Universidad Nacional Autónoma de México

Facultad de Ciencias

Programación entera: El problema de localización de plantas

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
ACTUARIO

PRESENTA:
MAURICIO LECANDA BRICEÑO

DIRECTOR DE TESIS:
DR. RICARDO STRAUSZ SANTIAGO

2013

1. Datos del alumno

Lecanda
Briceño
Mauricio
55 66 16 01
Universidad Nacional Autónoma de México
Facultad de Ciencias
302035195

2. Datos del tutor

Dr
Ricardo
Strausz
Santiago

3. Datos del Sinodal 1

Act
Germán
Valle
Trujillo

4. Datos del Sinodal 2

Act
Mauricio
Aguilar
González

5. Datos del Sinodal 3

M en C
Jesús Agustín
Cano
Garcés

6. Datos del Sinodal 4

M en I
María Isabel
Escalante
Membrillo

7. Datos del trabajo escrito

Programación Entera: Problema de localización de plantas
71 p
2013

Índice general

Agradecimientos

Esta tesis se la dedico con mucho amor a mis padres Mauricio Lecanda Terán y Patricia Elizabeth Briceño Duarte por haberme apoyado a lo largo de toda mi vida en los buenos y en los malos momentos, otorgándome día a día su gran sabiduría y su excelente educación y de quienes estoy muy orgulloso de ser su hijo, a mi hermano David Lecanda Briceño con el que compartí tantas cosas maravillosas en la niñez y en la adolescencia y que siempre hemos estado juntos para apoyarnos y a mi flaco Jorge Alberto Romero Mendoza que es el amor de mi vida, que me salvó del abismo y que ha estado conmigo durante todo este tiempo ayudándome con sus ánimos, su paciencia y su gran amor.

Agradezco a mis abuelitos Raúl, Fede, Yola y Pera que con amor crearon a las mejores familias en la que pude nacer, a todos mis tíos, tías y primos Lecanda y Briseño que tanto quiero y de quienes he recibido ayuda y consejos en todo momento y finalmente a mi suegra Betty Mendoza que me adoptó como un hijo más y que gracias a ella ahora con gran honor formo parte también de la familia Mendoza.

A mis queridos amigos Agustín, Daniel, Ángel, Zaira, Adriana y Hany con los que pasé los mejores años de mi carrera y que ahora se han convertido en mis hermanos.

Finalmente, quiero agradecer a mi asesor Ricardo Strausz Santiago, o mejor conocido como Dino que me ayudó enormemente durante casi 3 años a terminar mi tesis y a mis sinodales Germán Valle Trujillo, Agustín Cano Garcés, Mauricio Aguilar González y María Isabel Escalante Membrillo que con sus conocimientos y observaciones me ayudaron a enriquecer el contenido de la tesis.

Introducción

Optimizar es un tema que siempre está presente en la vida cotidiana, y es más necesario cuando se ve involucrado el dinero. Todos hemos escuchado de alguien que quiere *minimizar* gastos o *maximizar* ingresos y utilidades.

Este texto introduce al lector en el interesante mundo de la optimización, definiendo modelos y variables y mostrando los métodos existentes para optimizar, también enseña las dificultades encontradas cuando el problema que se quiere resolver obliga a utilizar variables *enteras*, tal es el caso del tema de la tesis *Problema de Localización de Plantas*, el cual busca la mejor manera de distribuir *plantas* para satisfacer la demanda de los clientes y al mismo tiempo minimizar los gastos generados.

El término *planta* se refiere a cualquier lugar que genere un producto, como por ejemplo una fábrica, o una agencia automovilística o bien también puede tratarse de un almacén o centro de distribución que en vez de generar un bien ofrece un servicio.

El *cliente* es cualquier persona, empresa o incluso una población que requiera del servicio que la planta ofrece y que está dispuesto a pagar por ello.

Los gastos generados por construir cualquier planta en cierta zona geográfica y por atender a cualquier cliente son los que se buscan reducir al mínimo sin dejar de satisfacer la demanda.

El objetivo de la tesis es mostrar una de las técnicas que encuentran una solución *rápida* del problema de localización de plantas aunque se incurra en un error, práctica que se hace comúnmente cuando se utilizan variables enteras en el modelo, pues hay ocasiones en las que el número de variables es tan grande que encontrar la mejor solución se complica en cuanto al tiempo requerido para resolver el problema.

En la primera parte de la tesis se incluye una base teórica de la programación lineal, la programación entera y la complejidad algorítmica. El problema central se define y se resuelve en la segunda parte de la tesis y es aquí donde con la ayuda de un ejemplo práctico se aprecia la sencillez del algoritmo.

Capítulo 1

Programación Lineal

El modelo de *programación lineal* (LP) por sus siglas en inglés, es uno de los más importantes en la investigación de operaciones y en las ciencias de la administración. Muchas situaciones cotidianas que busquen obtener una mejoría pueden ser resueltas o aproximadas por el LP; incluso en la programación entera, el LP es usado como subrutina.

1.1. Antecedentes Históricos

Los modelos matemáticos se usan desde siempre para resolver problemas y explicar fenómenos naturales. En particular *la programación matemática* surgió alrededor del siglo XVIII como una herramienta para modelar problemas en donde se debía optimizar cierta función continua $f(\mathbf{x})$ con $\mathbf{x} \in \mathbb{R}^n$. Los programas matemáticos más primitivos fueron utilizados por el economista Quesnay alrededor del año 1759. [32]

La *investigación de operaciones*, surgió con la revolución industrial y con la segunda Guerra Mundial, entre 1939 y 1945, cuando Gran Bretaña reunió a varios científicos de la época para idear estrategias de guerra con las que pudieran optimizar los recursos disponibles.

Tras finalizar la guerra, Gran Bretaña y Estados Unidos, inspirados en los excelentes resultados obtenidos en la milicia, empezaron a aplicar la investigación de operaciones también en los ámbitos industrial, laboral y gubernamental. Para 1951 ya se había introducido por completo en Gran Bretaña y estaba por

hacerlo en Estados Unidos.[30]

Al paso de los años, la investigación de operaciones fue evolucionando hasta convertirse en lo que es hoy en día: una de las disciplinas con más aplicaciones en el mundo real y sus herramientas se desarrollaron, en su mayoría, entre las décadas de 1950 y 1960; algunas de ellas son: la *programación lineal* (Dantzig 1947 [7]), la *programación dinámica* (Bellman 1953[27]), la *programación no lineal* (Kuhn y Tucker 1951 [20]), la *programación entera* (Gomory 1958 [12]), *las redes de optimización* (Ford y Fulckerson 1956 [9]), la *teoría de colas* (Erlang 1909 [24]), la *teoría de inventarios* (Arrow, Karlin, Scarf 1962 [29]) y la simulación (Montecarlo 1944 [33]).

1.2. Modelo de Programación Lineal

Un problema de programación lineal se puede definir como *optimizar* cierta función lineal sujeta a un conjunto de restricciones (funciones afines) las cuales pueden ser desigualdades o ecuaciones.

Dentro de un modelo de optimización se distinguen 3 partes: las *variables*, las *restricciones* y la *función objetivo*, los cuales se detallan a continuación

Variables

En un modelo de programación lineal se asume que algunas variables son *no controlables*, como la distancia entre dos lugares, el clima, la demanda, etc, mientras que otras son *controlables*, por ejemplo: decidir la ruta a tomar para viajar de un lado a otro, la época de siembra, el número de productos a fabricar, etc. Las controlables reciben el nombre de *variables de decisión*, mientras que las no controlables son valores estimados y dependiendo de qué tan realista se requiera el modelo, pueden o no formar parte de él.

Ejemplo

Supóngase que se quiere viajar de la ciudad **X** a la **Z**; la distancia d, medida en kilómetros, que existe entre **X** y **Z** es un ejemplo de variable *no controlable* pues no se tiene ningún control sobre ella.

Sin embargo, existen i rutas diferentes entre ambas ciudades, unas más largas que otras, lo que sugiere que la decisión es si conviene elegir la ruta i o no y esto aplica para cada ruta, por lo tanto las variables de decisión son las siguientes:

$$\text{Si } x_i = \begin{cases} 1 & \text{entonces se elige la ruta } i \\ 0 & \text{entonces no se elige la ruta } i \end{cases}$$

Los valores que toman las variables de decisión varían dependiendo del problema, en este caso cada variable puede tomar solo dos valores (1 ó 0) pero en otras situaciones, puede tomar valores enteros o racionales.

Función Objetivo

La función objetivo es una función lineal que depende de las variables de decisión y que se utiliza para medir el beneficio obtenido de la decisión tomada. Siguiendo con el mismo ejemplo, la gasolina que se gastaría a lo largo de la ruta i tiene un costo c_i.

El costo también es un ejemplo de variable no controlable y además sirve para definir la función objetivo:

$$\begin{aligned} f(\mathbf{x}) &: D \subset \mathbb{R} \to \mathbb{R} \\ f(\mathbf{x}) &= \mathbf{cx} \end{aligned}$$

en la cual $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t \in \mathbb{R}^n$ es el vector *columna* de variables de decisión, $\mathbf{c} = (c_1, c_2, \ldots, c_n) \in \mathbb{R}^n$ es el vector de costos o utilidades y D es el dominio en el cual se puede optimizar y es conocido también como *región factible*.

Restricciones

El dominio D o también llamado conjunto factible está constituido por las restricciones del problema, las cuales limitan el rango de valores que pueden tomar x.

En el ejemplo, la única restricción sería la siguiente:

$$\sum_i x_i = 1$$

ya que con esto se asegura que se debe elegir una y sólo una ruta para resolver el problema.

También existen restricciones sobre el signo de x y sobre el conjunto de valores permitidos (en el ejemplo x pertenece a un conjunto cuyos únicos valores son 0 y 1).

$$x_i \in \{0, 1\} \text{ para todo } i$$

De forma general se representan mediante funciones lineales de la forma:

$$R_i(\mathbf{x}) = \sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

donde i y j son los índices que numeran a las restricciones y a las variables respectivamente. Cabe aclarar que el signo $\leq$ puede ser reemplazado por el signo $\geq$ o por el signo $=$.

