:

$$\sum_{(i,j)\in\delta^+(S)} x_{i,j} \geq y_h,\ S \subseteq N, \mid S \mid \geq 2,\ h \in S \quad (5.2)$$

Podemos observar por su naturaleza que el número de este tipo de restricciones crece exponencialmente a medida que crece N, por lo que no es recomendable incluirlas en la formulación. Sin embargo, resultará muy útil su uso como planos de corte en el algoritmo de *Branch and Cut*, como veremos posteriormente.

Observación 5.15. *Estas restricciones se pueden usar como planos de corte pues son desigualdades válidas para cualquier solución factible de nuestro problema, algo que es sencillo de comprobar:*

- *Si $y_h = 0$, la desigualdad se cumple pues $x_{i,j} \geq 0 \; \forall (i,j) \in \delta^+(S)$, y por tanto $\sum_{(i,j)\in\delta^+(S)} x_{i,j} \geq 0 = y_h$.*
- *Si $y_h = 1$ con $h \in S$ el cliente h es visitado, lo que implica que en ese momento estamos dentro del conjunto de vértices S. Puesto que $S \subseteq N$ y el depósito de llegada no pertenece a N, existe al menos un arco que se toma para salir de S, es decir, $\sum_{(i,j)\in\delta^+(S)} x_{i,j} \geq 1 = y_h$, por lo que en este caso la desigualdad también es válida.*

Una vez que hemos visto que este tipo de restricciones son válidas para todas las soluciones factibles de nuestro problema, estudiaremos como podemos comprobar si una solución óptima del problema relajado eliminando las restricciones (7) no cumple alguna de estas restricciones, la cual añadiremos en ese caso como plano de corte. Para ello lo que haremos en primer lugar es tomar la solución óptima de ese problema relajado del tipo $(\overline{y}, \overline{x}, \overline{z})$ y considerar el grafo $\overline{G} = (V, \overline{A})$ inducido por la correspondiente solución. Esto quiere decir que el arco $(i,j) \in \overline{A}$ si la correspondiente variable $x_{i,j}$ toma un valor mayor que 0 en esa solución óptima ($\overline{x}_{i,j} > 0$). En este caso asociaremos a ese arco (i,j) una capacidad $c_{i,j} = \overline{x}_{i,j}$.

Partiendo de este grafo inducido y tomando la pareja de vértices $\{i, n+1\}$ con $i \in N$ plantearemos el problema de *máximo caudal, mínimo corte*, siendo i la fuente del caudal y $n+1$ el sumidero. Explicaremos a continuación en que consisten ambos problemas y cuál es su relación.

Problema del caudal máximo

Este problema se plantea utilizando un grafo $G = (V, A)$ y dos nodos $s, t \in V$, donde s es la fuente y t es el sumidero. Un *caudal-s,t* en este grafo consiste en un vector no negativo $x \in \mathbb{R}^{|A|}$ tal que cada variable x_e hace referencia al caudal que atraviesa el arco $e \in A$. Las restricciones a tener en cuenta en este problema son las siguientes:

- La cantidad de caudal que llega a cada vértice $v \in V \setminus \{s,t\}$ es la misma que el caudal que sale de ese vértice, por tanto:

$$\sum_{e\in\delta^-(v)} x_e - \sum_{e\in\delta^+(v)} x_e = 0, \forall v \in V \setminus \{s,t\} \tag{5.3}$$

- Puesto que el caudal se conserva al pasar por los vértices, la cantidad de caudal que sale de s es igual a la cantidad de caudal que llega a t, y a esta cantidad se le denomina el *valor del caudal s-t*:

$$\nu = val(x) = \sum_{e\in\delta^+(s)} x_e - \sum_{e\in\delta^-(s)} x_e \tag{5.4}$$

- Cada arco $e \in A$ tiene asociada una cierta capacidad c_e, por lo tanto un caudal es factible siempre y cuando el caudal que atraviesa el arco e, denotado por x_e, sea menor o igual que la capacidad de ese arco:

$$x_e \leq c_e \;\; \forall e \in A \tag{5.5}$$

Utilizando todas estas restricciones, el problema de máximo caudal para s y t busca maximizar el valor ν sujeto a estas restricciones vistas.