La intersección de todas ellas, define al conjunto D. El vector columna $\mathbf{b} = (b_1, b_2, \ldots, b_m)^t \in \mathbb{R}^m$, es llamado *vector de recursos.*

Los coeficientes a_{ij} forman una matriz de la forma:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

y con ella se puede definir el modelo de programación lineal:

$$\begin{array}{rl} \text{maximizar} & f(\mathbf{x}) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\ \text{sujeto a} & a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq b_1 \\ & \qquad\qquad\qquad \ddots \\ & a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq b_m \\ & x_1 \qquad x_2 \qquad \ldots \quad x_n \qquad \geq 0 \end{array} \tag{1.1}$$

o bien en forma matricial

$$\begin{array}{rrcl} \text{maximizar} & f(\mathbf{x}) & = & \mathbf{cx} \\ \\ \text{sujeto a} & \mathbf{Ax} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

el cual se dice que está en forma *canónica*; también existe la forma *estándar* en el que la única diferencia es que el término $\mathbf{Ax} \leq \mathbf{b}$ es reemplazado por $\mathbf{Ax} = \mathbf{b}$.

Con esto, se puede definir el modelo para el ejemplo que se utilizó anteriormente:

$$\begin{array}{rl} \text{minimizar} & f(\mathbf{x}) = c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{sujeto a} & x_1 + x_2 + \ldots + x_n = 1 \\ & x_1 \quad x_2 \quad \ldots \quad x_n \in \{0,1\} \end{array}$$

1.3. Región de Soluciones Factibles

Para resolver un modelo lineal, es necesario conocer las propiedades básicas de D; para ello, se definirán algunos términos.

Se llamará *solución factible* a cualquier vector $\mathbf{x} \in \mathbb{R}^n$ que cumpla con todas las restricciones, o dicho de otro modo, es factible si $\mathbf{x} \in D$.

El conjunto:

$$D = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

se llama *región de soluciones factibles* o *conjunto factible* para LP (cuando está en forma canónica).

1.3.1. Convexidad

Sean $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ un conjunto de puntos en $\mathbb{R}^n$, la combinación lineal convexa de estos puntos se define como:

$$\sum_{k=1}^{m} \lambda_k \mathbf{x}_k \text{ con } \sum_{k=1}^{m} \lambda_k = 1 \text{ y } \lambda_k \geq 0 \text{ para } k = 1, \ldots, m$$

Las dos propiedades más importantes de D son la convexidad y la existencia de puntos extremos.

Un conjunto $H \neq \emptyset$ es *convexo* si cumple lo siguiente: Para cualesquiera dos puntos $x, y \in H$, toda combinación lineal convexa de x y y, está totalmente contenida en H; es decir, $\lambda x + (1-\lambda)y \in H$ con $0 \leq \lambda \leq 1$.

Se demostrará ahora que la región factible D es un conjunto convexo:

Teorema 1.1. *Sea $D = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ el conjunto factible de cualquier problema lineal, entonces D es un conjunto convexo.*

Demostración: Sean $\overline{\mathbf{x}}, \overline{\mathbf{y}} \in D$.

Para que $\lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}}$ esté contenida en D, deben cumplirse dos cosas:

$$\mathbf{A}(\lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}}) \leq \mathbf{b} \quad \text{y} \quad \lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}} \geq 0$$

Notemos que en el caso de que el conjunto D solo tenga un punto, entonces la combinación lineal convexa de $\overline{\mathbf{x}}$ y $\overline{\mathbf{y}}$ también está en D, pues en este caso $\overline{\mathbf{x}} = \overline{\mathbf{y}}$ y $\lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}} = \overline{\mathbf{x}} \in D$. Por lo tanto nos enfocaremos al caso en que D contenga al menos dos puntos diferentes.

Primero se probará que $\lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}} \geq 0$:

Como $\overline{\mathbf{x}}, \overline{\mathbf{y}} \in D$, entonces $\overline{\mathbf{x}} \geq 0$ y $\overline{\mathbf{y}} \geq 0$, y considerando que $0 \leq \lambda \leq 1$ entonces ambos miembros de la suma son no negativos y por lo tanto la suma también lo es.

Ahora se demostrará que $\mathbf{A}(\lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}}) \leq \mathbf{b}$:

Se tiene que

$$\mathbf{A}(\lambda\overline{\mathbf{x}} + (1-\lambda)\overline{\mathbf{y}}) = \lambda\mathbf{A}\overline{\mathbf{x}} + (1-\lambda)(\mathbf{A}\overline{\mathbf{y}}) \leq \lambda\mathbf{b} + (1-\lambda)\mathbf{b} = \mathbf{b}$$

Por lo tanto queda demostrado que para cualesquiera 2 puntos de D, sus combinaciones lineales convexas también están en D, por lo que D es un conjunto convexo. □

1.3.2. Puntos Extremos

Los *puntos extremos* de un conjunto D se definen como aquellos que no pueden ser expresados como una combinación lineal convexa de cualesquiera otros dos puntos distintos pertenecientes al conjunto.

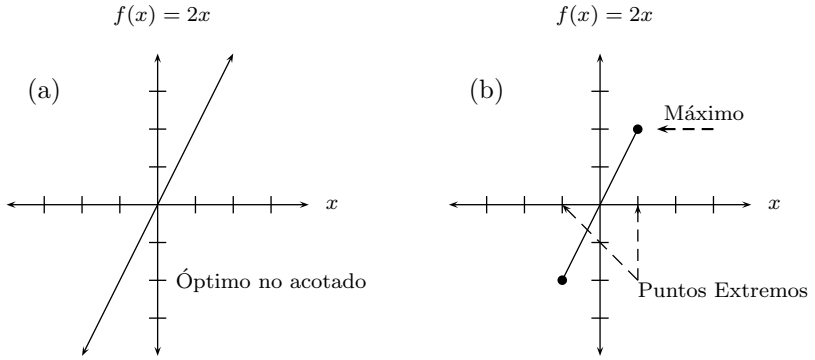

Figura 1.1: (a) $D = \mathbb{R}$, (b) $D = [-1, 1]$

Para ver la utilidad de los puntos extremos véanse los incisos (a) y (b) de la figura (1.1). En ambos incisos se intenta maximizar la función $f(x) = 2x$, la diferencia es que en (a), $D = \mathbb{R}$ el cual no tiene puntos extremos y en (b), $D = [-1, 1]$ cuyos puntos extremos son -1 y 1.

Con esto se intuye que el óptimo de cualquier función lineal *probablemente* se encuentre en alguno de los puntos extremos de D, es por esto que son tan importantes en la teoría de la programación lineal.

1.3.3. Variables Básicas y no Básicas

Se trabajará con la forma estándar del LP.

$$\begin{array}{lrcl} \text{maximizar} & f(\mathbf{x}) & = & \mathbf{cx} \\ \\ \text{sujeto a} & \mathbf{Ax} & = & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

La matriz $\mathbf{A}$ tiene m renglones y n columnas, eso quiere decir que hay m restricciones y n variables, se asumirá que $m \leq n$.

Supóngase que todos los renglones de $\mathbf{A}$ son linealmente independientes, entonces tomando m columnas linealmente independientes de $\mathbf{A}$ se obtiene $\mathbf{B}$, una submatriz de $\mathbf{A}$ de $m \times m$ a la que se llamará *matriz base*. Las $n - m$ columnas restantes de $\mathbf{A}$ forman a $\mathbf{N}$, otra submatriz de $\mathbf{A}$ de $(n - m) \times m$.

La siguiente expresión se utilizará para separar la matriz $\mathbf{A}$ en $\mathbf{B}$ y $\mathbf{N}$ suponiendo sin pérdida de generalidad, que las primeras m columnas de $\mathbf{A}$ son linealmente independientes

$$\mathbf{A} = (\mathbf{B}, \mathbf{N})$$

La misma expresión se usará para separar el vector de costos $\mathbf{c}$ y el vector de variables $\mathbf{x}$.

$$\begin{array}{rcl} \mathbf{x} & = & (\mathbf{x}_B, \mathbf{x}_N) \\ \mathbf{c} & = & (\mathbf{c}_B, \mathbf{c}_N) \end{array}$$

donde B es el conjunto de índices correspondientes a las columnas de $\mathbf{B}$ y N es el conjunto de índices correspondientes a las columnas de $\mathbf{N}$.

Sustituyendo estas nuevas matrices en el modelo, éste queda de la siguiente forma:

$$\begin{array}{lllll} \text{maximizar} & f(\mathbf{x}) = \mathbf{c}_B\mathbf{x}_B & + & \mathbf{c}_N\mathbf{x}_N & \\ & & & & \\ \text{sujeto a} & \mathbf{B}\mathbf{x}_B & + & \mathbf{N}\mathbf{x}_N = \mathbf{b} & (1.2) \\ & \mathbf{x}_B & & \mathbf{x}_N \geq 0 & \end{array}$$

Por lo tanto, la factibilidad está en función de las variables básicas y no básicas.

Una solución será básica factible si $\mathbf{x}_N$=0 y $\mathbf{x}_B \geq 0$. La matriz $\mathbf{B}$ asociada se llama *base factible.*

En el modelo (1.2) comúnmente se escribe z en lugar de $f(x)$

1.3.4. Variables básicas y puntos extremos

A continuación se tiene un ejemplo para visualizar todo lo anterior.

$$\begin{array}{rl} \text{maximizar} & f(x)=3\,x_1+4x_2 \\ \text{sujeto a} & 2\,x_1+\ x_2\leq 6 \\ & 2\,x_1+3x_2\leq 9 \\ & x_1,\quad x_2\geq 0 \end{array}$$

En la figura (1.2) se puede ver la región factible del problema anterior de dos variables y dos restricciones, en él se distinguen algunos puntos, los cuales corresponden a las intersecciones de las rectas y corresponden a las *soluciones básicas factibles* y para identificarlas es necesario estandarizar el problema, esto se hace añadiendo variables de *holgura*, las cuales no tienen ningún costo para la función objetivo sirven para que una desigualdad se vuelva igualdad:

$$\begin{array}{rl} \text{maximizar} & f(x)=3\,x_1+4x_2 \\ \text{sujeto a} & 2\,x_1+\ x_2+x_3 \qquad =6 \\ & 2\,x_1+3x_2 \qquad +x_4 =9 \\ & x_1,\quad x_2,\ x_3,\ x_4,\geq 0 \end{array}$$