Problema del corte mínimo

Un problema estrechamente relacionado con este problema del caudal máximo es el problema del corte mínimo. Partiendo de nuevo de un grafo $G = (V, A)$ y de dos nodos $s, t \in V$, podemos definir los siguientes conceptos:

Definición 5.16. Un *corte-s,t* se define como un conjunto de arcos de la forma $\Gamma := \delta^+(S)$ donde $s \in S \subseteq V \setminus \{t\}$.

Definición 5.17. La capacidad del corte Γ se define como $c(\Gamma) := \sum_{e \in \delta^+(S)} c_e$.

El problema del corte mínimo consiste en encontrar un *corte-s,t* tal que su capacidad sea mínima, y la solución del problema es por tanto un conjunto $S \subseteq V \setminus \{t\}$ de vértices de V que satisfacen que la suma de capacidades de $e \in \delta^+(S)$ es mínima.

Utilizando que el problema de corte mínimo se puede formular como el dual del problema de caudal máximo, el teorema de *Ford-Fulkerson* nos permite afirmar que en cualquier red el caudal máximo que fluye de la fuente al sumidero es igual a la capacidad del corte mínimo que separa a la fuente del sumidero.

Teorema 5.1. *(Ford-Fulkerson, Caudal máximo-Corte mínimo): Dado un grafo $G = (V, A)$, dos nodos distintos $s, t \in V$, y capacidades no negativas c_e, $e \in A$, entonces:*

$$max\{val(x)\,;\, x\, caudal - s,t\ factible\} = min\{c(\Gamma) :\ \Gamma\ \ corte - s,t\}$$

Presentaremos un breve ejemplo que nos permite comprender mejor este tipo de problemas:

Ejemplo 5.2. *Partiendo del siguiente grafo con fuente s, sumidero t, vértices A, B, C, D y las capacidades indicadas en cada arista, estudiaremos el problema del caudal máximo y el corte mínimo.*

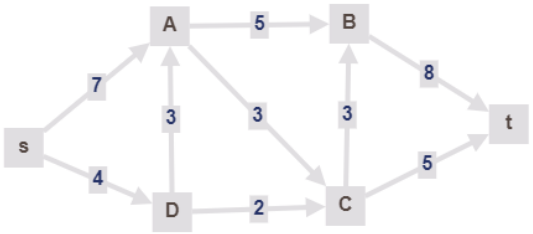

Figura 5.2: Grafo con fuente s, sumidero t y capacidades de cada arista especificadas

Teniendo en cuenta la capacidad de cada arista, podemos comprobar que el caudal máximo tiene un valor de $\nu = 10$.

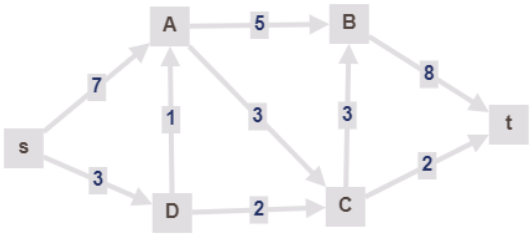

Figura 5.3: Grafo con fuente s, sumidero t y caudal que lleva cada arista

Por último, podemos comprobar que la capacidad del corte mínimo coincide con 10. El resultado del corte es el subconjunto de vértices $S = \{s, A, D\}$*, y los arcos pertenecientes a* $\delta^+(S)$ *son* $\{AB, AC, DC\}$.

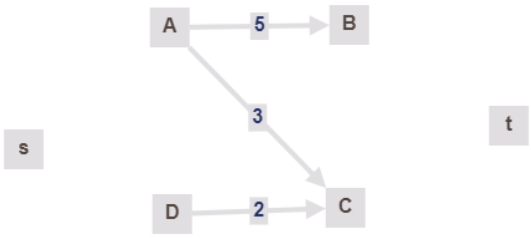

Figura 5.4: Grafo que representa el corte-s,t mínimo y la capacidad de las aristas pertenecientes a $\delta^+(S)$

Volviendo a nuestro problema original podemos utilizar esta relación entre ambos problemas para detectar si existe alguna restricción de conectividad tal que la solución obtenida $(\overline{y}, \overline{x}, \overline{z})$ no la cumpla. Para ello podemos estudiar el problema de mínimo corte siendo $i \in N$ la fuente y $n+1$ el sumidero.