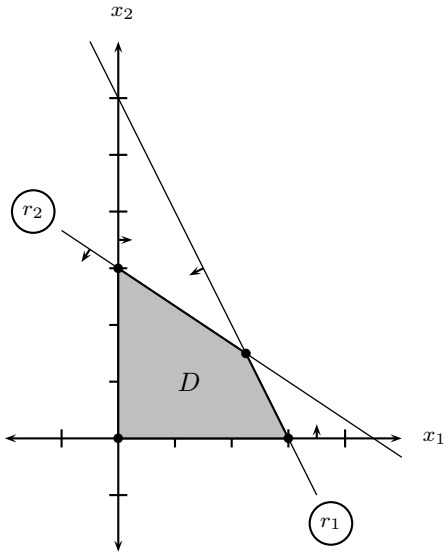

Figura 1.2: Representación gráfica y región factible del problema

La matrices **A** y **b** son por tanto:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 2 & 3 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 6 \\ 9 \end{pmatrix}$$

La base asociada de cualquier solución como por ejemplo $(x_1, x_2, x_3, x_4) = (3, 0, 0, 3)$, se pude encontrar identificando las variables básicas y no básicas, las cuales en el ejemplo son fáciles de localizar pues como sólo hay dos restricciones entonces sólo hay dos básicas $x_1 = 3$ y $x_4 = 3$ que por definición no pueden ser no básicas. Por lo tanto, la base **B** consta de las columnas a_1 y a_4 de **A**:

$$\mathbf{B} = \begin{pmatrix} 2 & 0 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_4 \end{pmatrix} \qquad \mathbf{B}^{-1} = \begin{pmatrix} 1/2 & 0 \\ -1 & 1 \end{pmatrix}$$

Haciendo el mismo análisis para todas las soluciones básicas de la región, se pueden obtener sus bases asociadas:

$$x_{B_1} = (0, 0, 6, 9) \qquad \mathbf{B}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad x_{B_2} = (3, 0, 0, 3) \qquad \mathbf{B}_2 = \begin{pmatrix} 2 & 0 \\ 2 & 1 \end{pmatrix}$$

$x_{B_3} = (0, 3, 3, 0)$ $\mathbf{B}_3 = \begin{pmatrix} 1 & 1 \\ 3 & 0 \end{pmatrix}$ $x_{B_4} = (\frac{9}{4}, \frac{3}{2}, 0, 0)$ $\mathbf{B}_4 = \begin{pmatrix} 2 & 1 \\ 2 & 3 \end{pmatrix}$

$x_{B_5} = (0, 6, 0, -9)$ $\mathbf{B}_5 = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$ $x_{B_6} = (\frac{9}{2}, 0, -3, 0)$ $\mathbf{B}_6 = \begin{pmatrix} 2 & 1 \\ 2 & 0 \end{pmatrix}$

Como se puede ver, B_5 y B_6 son bases no factibles, pues sus respectivas soluciones tienen elementos negativos.

1.4. Método Simplex

Ya se conocen todas la propiedades de la región factible, ahora sólo falta encontrar el óptimo. La función $f(x) = 3x_1 + 4x_2$ encuentra su valor máximo en algún punto extremo factible o solución básica factible, el cual en este caso es $x_{B_4} = (x_1, x_2, x_3, x_4) = (\frac{9}{4}, \frac{3}{2}, 0, 0)$ con un valor en la función objetivo de $f(x_{B_4}) = 12.75$.

La técnica para resolver cualquier problema lineal fue desarrollada por George B. Dantzig y se llama **método simplex** [7].

1.4.1. Coeficientes de costo reducido

El método simplex trabaja siempre con la forma estándar del problema y en cada iteración calcula una nueva solución básica factible basada en los criterios de optimalidad.

Recuérdese que en términos de variables básicas, el problema se ve de la forma:

$$\begin{aligned} \text{maximizar} \quad z &= \mathbf{c}_B\mathbf{x}_B + \mathbf{c}_N\mathbf{x}_N \\ \text{sujeto a} \quad \mathbf{x}_B &= \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N \\ \mathbf{x}_B, \mathbf{x}_N &\geq 0 \end{aligned} \tag{1.3}$$

Si se sutituye a $\mathbf{x}_B$ en z, se obtiene la expresión:

$$z = \mathbf{c}_B(\mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N) + \mathbf{c}_N\mathbf{x}_N$$

Distribuyendo términos y factorizando:

$$z = \mathbf{c}_B\mathbf{B}^{-1}\mathbf{b} - (\mathbf{c}_B\mathbf{B}^{-1}\mathbf{N} - \mathbf{c}_N)\mathbf{x}_N \tag{1.4}$$

Cuando se está en una solución básica factible, el valor de la función objetivo es $z = \mathbf{c}_B\mathbf{B}^{-1}\mathbf{b}$ pues $\mathbf{x}_N = 0$ y para fines prácticos, al término $\mathbf{c}_B\mathbf{B}^{-1}\mathbf{b}$ se le llamará z_0.

Sin embargo, el coeficiente de $\mathbf{x}_N$ en (1.4), sirve de indicador para ver si la solución actual es óptima.

Recordando que a_j es la columna j de la matriz A entonces se puede hacer lo siguiente:

$$(\mathbf{c}_B\mathbf{B}^{-1}\mathbf{N}-\mathbf{c}_N)\mathbf{x}_N = \sum_{j\in N}(\mathbf{c}_B\mathbf{B}^{-1}a_j - c_j)x_j = \sum_{j\in N}(\mathbf{c}_B y_j - c_j)x_j = \sum_{j\in N}(z_j - c_j)x_j$$

Sustituyendo, se tendría:

$$z = z_0 - \sum_{j\in N}(z_j - c_j)x_j$$

El coeficiente $z_j - c_j$ de la variable no básica x_j es llamado *coeficiente de costo reducido* e indica si la solución actual es la óptima, según el criterio de optimalidad.

1.4.2. Criterios de optimalidad y de selección

CRITERIO DE OPTIMALIDAD

- Si el problema es de minimización, se tendrá la solución óptima cuando $z_j - c_j \leq 0 \ \forall j \in N$.
- Si el problema es de maximización: se obtendrá la solución óptima cuando $z_j - c_j \geq 0 \ \forall j \in N$.

CRITERIOS DE SELECCIÓN

Si la solución actual no cumple con el criterio de optimalidad, lo que se hace es sustituir una de las variables básicas por una no básica, creando así una nueva solución básica y garantizando a la vez la factibilidad por medio de algunos criterios de selección.

Selección de la variable no básica (caso de minimización)

Como primer paso, se selecciona la variable no básica cuyo coeficiente de costo reducido $z_j - c_j$ sea el mayor (positivo). Cabe aclarar que éste es sólo un criterio y no garantiza que sea la forma más rápida de encontrar el óptimo.

Selección de la variable básica (caso de minimización)

Luego se debe seleccionar la variable básica que dejará de serlo, para esto se define al vector $\overline{\mathbf{b}}$ como sigue:

$$\mathbf{B}^{-1}\mathbf{b} = \overline{\mathbf{b}} = \begin{pmatrix} \overline{\mathbf{b}}_1 \\ \overline{\mathbf{b}}_2 \\ \vdots \\ \overline{\mathbf{b}}_m \end{pmatrix}$$

Supóngase, sin pérdida de generalidad, que las primeras m variables son básicas y que x_j con $j > m$ es la candidata a ser básica; entonces se tiene:

$$\mathbf{B}^{-1}\mathbf{N} = [y_{m+1}, y_{m+2}, \ldots, y_j, \ldots, y_n] \quad j \in N$$

con

$$y_j = (y_{1j}, y_{2j}, \ldots, y_{ij}, \ldots, y_{mj})^t \quad i \in B$$

la columna y_j (relacionada con la variable x_j) es llamada *columna pivote* y de ella se selecciona un elemento llamado *pivote* por medio de la siguiente expresión:

$$\frac{\overline{\mathbf{b}}_k}{y_{kj}} = \min_{1 \leq i \leq m} \left\{ \frac{\overline{\mathbf{b}}_i}{y_{ij}} | \text{ con } y_{ij} > 0 \right\}$$

el pivote y_{kj} indica que la variable básica que debe salir de la base es x_k.

Una vez que se tenga la nueva base, se recalcula todo y el proceso se repite hasta que se obtenga la solución óptima, o bien se concluya que la solución es no acotada.

Si el problema a resolver tiene restricciones del tipo $\leq$ ó $\geq$, entonces se deben convertir en igualdades, lo cual se logra sumándoles o restándoles la cantidad necesaria para que se cumpla la igualdad. A dichas cantidades se les llama variables de *holgura*.

Con base a todo lo anterior, el algoritmo es:

Algoritmo Simplex

Entrada : Una matriz $\mathbf{A} \in \mathbb{R}^{m \times n}$, dos vectores columna $\mathbf{c} \in \mathbb{R}^n$ y $\mathbf{b} \in \mathbb{R}^m$ y un conjunto de soluciones factibles $D = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}$.

Salida : Una solución básica factible $\overline{x}$ tal que $c\overline{x} = \text{máx}\{cx | x \in D\}$ o bien concluir que la solución es no acotada.

① Estandarizar el problema añadiendo variables de holgura.
Seleccionar un punto extremo factible $\overline{x} \in D$

② Calcular $\mathbf{B}, \mathbf{N}, B, N$ y los coeficientes de costo reducido asociados a $\overline{x}$.

③ Si la solución actual cumple el criterio de optimalidad entonces regresar $\overline{x}$
Terminar.

④ Seleccionar el índice $j \in N$ de la variable no básica que es candidata a ser básica según el criterio de selección.
Calcular el vector $y_j = \mathbf{B}^{-1} a_j$
Si $y_j \leq 0$ entonces la solución es no acotada
Terminar.

⑤ Calcular el cociente mínimo según el criterio de selección.
Seleccionar el índice $k \in B$ de la variable básica que dejará de ser básica.

⑥ Hacer $N = N \backslash \{j\} \cup \{k\}$ y $B = B \backslash \{k\} \cup \{j\}$.
Ir al paso 2

1.5. Dualidad

Todos los problemas de optimización, llamados también *primales*, tienen asociado un problema equivalente llamado *dual* el cual tiene muchas aplicaciones dentro de la programación lineal.

La teoría de la dualidad fue desarrollada por el matemático John Von Neumann en 1947, mismo año en el que Dantzig desarrollaba el algoritmo simplex. [7]

Von Neumann propuso que el dual del problema primal:

$$\begin{array}{lrcl} \text{maximizar} & z & = & \mathbf{cx} \\ \\ \text{sujeto a} & \mathbf{Ax} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array}$$

es el siguiente

$$\begin{array}{lrcl} \text{minimizar} & \nu & = & \mathbf{b}^t\mathbf{y} \\ \\ \text{sujeto a} & \mathbf{yA} & \geq & \mathbf{c} \\ & \mathbf{y} & \leq & \mathbf{0} \end{array}$$

1.5.1. Relación entre los problemas primal y dual

Existen varias relaciones entre un problema primal y el dual asociado:

- El número de *restricciones* del primal es igual al número de *variables* del dual.
- El número de *variables* del primal es igual al número de *restricciones* del dual.
- El dual del dual es el problema primal.