El resultado de este problema consiste en un subconjunto S del conjunto de vértices tal que $S \subseteq V \setminus \{n+1\}$. Sabemos que la capacidad mínima de este corte se define como $\sum_{e \in \delta^+(S)} c_e$, y puesto que en el grafo $\overline{G}$ que estamos estudiando la capacidad de cada arista es $c_{i,j} = \overline{x}_{i,j}$, llegamos a la siguente expresión:$\sum_{(i,j) \in \delta^+(S)} \overline{x_{i,j}}$. Es decir, hemos llegado a la parte izquierda de la restricción de conectividad vista en la ecuación 5.2 sustituyendo las variables $x_{i,j}$ por el valor que toman en la solución óptima que estamos estudiando. Puesto que la capacidad mínima del corte coincide con el caudal máximo entre i y $n+1$, si encontramos un vértice $v \in S$ tal que $\overline{y}_v$ sea mayor que el máximo caudal, entonces hemos encontrado un subconjunto S y un vértice $v \in S$ tal que la restricción de conectividad no se satisface para la solución actual:

$$\sum_{(i,j) \in \delta^+(S)} \overline{x_{i,j}} < \overline{y_v} \;\; v \in S \tag{5.6}$$

De esta forma podemos añadir esta restricción de conectividad a la formulación inicial del problema y resolverlo de nuevo, encontrando en este caso una solución óptima que difiere de la encontrado en la iteración anterior pues la solución previa no satisface la restricción añadida. En el caso que existe más de un vértice $v \in S$ tal que se verifique la ecuación 5.6, tomaremos aquel $v \in S$ asociado al y_v con un valor mayor.

5.3.1. Implementación del método y análisis de los resultados

Utilizando las restricciones lineales (1)-(6) vistas podemos resolver el problema de maximizar los beneficios obtenidos en las diferentes rutas, obteniendo una solución óptima de la forma $(\overline{y}, \overline{x}, \overline{z})$. Si esta solución no es entera en las variables $(y_i, x_{i,j})$, podemos seguir los pasos explicados en el apartado anterior: estudiar el grafo inducido por la solución óptima y buscar una restricción de conectividad que funcione como plano de corte de tal forma que la solución óptima encontrada no satisfaga esa restricción. Existen dos opciones:

- En el caso de encontrarla, podemos añadir esta restricción a nuestro problema y volver a resolverlo con la nueva restricción añadida.
- En el caso de no encontrarla, puesto que la solución encontrada no es factible pues no cumple las restricciones enteras, existe al menos una variable y_i o $x_{i,j}$ que toma un valor diferente a 0 ó 1; por lo tanto, podemos ramificar el problema fijando el valor de esa variable como 0 y como 1, y volver a resolver el problema con esta variable fijada.

Siguiendo estos pasos podemos ir desarrollando el árbol de soluciones hasta que no existan nodos a estudiar, y en este caso habremos encontrado la solución óptima. Al igual que en el caso anterior, los experimentos son hechos sobre los 387 casos que hemos detallado en la tabla 5.1.

La siguienta tabla compara el tiempo medio de computación (en segundos) y los nodos estudiados en el caso de utilizar los cortes relacionados con las restricciones de conectividad y en el caso de no utilizarlos. Las tres primeras columnas hacen referencia a los diferentes conjuntos en los que se dividen los 387 casos planteados, la segunda al número de vehículos disponibles y la tercera a la cantidad de problemas que pertenecen al conjunto donde nos encontramos con ese número de vehículos disponibles.

Las tres columnas siguientes indican el número de problemas resueltos, el tiempo medio de computación (en segundos) y el número de nodos medios estudiados en el caso de implementar el método de *Branch and Cut* utilizando las restricciones de conectividad como planos de corte. Las tres últimas columnas indican los mismos datos pero en este caso aplicando un algoritmo de *Branch and Bound* para la resolución del problema.

Los datos han sido tomados de las tablas 5.4 y 5.5, recogidas en el artículo [11] y las cuales se pueden consultar al final de esta sección. La tabla 5.4 recoge diferentes datos relacionados con la resolución del *Team Orienteering Problem* utilizando la formulación planteada y añadiendo los planos de corte. Los datos que hemos utilizado en la tabla que se muestra a continuación son una parte de los que recoge la tabla 5.4, pues esta también incluye el número de problemas en los que no se ha encontrado la solución óptima y el porcentaje medio de la diferencia entre la solución óptima del problema y la solución encontrada con este algoritmo. La tabla 5.5 recoge los mismos datos pero en el caso de la resolución del problema sin añadir las restricciones de conectividad.

La tabla es la siguiente:

			Con CC's			Sin CC's		
			Resueltos			No resueltos		
Conjunto	m	#	#	Tiempo	Nodos	#	Tiempo	Nodos
1	2	18	18	2.233	290.7	18	4.728	605.4
	3	18	18	2.967	1042.3	18	6.489	1335.9
	4	18	18	0.611	259.4	18	1.628	537.2
2	2	11	11	0.973	272.1	11	0.855	282.1
	3	11	11	0.064	17.9	11	0.009	4.2
	4	11	11	0.000	0.0	11	0.000	0.0
3	2	20	20	28.930	5014.2	20	66.380	6895.3
	3	20	19	469.758	71,156.9	20	787.605	35,486.1
	4	20	19	276.147	61471.2	19	198.621	48,443.3
4	2	20	17	1304.765	30,906.0	14	1927.436	45,082.1
	3	20	9	643.978	115,191.7	9	802.944	115,258.3
	4	20	9	282.322	125,023.1	9	327.222	112,327.5
5	2	26	24	648.763	59,728.4	25	409.544	87,440.7
	3	26	13	92.208	291,827.1	14	251.764	311,263.5
	4	26	17	487.365	307,584.4	17	471.153	272,844.4
6	2	14	11	405.091	75,377.9	12	118.417	141,742.6
	3	14	11	656.191	127,752.2	12	364.100	93,577.4
	4	14	14	48.186	12,613.0	14	30.250	7,160.4
7	2	20	19	1293.295	30,225.4	15	2178.367	75,520.9
	3	20	12	1018.058	137,497.6	11	846.218	148,221.8
	4	20	10	304.410	264,050.8	11	648.318	214,593.0
			Total	Media	Media	Total	Media	Media
			311	394.279	95,608.9	309	438.032	95,295.4

Tabla 5.3: Comparativa del tiempo de computación y la media de nodos estudiados

Observando los datos que ofrece la tabla podemos comprobar en primer lugar que la implementación del algoritmo de *Branch and Cut* permite encontrar la solución óptima de 311 de los 387 problemas planteados, mientras que en el caso de no utilizar los planos de corte este número se reduce a 309. Además del número total de casos en los que el algoritmo es capaz de encontrar la solución óptima, también existe una diferencia en el tiempo medio de computación: mientras que el algoritmo de *Branch and Cut* ofrece un tiempo medio de computación de $394.279s$, la alternativa del *Branch and Bound* repercute en un aumento de este tiempo de computación hasta una media de $438.032s$. Esta mejora en el tiempo de computación utilizando las restricciones de conectividad como planos de corte se puede observar en la mayoría de casos estudiados, a excepción de los casos pertenecientes al conjunto 6. En este conjunto si observamos una diferencia favorable al algoritmo que no utiliza las restricciones de conectividad frente al que sí las utiliza.

Finalmente, si atendemos al número medio de nodos estudiados podemos observar que la media es ligeramente más elevada en el caso del uso de las $CC's$, 95,608.9 frente a 95,295.4. No obstante, esta leve diferencia no implica que el primero sea un algoritmo más lento que el segundo, sino al contrario, como hemos visto en los datos referidos al tiempo de computación. En definitiva, esta tabla muestra la efectividad del algoritmo de *Branch and Cut* añadiendo las restricciones de conectividad como planos de corte frente a la aplicación de un algoritmo de *Branch and Bound* sin utilizar estos cortes.