- Sea $\mathbf{x}^*$ y $\mathbf{y}^*$ las soluciones óptimas del primal y del dual respectivamente, entonces las funciones objetivo de ambos problemas son iguales, es decir $z^* = \nu^*$.
- Si el problema primal (dual) es no acotado, entonces el dual (primal) es no factible.

Hay ocasiones en las que el dual es más fácil de resolver que el primal, por esta razón se necesita saber una manera de encontrar el problema dual a partir de un primal dado, lo cual se logra con la siguiente tabla:

	Primal	Dual	
	minimizar	maximizar	
Restricciones	$\leq$	≤ 0	Variables
	$\geq$	≥ 0	
	$=$	s.r.s.	
Variables	≥ 0	$\leq$	Restricciones
	≤ 0	$\geq$	
	s.r.s.	$=$	

Capítulo 2

Programación Entera

2.1. Antecedentes Históricos

La programación entera o *IP* por sus siglas en inglés, surgió alrededor del año *1958* por la necesidad de resolver problemas de optimización en los que la solución, además de cumplir con todas las restricciones, debe ser entera para *al menos* una de sus variables.

Algunos de los eventos más relevantes que desarrollaron a la programación entera, se citan a continuación:

1958 [12]: El matemático Ralph E. Gomory publica un artículo en el que se habla por primera vez del método de *planos de corte* para resolver problemas enteros.

1960 [21]: Land y Doig describen otro de los métodos más importantes de la IP, el llamado *método de ramificación y acotamiento.*

2.2. Programación Entera Lineal

2.2.1. Modelo de Programación Entera Lineal

El algoritmo simplex funciona para resolver prácticamente cualquier LP, sin embargo otros más interesantes se modelan usando la IP y el motivo por el que no se puede usar el algoritmo simplex es que el conjunto de soluciones factibles de cualquier IP es *no convexo*.

Existen diferentes clases de IP's:

- Problema entero puro. Cuando todas las variables son enteras.
- Problema entero mixto. Cuando al menos una variable es entera y existe otra que no lo es.
- Problema entero lineal. Cuando todas las restricciones y la función objetivo son lineales.
- Problema entero no lineal. Cuando al menos una restricción es no lineal.

El modelo IP de la clase *entero puro* en su forma general es el siguiente:

$$\begin{aligned} \text{minimizar} \quad & z = \mathbf{cx} \\ \text{sujeto a} \quad & \\ & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \in Z^n \end{aligned}$$

El conjunto de soluciones factibles para IP se denotará por F:

$$F = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \in Z^n\}$$

2.3. Técnica Branch & Bound

Las técnicas más famosas que existen para resolver IP's se clasifican en dos tipos: *planos de corte* o *cutting plane* y *ramificación y acotamiento* o *branch and*

bound. A continuación se explica brevemente uno de los algoritmos, basados en ramificación y acotamiento, más representativos.

2.3.1. Algoritmo de Dakin (Branch and Bound)

El primer algoritmo de ramificación y acotamiento fue introducido en 1960 por Land y Doig [21], el cual consiste principalmente en dividir el problema original en n subproblemas y resolverlos usando el algoritmo simplex. Una vez resueltos, a todos aquellos cuya solución no fue entera se les aplica el mismo proceso. Las soluciones óptimas enteras se conservan y al final se elije la mejor de todas.

Posteriormente Dakin [6] propuso otro algoritmo de ramificación en el que cada problema estudiado se divide solo en 2 subproblemas en cada iteración, disminuyendo así el número de problemas por analizar.

Para fines prácticos, el problema original se llamará P_0 y su región factible será $F_0 = F$.

El primer paso en este algoritmo es relajar la integralidad (integralidad significa que las variables son enteras) de P_0 para volverlo un LP que se llamará P_0'. La región factible de P_0' es D_0, con $F_0 \subseteq D_0$, donde:

$$D_0 = \{x | Ax \leq b,\ x \geq 0\}$$

P_0' se resuelve usando simplex. Si la solución óptima $\overline{x_0}$ cumple con la restricción de integralidad, entonces $\overline{x_0}$ también es solución para P_0. Si no es así, P_0 debe ser separado en dos subproblemas (P_1 y P_2) de tal forma que $\overline{x_0}$ sea no factible en ambos.

Supóngase que la j-ésima variable de $\overline{x_0}$ no es entera, entonces:

$$x_0^j = [x_0^j] + f_j \text{ con } 0 < f_j < 1$$

para lograr que $\overline{x_0}$ sea no factible para P_1 y P_2 entonces sus regiones factibles son:

$$F_1 = F \cap \{x | x_0^j \leq [x_0^j]\}$$
$$F_2 = F \cap \{x | x_0^j \geq [x_0^j] + 1\}$$

el proceso anterior es lo que se conoce como *ramificación* y una vez concluido, se procede a analizar a P_1 y P_2 de la misma manera que se hizo con P_0, y este proceso se repite hasta que todos los subproblemas den como resultado una solución óptima entero-factible.

Sin embargo, existen cotas que indican si conviene o no ramificar un subproblema. Al proceso de calcular tales cotas se le conoce como *acotamiento*.

Para explicar como funciona el acotamiento, supóngase que el valor óptimo de la función objetivo de P_0' es z_0' y el de P_0 es z_0^*, entonces como $F_0 \subseteq D_0$, $z_0^* \leq z_0'$ para el caso de maximización y $z_0^* \geq z_0'$ para el caso de minimización.

Para P_0, la cota superior es $\overline{z_0} = z_0'$ y la cota inferior se puede tomar como $\underline{z_0} = f(x')$ para cualquier $x' \in F_0$.

La cota inferior de P_0 sirve como criterio para saber si alguno de sus subproblemas debe ser ramificado o no. Por ejemplo, en el caso de maximización si la cota superior de P_1 es menor que la cota inferior de P_0 entonces no tiene sentido seguir ramificando hacia P_1 pues jamás se obtendrá una mejor solución, es entonces cuando P_1 se vuelve *no prometedor*.

Se define a L como la lista de problemas *pendientes* por analizar. Un problema deja de pertenecer a L cuando: se ramificó en otros dos, cuando se consideró no prometedor o bien cuando su solución es entero-factible.

Para ayudar a representar la ejecución del algoritmo, se utilizan árboles en el que la raíz representa al problema original y los vértices representan los subproblemas del original, los cuales sólo tienen dos ramas y en cada una de ellas se encuentra la restricción que se añadió. También se coloca al lado de cada vértice una etiqueta con las cotas inferior y superior encontradas durante el acotamiento $[\underline{z}, \overline{z}]$.

2.3.2. Descripción del algoritmo

Algoritmo de Dakin

Entrada : Una instancia (F, c) de cualquier problema entero.

Salida : Una solución $x^* \in F$ tal que cx^* es óptimo.

① **Inicializar** la cota $\underline{z} = -\infty$ y hacer $L = \{P_0\}$.

② **Elegir** aleatoriamente un problema P_n de L.

Relajar a P_n y nombrar al problema relajado como P'_n, resolver P'_n usando el algoritmo simplex.

③ **Si** P'_n es no factible, **entonces**: $L = L\backslash\{P_n\}$, **Ir** al paso 5.

En otro caso:

$\overline{z_n} = z_n^*$, donde z_n^* es el valor objetivo óptimo de P'_n.

Si la solución óptima de P'_n es entero-factible **entonces**

Si $\overline{z_n} > \underline{z}$ **entonces** $\underline{z} = \overline{z_n}$, etiquetar a la solución óptima $\overline{x_n^*}$ de P'_n como solución candidata.

En otro caso $L = L\backslash\{P_n\}$, Ir al paso 5.

④ **Si** $\overline{z_n} \leq \underline{z}$ **entonces** $L = L\backslash\{P_n\}$, **Ir** al paso 5.

En otro caso Seleccionar alguna variable x_j que no sea entera de la solución óptima de P'_n y construir dos problemas P_k y P_{k+1} con regiones factibles F_k y F_{k+1} como sigue:

$$F_k = F_n \cap \{x | x_j \leq [x_j]\}$$
$$F_{k+1} = F_n \cap \{x | x^j \geq [x^j] + 1\}$$

Hacer $L = L\backslash\{P_n\} \cup \{P_k, P_{k+1}\}$.

⑤ **Si** $L \neq \emptyset$, **Ir** al paso 2.

En oto caso La solución óptima entero-factible de P_0 es $\overline{x_n^*}$ y su valor objetivo es $\underline{z}$.

Para visualizar mejor la ejecución del algoritmo, se resolverá el siguiente ejemplo:

$$\begin{array}{rrcrl} \text{maximizar} & 3x_1 & + & 4x_2 & \\ \text{sujeto a} & 2x_1 & + & x_2 & \leq 6 \\ & 2x_1 & + & 3x_2 & \leq 9 \\ & x_1 & & x_2 & \in \mathbb{Z}^+ \end{array}$$

En las siguientes gráficas la región sombreada representa al conjunto de soluciones factibles D_j del problema lineal P'_j asociado a P_j, los puntos representan el conjunto de soluciones enteras factibles F_j y la línea punteada representa una curva de nivel $z = 0$ de la función objetivo.

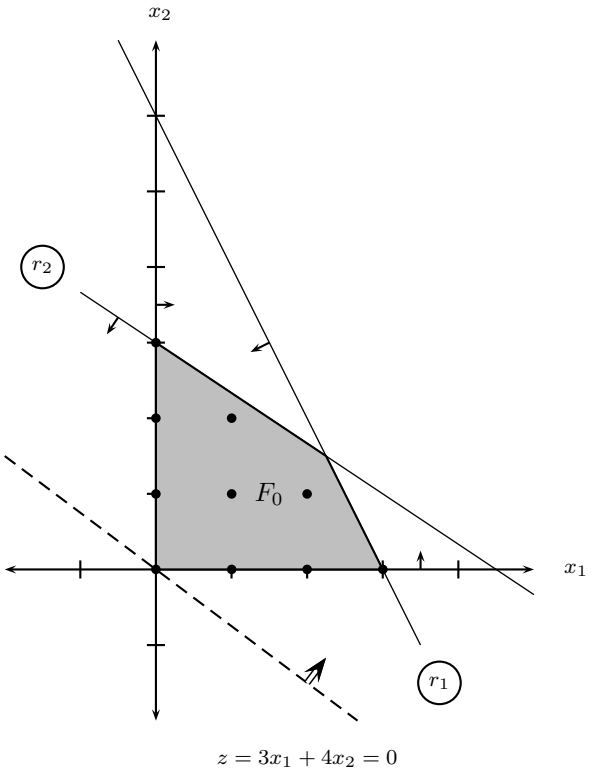

Figura 2.1: Región factible del problema

El algoritmo empieza con el único problema que pertenece a L, es decir P_0, e inicializando la cota inferior del problema como $\underline{z} = -\infty$.

En la figura 2.1 se encuentra el problema original, y sus dos regiones factibles D_0 y F_0.

Como siguiente paso, se debe resolver el problema lineal P_0' asociado a P_0, la cual puede apreciarse en la figura 2.1 que su solución óptima es $x_0^* = (\frac{9}{4}, \frac{3}{2})$ y su valor objetivo es $z_0^* = \overline{z_0} = \frac{51}{4}$.

La solución no es entera, sin embargo como $\overline{z_0} > \underline{z}$, se debe ramificar a P_0 en P_1 y P_2 tal como lo muestra la figura 2.2.

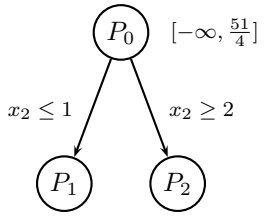

Figura 2.2: Árbol asociado al primer paso del algoritmo

L se actualiza con los dos nuevos problemas, quedando como $L = \{P_1, P_2\}$. Las regiones factibles de P_1 y P_2 se muestran en la figura 2.3.

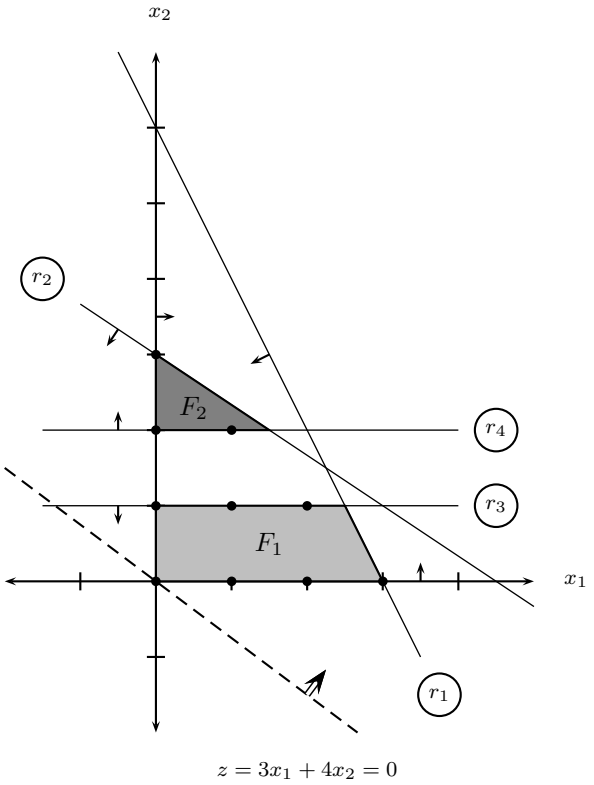

Figura 2.3: Región factible de P_1 y P_2

De la lista $L = \{P_1, P_2\}$ se elije para ramificar a P_2. Se resuelve su problema lineal asociado P_2', cuya solución óptima es $x_2^* = (\frac{3}{2}, 2)$ con cota superior $\overline{z_2} = \frac{25}{2}$.

Como la solución no es entera y $\overline{z_2} > \underline{z}$, se procede a ramificar a P_2 en P_3 y P_4, tal como lo muestra la figura 2.4. Se actualiza a L, quedando como $L = \{P_1, P_3, P_4\}$

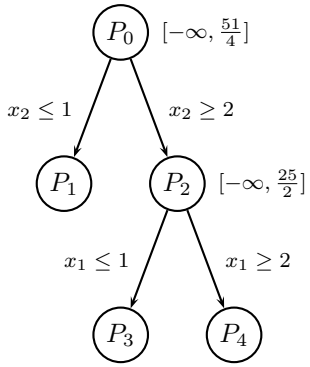

Figura 2.4: Árbol generado tras ramificar a P_2

Las regiones factibles F_3 y F_4 se encuentran en la figura (2.5) y como se puede ver, $F_4 = \emptyset$ y por lo tanto P_4 sale de L, la cual queda actualizada como $L = \{P_1, P_3\}$. Se elige arbitrariamente a P_3 y se resuelve P_3'.

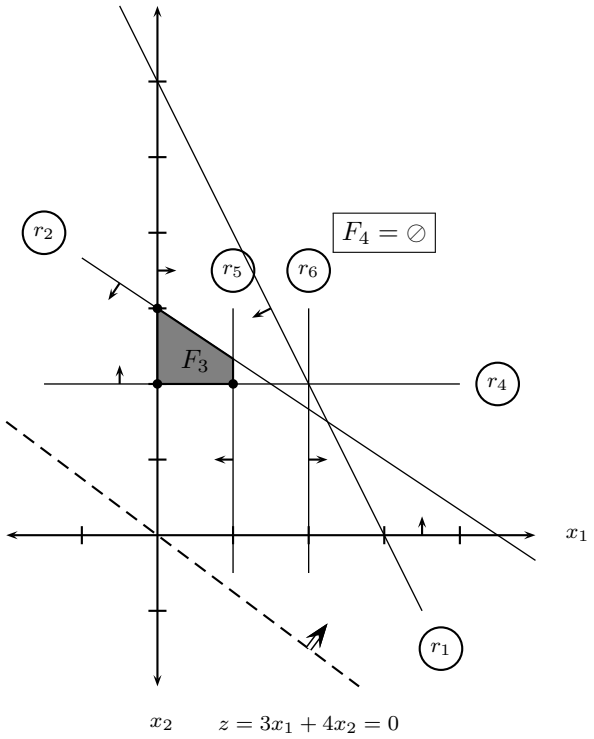

Figura 2.5: Regiones factibles de P_3 y P_4

La solución óptima de P_3' es $x_3^* = (1, \frac{7}{3})$ y su cota superior es $\overline{z}_3 = z_3^* = \frac{37}{3}$. Como no es entera y $\overline{z}_3 > \underline{z}$ entonces se ramifica P_3 en P_5 y P_6. En la figura 2.6

se encuentra el árbol asociado a este paso de algoritmo y las regiones factibles asociadas son $F_5 = \{(0,2)(1,2)\}$ y $F_6 = \{(0,3)\}$.

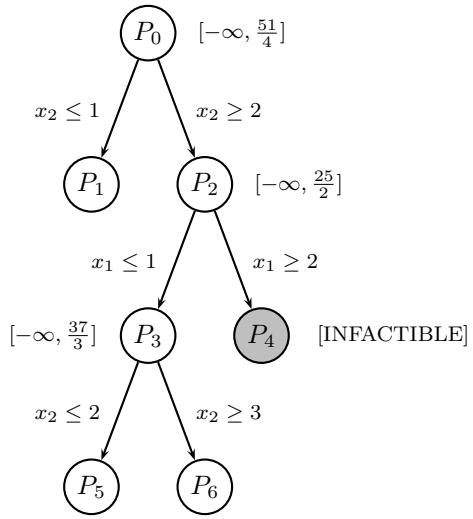

Figura 2.6: Árbol generado después de ramificar a P_3

Por último, de la lista $L = \{P_1, P_5, P_6\}$, se elige a P_6; su solución óptima es $x_6^* = (0,3)$ y su valor objetivo es $\overline{z_6} = z_6^* = 12$.

La solución x_6^* es entero-factible, por lo que la cota inferior del problema se actualiza por $\underline{z} = \overline{z_6}$.

Como $\overline{z_1} = 8{,}5 < \underline{z}$, entonces el problema P_1 se vuelve un problema no prometedor y por consiguiente, la lista queda como $L = \{P_5\}$.

La solución óptima de P_5' es $x_5^* = (1,2)$ y la cota superior es $\overline{z_5} = z_5^* = 11$, sin embargo, también es no prometedor por lo que la lista queda vacía $L = \emptyset$ y por tal motivo, la solución óptima es $x^* = (0,3)$ con valor objetivo $z^* = \underline{z} = 12$. El último árbol asociado al algoritmo, se encuentra en la figura 2.7.

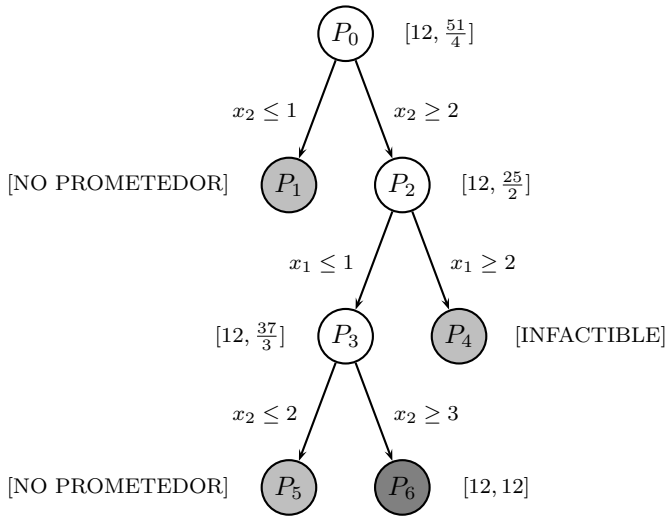

Figura 2.7: Árbol final

Capítulo 3

Complejidad Algorítmica en la Programación Entera

La *complejidad algorítmica* es una herramienta para analizar algoritmos que ayuda a calcular su rapidez y eficiencia al momento de usarlos para resolver algún problema en particular.

Surgió en el año 1972 cuando el matemático Karp [18], basándose en un artículo publicado por Cook en 1971 [5], introdujo las clases de problemas P y NP y formuló la pregunta ¿$P \neq NP$? la cual, hoy en día, sigue sin tener solución.