			Con CC's					
			Resueltos		No resueltos			
Conjunto	m	C	C	Tiempo (seg)	C	gap ()	Nodos	Cortes
1	2	18	18	2.233	0	-	290.7	38.3
	3	18	18	2.967	0	-	1042.3	24.0
	4	18	18	0.611	0	-	259.4	14.1
2	2	11	11	0.973	0	-	272.1	22.4
	3	11	11	0.064	0	-	17.9	4.7
	4	11	11	0.000	0	-	0.0	0.2
3	2	20	20	28.930	0	-	5014.2	35.5
	3	20	19	469.758	1	2.88	71,156.9	23.9
	4	20	19	276.147	1	1.79	61471.2	29.4
4	2	20	17	1304.765	3	0.68	30,906.0	453.7
	3	20	9	643.978	11	1.56	115,191.7	283.2
	4	20	9	282.322	11	3.91	125,023.1	106.5
5	2	26	24	648.763	2	1.57	59,728.4	642.8
	3	26	13	92.208	13	3.60	291,827.1	350.1
	4	26	17	487.365	9	4.58	307,584.4	98.0
6	2	14	11	405.091	3	1.96	75,377.9	1904.7
	3	14	11	656.191	3	2.67	127,752.2	143.7
	4	14	14	48.186	0	-	12,613.0	9.4
7	2	20	19	1293.295	1	1.06	30,225.4	603.3
	3	20	12	1018.058	8	2.89	137,497.6	293.5
	4	20	10	304.410	10	2.99	264,050.8	192.7
			Total	Media	Total	Media	Media	Media
			311	394.279	76	2.97	95,608.9	256.0

Tabla 5.4: Resultados de la ejecución del segundo algoritmo utilizando CC's

			Sin CC's				
			Resueltos		No resueltos		
Conjunto	m	C	C	Tiempo (seg)	C	gap ()	Nodos
1	2	18	18	4.728	0	-	605.4
	3	18	18	6.489	0	-	1335.9
	4	18	18	1.628	0	-	537.2
2	2	11	11	0.855	0	-	282.1
	3	11	11	0.009	0	-	4.2
	4	11	11	0.000	0	-	0.0
3	2	20	20	66.380	0	-	6895.3
	3	20	20	787.605	0	-	35,486.1
	4	20	19	198.621	1	1.79	48,443.3
4	2	20	14	1927.436	6	1.74	45,082.1
	3	20	9	802.944	11	2.32	115,258.3
	4	20	9	327.222	11	4.49	112,327.5
5	2	26	25	409.544	1	0.96	87,440.7
	3	26	14	251.764	12	3.31	311,263.5
	4	26	17	471.153	9	4.13	272,844.4
6	2	14	12	118.417	2	1.58	141,742.6
	3	14	12	364.100	2	1.82	93,577.4
	4	14	14	30.250	0	-	7,160.4
7	2	20	15	2178.367	5	1.22	75,520.9
	3	20	11	846.218	9	2.86	148,221.8
	4	20	11	648.318	9	3.44	214,593.0
			Total	Media	Total	Media	Media
			309	438.032	78	3.01	95,295.4

Tabla 5.5: Resultados de la ejecución del segundo algoritmo utilizando CC's

Bibliografía

[1] George Nemhauser y Laurence A. Wolsey, *«Integer and Combinatorial Optimization»*,1988.

[2] Laurence A. Wolsey, *«Integer programming»*, segunda edición, 2020.

[3] Michele Conforti, Gérard Cornuéjols y Giacomo Zambelli, *«Integer programming»*, 2014.

[4] Frederick S. Hillier y Gerald J. Lieberman, *«Introducción a la investigación de operaciones»*, novena edición, 2010.

[5] Alexander Schrijver, *«Theory of linear and integer programming»*, 1998.

[6] John E. Mitchell, *«Integer Programming: Branch and Cut Algorithms»*,*Encyclopedia of Optimization.*

[7] Emilie Danna, Edward Rothberg y Claude Le Pape,*«Exploring relaxation induced neighborhoods to improve MIP solutions»*, 2004.

[8] Matteo Fischetti, Fred Glover y Andrea Lodi,*«The feasibility pump»*, 2004.

[9] I-Ming Chao,Bruce L. Golden y Edward A. Wasil, *«The team orienteering problem »*,*European Journal of Operational Research*,1996.

[10] Duc-Cuong Dang, Racha El-Hajj y Aziz Moukrim,*«A Branch-and-Cut Algorithm for Solving the Team Orienteering Problem»*, 2013.

[11] Nicola Bianchessi,Renata Mansini y M. Grazia Speranza, *«A branch-and-cut algorithm for the Team Orienteering Problem»*,2016.