3.1. Definiciones Básicas

3.1.1. Tipos de Algoritmos

En cuanto al tiempo de ejecución, existen los algoritmos de tiempo *polinomial* y los de tiempo *no polinomial*.

Algoritmos polinomiales y no polinomiales

En general, mientras más variables existan, mayor es el tiempo que tarda un algoritmo en resolver el problema. Los *algoritmos polinomiales* son aquellos en

los que el tiempo que consumen al resolver algún problema puede ser acotado por un polinomio. Por el contrario, los *algoritmos no polinomiales* carecen de esa propiedad.

Con respecto a la exactitud, los algoritmos se clasifican en *óptimos* y *aproximados*.

Algoritmos óptimos y aproximados

Al tratar de resolver problemas IP, los algoritmos clásicos como el algoritmo de Dankin, aunque *garantizan* encontrar la solución óptima, son no polinomiales. Sin embargo, hay situaciones prácticas en las que se necesita tener una respuesta rápida aunque la solución no sea necesariamente la mejor.

Se definirá como *instancia* al conjunto de elementos necesarios para definir algún problema.

Los *algoritmos aproximados* se acercan al óptimo de la siguiente forma:

Sea I la instancia de un problema de optimización, y sean $OPT(I)$ la solución óptima y $A(I)$ la calculada por el algoritmo A entonces:

$$\frac{1}{k}OPT(I) \leq A(I) \leq kOPT(I) \quad \text{con } k \geq 1$$

Al número k se le conoce como *factor de aproximación* pues nos indica que tan lejos se está del óptimo.

Los *algoritmos óptimos* son aquellos en los que $k = 1$.

Algoritmos deterministas y no-deterministas

Hay otra clasificación de algoritmos relevante en la teoría de la complejidad, los *no-deterministas* en los que en contraposición a los *deterministas*, se agrega una fase previa que *adivina* la solución del problema (el componente que adivina es llamado *oráculo*) para luego checar que efectivamente dicha adivinanza es solución del problema.

3.1.2. Problemas de decisión

La complejidad algorítmica está basada principalmente en los *problemas de decisión*, los cuales están formados por una *instancia* y una *pregunta* cuya respuesta sólo puede ser *si* o *no*, por ejemplo:

Factibilidad Entera

Instancia : Una matriz $A \in \mathbb{Z}^{m \times n}$ y un vector $b \in \mathbb{Z}^m$.

Pregunta : ¿Existe un vector $x \in \mathbb{Z}^n$ tal que $Ax \leq b$?.

formalmente se definen así:

Definición 3.1. *Un problema de decisión* $\mathcal{P}= (X, Y)$ *es un conjunto de instancias* X *y un subconjunto* $Y \subseteq X$ *que son aquellas que regresan si (las llamadas si-instancias)*

3.1.3. Tipos de problemas

Dentro de la complejidad algorítmica, también existe una clasificación de problemas que sirve para identificar su *dificultad* al momento de resolverlos. Las clasificaciones más importantes son: $P, NP, NP-Completo$ y $NP-Duro$.

Clase P

La clase P contiene a todos aquellos problemas de decisión en los que verificar que alguna instancia del conjunto X pertenece a Y, se puede realizar con un algoritmo polinomial.

Clase NP

Se dice que un problema de decisión $\mathcal{P}= (X, Y)$ pertenece a la clase NP cuando se puede verificar que alguna instancia x pertenece a Y en tiempo polinomial usando un algoritmo no-determinista.

Se sabe que $P \subseteq NP$, pero la pregunta que aún sigue abierta dentro de la complejidad algorítmica es ¿$P = NP$?.

Transformaciones de problemas

Sean $\mathcal{P}_1 = (X_1, Y_1)$ y $\mathcal{P}_2 = (X_2, Y_2)$ dos problemas de decisión. Se dice que $\mathcal{P}_1$ se *transforma polinomialmente* en $\mathcal{P}_2$ si las instancias X_1 se transformen en instancias X_2 y las sí-instancias de $\mathcal{P}_1$ se transforman en sí-instancias de $\mathcal{P}_2$, usando un algoritmo polinomial.

Se sigue inmediatamente de la definición que:

Proposición 3.1. *Sean dos problemas de decisión $\mathcal{P}_1$ y $\mathcal{P}_2$, Si $\mathcal{P}_1$ se transforma polinomialmente en $\mathcal{P}_2$ y existe un algoritmo polinomial para $\mathcal{P}_2$, entonces existe un algoritmo polinomial para $\mathcal{P}_1$. [19]*

Clase NP-Completo

Se dice que $\mathcal{P} \in NP$ pertenece a la clase $NP - Completo$ si *todos* los problemas de la clase NP se *transforman polinomialmente* en $\mathcal{P}$.

La clase $NP - Completo$ es muy importante, pues gracias a la proposición anterior, si se encuentra que alguno de los problemas de esta clase tiene un algoritmo polinomial, entonces todos los demás también lo tienen y por lo tanto, P sería igual a NP.

Clase NP-Duro

Por otro lado, $\mathcal{P}$ pertenece a $NP-Duro$ si *todos* los problemas de la clase NP se *transforman polinomialmente* en $\mathcal{P}$ aunque $\mathcal{P}$ no necesariamente pertenezca a NP.

La forma más fácil de demostrar que un problema de optimización pertenece a la clase $NP - Duro$ es encontrando su problema de decisión asociado y demostrar que éste pertenece a la clase $NP - Completo$. [25]

3.2. Ejemplos de problemas NP-Completos

Gracias al matemático Cook [5], el primer problema que se demostró que pertenece a la clase $NP - Completo$ es el de *satisfacibilidad*, el cual consta de un número de variables binarias en las que 1 significa *verdadero* y 0 significa *falso* y un conjunto de *clausulas* que son la unión da varias variables. Un ejemplo de tres variables sería:

$$C = \{x_1 \vee x_2 \vee x_3\}$$

Para que C sea válida es necesario que al menos una de las x_i con $i = 1, 2, 3$ sea verdadera. Una *familia* $\mathcal{Z}$ de clausulas es la intersección de todas ellas:

$$\mathcal{Z} = C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

una *asignación de verdad* es una función T tal que $T(x) = verdadero$ si $x = 1$ y viceversa. Se define como $\overline{x}$ al complemento de x y $\overline{x} = 1$ si y solo si $x = 0$.

Si existe una asignación de verdad que satisfaga todas las clausulas al mismo tiempo entonces el problema es *satisfacible*.

Posterior al trabajo de Cook, el matemático Richard Karp utilizó el hecho de que el problema de satisfacibilidad pertenecía a la clase $NP - Completo$ para demostrar que otros 21 problemas de decisión pertenecían a la misma clase [18], entre los cuales destacan: el problema del clan, el del conjunto estable o conjunto independiente y el de la cubierta por vértices.

Sea $G = (V(G), E(G))$ una gráfica en donde $V(G)$ es el conjunto de vértices y $E(G)$ es el conjunto de aristas. Se define *clan*, *estable* y *cubierta de vértices* como sigue:

- *Clan*: Es un subconjunto $X \subseteq V(G)$ tal que para cualesquiera dos elementos $v, w \in X$ existe una arista que los conecta.
- *Estable*: Al contrario del clan, es un subconjunto $Y \subseteq V(G)$ tal que ningún $y \in Y$ está conectado.
- *Cubierta por vértices*: es un conjunto $S \subseteq V(G)$ tales que cada arista de G tiene como extremo al menos un vértice de S.

Los últimos tres problemas son fáciles de relacionar gracias a la siguiente proposición:

Proposición 3.2. *Sea G una gráfica y $X \subseteq V(G)$. Entonces los siguientes 3 enunciados son equivalentes [19]:*

1. *X es una* cubierta por vértices *en G.*
2. *$V(G) \backslash X$ es un* conjunto estable *de G.*

3. $V(G)\backslash X$ *es un* clan *en el complemento de* G.

Problema del Clan

Instancia : Una gráfica G y un entero k.

Pregunta : ¿Existe un clan de cardinalidad k en G?.

Problema del Conjunto Estable

Instancia : Una gráfica G y un entero k.

Pregunta : ¿Existe en G un conjunto estable de k vértices?.

Problema de la cubierta por vértices

Instancia : Una gráfica G y un entero k.

Pregunta : ¿Existe una cubierta por vértices de cardinalidad k en G?.

El siguiente resultado se debe a Karp [18] pero los detalles de la demostración se omitieron por no ser relevantes para la tesis.

Teorema 3.1. *El problema del clan, el problema de la cobierta por vértices y el problema del conjunto estable son problemas que pertenecen a la clase* $NP-Completo$ *[18]*

3.3. Problema de la mínima cubierta por conjuntos

Un problema importante que pertenece a la clase $NP-Duro$ es el de la cubierta por conjuntos con cardinalidad mínima o mejor conocido como el problema de la mínima cubierta por conjuntos.

Problema de la mínima cubierta por conjuntos

Instancia : Un sistema (U,$\mathcal{S}$) tal que $\bigcup_{S\in\mathcal{S}} S = U$.

Objetivo : Encontrar una cubierta con cardinalidad mínima $\mathcal{R} \subseteq \mathcal{S}$ tal que $\bigcup_{R\in\mathcal{R}} R = U$.

Suponiendo que para toda $x \in U$, $|\{S \in \mathcal{S} : x \in S\}| = 2$ entonces éste se transforma en el de la mínima cubierta por vértices en la siguiente forma:

Dada una gráfica G de una instancia de la mínima cubierta por vértices, la instancia equivalente de la mínima cubierta por conjuntos se define como $U := E(G)$ y $S = \{\delta(v) : v \in V(G)\}$ para todo $v \in V(G)$, donde $\delta(v)$ es el conjunto de todos los vértices adyacentes a v.

Por el teorema 3.1, se sabe que el problema de la cubierta por vértices pertenece a la clase $NP - Completo$, por lo tanto el de la mínima cubierta por vértices pertenece a la clase $NP - Duro$ y a su vez también el de la mínima cubierta por conjuntos.

3.4. Tiempos de ejecución

Conocer la clasificación de los problemas en cuanto a su complejidad (P, NP, $NP - Completo$ ó $NP - Duro$) es importante, sin embargo resulta útil conocer el tiempo que utilizan en ejecutarse.

Se dice que cada vez que en el código de un algoritmo, se asigne un valor a una variable, se comparen dos variables, se realicen operaciones aritméticas básicas (suma, resta, multiplicación, división, raíz cuadrada) o se ejecute una condición (*si-entonces*) se utiliza una unidad de tiempo.

Para medir el tiempo de ejecución, se debe estimar el *número de veces* que se ejecutarán cada uno de los pasos del algoritmo hasta obtener el resultado, basándose en la unidad de medida definida anteriormente.

Aunque no se puede saber con exactitud el número de veces que se ejecutará algún paso del algoritmo, existen enfoques como el *pesimista* que siempre supone que se está en el peor caso o el *optimista* que supone lo contrario, y esto ayuda a estimar el tiempo empleado por un algoritmo. El enfoque que se utilizará es el *pesimista*, ya que es el que más se aplica en la realidad.

Sea A un algoritmo y n el número de variables del problema a resolver con A, se utiliza la notación $\mathcal{O}(f(n))$ para decir que el algoritmo se ejecuta en un tiempo del orden de la función $f(n)$. Se dice que el algoritmo se ejecuta en tiempo constante cuando $f(n) = 1$.

A continuación unos ejemplos para explicar lo anterior:

Sea $T(A(n))$ el tiempo de ejecución de A.

1. $T(A(n)) = 3n + 5$, entonces su tiempo es del orden $\mathcal{O}(n)$.

2. $T(A(n)) = 5$, entonces su tiempo es del orden $\mathcal{O}(1)$.

3. $T(A(n)) = n\log_2(n) + 4n + 3$, entonces es del orden $\mathcal{O}(n\log_2(n))$.

A continuación se muestra una tabla de tiempos de ejecución para instancias de diferentes tamaños y algoritmos polinomiales y no polinomiales

	Tamaño de n					
Tiempos	10	20	30	40	50	60
$\mathcal{O}(n)$	0.00001 segs	0.00002 segs	0.00003 segs	0.00004 segs	0.00005 segs	0.00006 segs
$\mathcal{O}(n^2)$	0.0001 segs	0.0004 segs	0.0009 segs	0.0016 segs	0.0025 segs	0.0036 segs
$\mathcal{O}(n^3)$	0.001 segs	0.008 segs	0.027 segs	0.064 segs	0.125 segs	0.216 segs
$\mathcal{O}(n^5)$	0.1 segs	3.2 segs	24.3 segs	1.7 mins	5.2 mins	13.0 mins
$\mathcal{O}(2^n)$	0.001 segs	1.0 seg	17.9 mins	12.7 días	35.7 años	366.0 siglos
$\mathcal{O}(3^n)$	0.059 segs	58.0 mins	6.5 años	3855 siglos	$2 * 10^8$ siglos	N/D

3.4.1. Algoritmo Merge-Sort

Uno de los problemas más importantes y con muchas aplicaciones es encontrar el mejor método para ordenar una lista finita de números.

Aunque existen muchos algoritmos para resolverlo, existe uno que tiene un tiempo de ejecución eficiente, el cual recibe el nombre de *Algoritmo Merge-Sort*.

A continuación se describirá el algoritmo y se demostrará que tiene un tiempo de ejecución del orden $\mathcal{O}(n\log_2(n))$ donde n es el total de números de la lista.

Algoritmo Merge-Sort

Entrada : Una lista $a_1, \ldots, a_n$ de números reales.

Salida : Una permutación $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ tal que para todo $i = 1, \ldots, n$, $a_{\pi(i)} \leq a_{\pi(i+1)}$.

① **Si** $n = 1$ **entonces**, $\pi(1) := 1$ y **detener** (**regresar** π).

② Definir $m := \lfloor \frac{n}{2} \rfloor$.

ρ:=**Merge-Sort**$(a_1, \ldots, a_m)$.

σ:=**Merge-Sort**$(a_{m+1}, \ldots, a_n)$.

③ Hacer $k := 1, l = 1$.

Mientras $k \leq m$ y $l \leq n - m$, **Hacer**:

Si $a_{\rho(k)} \leq a_{m+\sigma(1)}$ **entonces** $\pi(k + l - 1) := \rho(k)$ y $k := k + 1$

en otro caso $\pi(k + l - 1) := m + \sigma(l)$ y $l = l + 1$.

Mientras $k \leq m$ **hacer**: $\pi(k + l - 1) := \rho(k)$ y $k := k + 1$.

Mientras $l \leq n - m$ **hacer**: $\pi(k + l - 1) := m + \sigma(l)$ y $l := l + 1$.

Teorema 3.2. *El algoritmo* Merge-Sort *se ejecuta en un tiempo del orden* $O(n \log n)$.

Demostración. Se denota por $T(n)$ al tiempo de ejecución que se necesita para resolver instancias que constan de n números.

Obsérvese que $T(1) = 1$ y que $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 3n + 6$, donde $\lfloor \frac{n}{2} \rfloor$ es el entero menor o igual que $\frac{n}{2}$ y por el contrario $\lceil \frac{n}{2} \rceil$ es el mayor o igual que $\frac{n}{2}$. (El término $3n + 6$ depende de que tan exacto se quiera medir el tiempo, pero para efectos prácticos no es importante).

Se propone ahora que $T(n) \leq 12n \log n + 1$. Como es fácil demostrarlo para $n = 1$, se procederá vía inducción.

Para $n \geq 2$, se asume que la desigualdad es verdadera para $1, \ldots, n - 1$, por lo tanto:

$$\begin{aligned} T(n) &\leq 12\left\lfloor \frac{n}{2} \right\rfloor \log\left(\frac{2}{3}n\right) + 1 + 12\left\lceil \frac{n}{2} \right\rceil \log\left(\frac{2}{3}n\right) + 1 + 3n + 6 \\ &= 12n(\log n + 1 - \log 3) + 3n + 8 \\ &\leq 12n \log n - \frac{13}{2}n + 3n + 8 \leq 12n \log n + 1 \end{aligned}$$

pues $\log 3 \geq \frac{37}{24}$. □

Capítulo 4

Problema de ubicación de plantas

4.1. Antecedentes Históricos

Existen situaciones en donde las empresas deben de decidir en que lugar construir sus centros de servicio para satisfacer la demanda eficientemente. Algunos ejemplos pueden ser: fábricas, bodegas, depósitos, centros de distribución, centros de atención al cliente, o también pueden ser librerías, supermercados, estaciones de bomberos, hospitales, escuelas, etc. Tales situaciones se pueden resolver matemáticamente mediante un *modelo de ubicación de plantas*.

El problema de ubicación de plantas ha sido un tema interesante para los investigadores desde los años 1960's; ha sido estudiado desde la perspectiva del peor caso, desde el punto de vista probabilístico, el combinatorio y el heurístico (ver [11] y [26]).

A lo largo de todos estos años se han desarrollado diferentes algoritmos de aproximación para resolver el problema. El primero de todos lo propuso la investigadora Dorit S. Hochbaum [14] aproximadamente por los años 80's para resolver el caso general no métrico, el cual tenía un factor de aproximación de $O(\log(n))$.

El primer algoritmo con factor de aproximación *constante* para este problema lo propusieron Shmoys, Tardos y Aardal en 1997 [8], ellos resolvieron el problema con un factor de aproximación de 4, el cual posteriormente fue mejorado a 1.736

por Chudak y Shmoys [4]y 1.582 por Svirindeko [15].

Recientemente el problema de localización de plantas se ha podido aplicar también a problemas de diseño de redes para servidores, caches e información digital. [1, 2, 13, 22]

4.2. Definición del problema

Problema de ubicación de plantas sin límite de capacidad (PUPsC)

Instancia : Un conjunto finito de ciudades denotado por $\mathcal{D}$, un conjunto finito de zonas denotado por $\mathcal{F}$, un costo fijo $f_i \in \mathbb{R}^+$ por construir una planta en la zona $i \in \mathcal{F}$ y un costo de servicio $c_{ij} \in \mathbb{R}^+$ por servir a la ciudad $j \in \mathcal{D}$ con una planta ubicada en la zona $i \in \mathcal{F}$.

Objetivo : Encontrar un subconjunto $X \subseteq \mathcal{F}$ de zonas en las que se abrirán plantas y una asignación $\sigma : \mathcal{D} \to X$ de ciudades con plantas abiertas, tal que la suma de los costos de las plantas más los costos de servicio

$$\sum_{i \in X} f_i + \sum_{j \in \mathcal{D}} c_{\sigma(j)j}$$

sea mínima.

Comúnmente, los costos de servicio están estimados en proporción a las distancias geométricas que existen entre las zonas donde se construirá una planta y los clientes, por lo que si se seleccionan al azar 2 zonas $i, i' \in \mathcal{F}$ y dos clientes $j, j' \in \mathcal{D}$ se forma una figura de 4 lados (que representan las distancias entre las zonas y los clientes) en la que siempre se cumple que la suma de 3 de sus lados es mayor que el cuarto lado. Por esta razón, si los costos de servicio están en proporción a las distancias geométricas de los clientes y las zonas, entonces se cumple la siguiente restricción:

$$c_{ij} + c_{i'j} + c_{i'j'} \geq cij' \qquad \text{para todas } i, i' \in \mathcal{F} \text{ y } j, j' \in \mathcal{D} \tag{4.1}$$

Si esto se cumple, se dice que el problema de ubicación de plantas es *métrico*.

El problema métrico de ubicación de plantas pertenece a la clase $NP-Duro$ como se demuestra en el siguiente:

Teorema 4.1. *El problema métrico de ubicación de plantas sin límite de capacidad (PUPsC métrico) es un problema $NP-Duro$.*

Demostración. Se demostrará que el problema de la mínima cubierta por conjuntos, el cual pertenece a la clase $NP-Duro$, se transforma polinomialmente en el PUPsC.

Sea $(U, \mathcal{S})$ cualquier instancia del problema de cubierta por conjuntos de peso mínimo, en particular y sin pérdida de generalidad, con pesos unitarios.

El conjunto U es el *universo* que se debe *cubrir* por un subconjunto $\mathcal{R}$ de conjuntos de $\mathcal{S}$.

Claramente $\mathcal{D} = U$ puesto que el objetivo del problema de la cubierta por conjuntos es crear, a partir de $\mathcal{S}$, una familia finita de conjuntos que cubra en su totalidad al conjunto U y en el **PUPsC métrico** el objetivo es cubrir la demanda de los clientes $\mathcal{D}$ con un conjunto finito de plantas ubicadas en el conjunto de zonas $\mathcal{F}$; de aquí mismo se ve que $\mathcal{S} = \mathcal{F}$ pues ambos conjuntos cumplen con la misma función. También los conjuntos X y $\mathcal{R}$ son equivalentes.

Los costos de ambos problemas, se comparan de la siguiente manera: para todo $S \in \mathcal{S}$,

$$f_S = c(S) = 1$$

$$c_{Sj} = \begin{cases} 1 & \text{para } j \in S \\ 3 & \text{para } j \in U \backslash S \end{cases} \tag{4.2}$$

El costo c_{Sj} es igual a 3 para no violar la desigualdad 4.1, sin embargo puede ser reemplazado por cualquier número entre 1 y 3.

Por último, los óptimos de ambos problemas deben coincidir y en el caso del **PUPsC métrico** es:

$$\sum_{i \in X} f_i + \sum_{j \in D} c_{\sigma(j)j}$$

la primera suma es fácil de analizar pues si recordamos que $f_i = 1$ para todo $i \in \mathcal{F}$ entonces:

$$\sum_{i \in X} f_i = \overbrace{1 + 1 + \ldots + 1}^{k} = k \qquad \text{en donde } k = |X|$$

Por el término (4.1), $c_{ij} = 1$ para todos los clientes asignados, por lo tanto, la suma total del costo del problema **PUPsC métrico** es $|X| + |D|$, el cual es óptimo si $|\mathcal{R}| = |X|$, pues $|\mathcal{R}|$ es el valor óptimo del problema de la mínima cubierta de conjuntos con pesos unitarios. □

4.3. Modelo

Como en cualquier problema de programación matemática, se necesitan identificar cuales son las variables de decisión, las restricciones y la función objetivo del PUPsC.

Variables de decisión

En este problema hay dos tipos de decisiones, la primera es decidir en que zona construir la planta y la otra es decidir que ciudades se atenderán con que plantas.

Por tal motivo, existen dos tipos de variables de decisión que se definen de la siguiente manera:

$$\text{Si } x_{ij} = \begin{cases} 1 & \text{entonces la ciudad } j \text{ se asigna a la planta ubicada en la zona } i \\ 0 & \text{entonces } j \text{ no se asigna a la planta en } i \end{cases}$$

$$\text{Si } y_i = \begin{cases} 1 & \text{entonces se construye una planta en la zona } i \\ 0 & \text{entonces no se construye ninguna planta en } i \end{cases}$$

para todo $i \in \mathcal{F}$ y para todo $j \in \mathcal{D}$

Restricciones

El primer tipo de restricciones viene en la definición del problema, pues a cada ciudad se le asigna sólo una planta para satisfacer su demanda:

$$\sum_{i \in \mathcal{F}} x_{ij} = 1 \qquad \text{para todo } j \in \mathcal{D}$$

El segundo tipo de restricción tiene que ver más con un aspecto lógico, ya que no se puede asignar una ciudad a una planta que no se construirá, por tanto se deben incluir $|\mathcal{F}||\mathcal{D}|$ restricciones del siguiente tipo:

$$x_{ij} \leq y_i \qquad \text{para todo } i \in \mathcal{F}, j \in \mathcal{D}$$

El último tipo de restricciones, tiene que ver con los valores que pueden tomar las variables:

$$\begin{aligned} x_{ij} &\in \{0,1\} \qquad \text{para todo } i \in \mathcal{F}, j \in \mathcal{D} \\ y_i &\in \{0,1\} \qquad \text{para todo } i \in \mathcal{F} \end{aligned}$$

Función Objetivo

Tal cual lo muestra la definición, el objetivo del problema es minimizar los costos totales de servicio y de construcción, por lo que la función objetivo sería:

$$\text{minimizar } z = \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij} + \sum_{i \in \mathcal{F}} f_i y_i$$

El modelo queda entonces así:

$$\begin{aligned} \text{minimizar} \quad & \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij} + \sum_{i \in \mathcal{F}} f_i y_i & \\ \text{sujeto a} \quad & & \\ & x_{ij} \leq y_i & \text{para todo } i \in \mathcal{F}, j \in \mathcal{D} \\ & \sum_{i \in \mathcal{F}} x_{ij} = 1 & \text{para todo } j \in \mathcal{D} \\ & x_{ij}, y_i \in \{0,1\} & \text{para todo } i \in \mathcal{F}, j \in \mathcal{D} \end{aligned}$$

Una de las técnicas usadas para resolver cualquier problema entero, es relajar la restricción de integralidad para que se vuelva lineal y así obtener información valiosa sobre el problema, por ejemplo una cota superior para el óptimo.

Después de relajar, el modelo queda así:

$$
\begin{aligned}
\text{minimizar} \quad & \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{D}} c_{ij}x_{ij} + \sum_{i\in\mathcal{F}} f_i y_i \\
\text{sujeto a} \quad & \\
& x_{ij} \leq y_i && \text{para todo } i \in \mathcal{F}, j \in \mathcal{D} \\
& \sum_{i\in\mathcal{F}} x_{ij} = 1 && \text{para todo } j \in \mathcal{D} \\
& x_{ij}, y_i \geq 0 && \text{para todo } i \in \mathcal{F}, j \in \mathcal{D}
\end{aligned}
$$

El problema dual asociado, es el siguiente:

$$
\begin{aligned}
\text{maximizar} \quad & \sum_{j\in\mathcal{D}} \nu_j \\
\text{sujeto a} \quad & \\
& \nu_j - w_{ij} \leq c_{ij} && \text{para todo } i \in \mathcal{F}, j \in \mathcal{D} \\
& \sum_{j\in\mathcal{D}} w_{ij} \leq f_i && \text{para todo } i \in \mathcal{F} \\
& w_{ij} \geq 0 && \text{para todo } i \in \mathcal{F}, j \in \mathcal{D}
\end{aligned}
$$

La variable ν_j con $j \in \mathcal{D}$ se interpreta como el presupuesto de la ciudad j y la variable w_{ij} con $i \in \mathcal{F}, j \in \mathcal{D}$ representa la aportación que hace la ciudad j para que se abra una planta en la zona i.

4.4. Algoritmos de aproximación

Aunque los algoritmos propuestos por Shmoys, Tardos, Chudak, Aardal y Svirindeko alcanzaron un factor de aproximación de 1.582, son considerados poco eficientes debido al excesivo uso de la programación lineal.

Por otro lado, en el año 2001 [16], Jain y Vazirani propusieron un nuevo algoritmo con un factor de aproximación de 3 y un tiempo de ejecución de $O(m \log_2(m))$. El algoritmo de Jain y Vazirani, de ahora en adelante nombrado *JV*, calcula la solución primal y la dual al mismo tiempo.

4.4.1. Descripción del algoritmo de Jain-Vazirani

Se definen los siguientes conjuntos:

- Y: que consta de todas las zonas que aún no están *tentativamente* seleccionadas para abrir un planta en ellas.
- V: que contiene a todas aquellas zonas que sí están *tentativamente* seleccionadas.
- U: formado por todas las ciudades que aún no han sido *conectadas* o *asignadas* a ninguna planta.

nótese también que la variable w_{ij} puede verse como que es mayor o igual que máx$\{0, \nu_j - c_{ij}\}$, para todo $i \in \mathcal{F}, j \in \mathcal{D}$.

El algoritmo consiste principalmente en incrementar de manera constante el valor de las variables duales (que inicialmente valen cero). La variable ν_j y sus correspondientes variables w_{ij} se congelan cuando la ciudad $j \in \mathcal{D}$ es tentativamente asignada a alguna planta, lo cual ocurre en alguna de las dos situaciones:

1. $\nu_j = c_{ij}$ para $i \in V$ y $j \in U$.
2. $\sum_{j \in \mathcal{D}} w_{ij} = f_i$ para $i \in Y$

Si se cumple la primera, entonces $\sigma(j) = i$ y se congela ν_j, y si se cumple la otra entonces $Y = Y \backslash \{i\}$ y $V = V \cup \{i\}$ y para todos los ciudades $j \in U$ que cumplan con que $\nu_j \geq c_{ij}$ se hace $\sigma(j) := i$ y se congela ν_j.

Ahora sea E el conjunto de todas las parejas $\{i, i'\}$ con $i \neq i'$ e $i, i' \in V$, tales que existe algún j con $\sigma(j) \in V$ con $w_{ij} > 0$ y $w_{i'j} > 0$. Se escoge el conjunto estable maximal X de la gráfica formada por los conjuntos V, E y por último, se abren todas las plantas que pertenecen a X y a todas las ciudades que no están asignadas a ninguna planta de X ($\sigma(j) \notin X$) se asignan a algún vecino abierto de $\sigma(j)$ en (V, E).

Algoritmo de Jain-Vazirani

Entrada : Una instancia $(\mathcal{D}, \mathcal{F}, (f_i)_{i\in\mathcal{F}}, (c_{ij})_{i\in\mathcal{F}, j\in\mathcal{D}})$ del PUPsC.

Salida : Una solución $X \subseteq \mathcal{F}$ y $\sigma : \mathcal{D} \to X$.

① **Inicializar** $X = \emptyset$, $U = \mathcal{D}$ y $Y = \mathcal{F}$.

② **Definir** $t_1 = \text{mín}\{c_{ij} | i \in V, j \in U\}$.
Definir $t_2 = \text{mín}\{\tau | \exists i \in Y | \omega(i, \tau) = f_i\}$ donde
$\omega(i, \tau) = \sum_{j\in U} \text{máx}\{0,\ \tau - c_{ij}\} + \sum_{j\in\mathcal{D}\setminus U} \text{máx}\{0,\ \nu_j - c_{ij}\} = f_i$.
Definir $t = \text{mín}\{t_1, t_2\}$

③ **Para** $i \in Y$ con $\omega(i, t) = f_i$ **Hacer**:
$Y = Y\setminus\{i\}$
Si hay un $i' \in X$ y $j \in \mathcal{D}\setminus U$ con $\nu_j > c_{ij}$ y $\nu_j > c_{i'j}$
Entonces $\varphi(i) = i'$
En otro caso $\varphi(i) = i$ y $X = X \cup \{i\}$
Para $j \in U$ con $c_{ij} \leq t$ **Hacer**:
$\sigma(j) = \varphi(i)$, $\nu_j = t$ y $U = U\setminus\{j\}$

④ **Para** $i \in V$, $j \in U$ con $c_{ij} = t$ **Hacer**:
$\sigma(j) = \varphi(i)$, $\nu_j = t$ y $U = U\setminus\{j\}$

⑤ **Si** $U \neq \emptyset$ **entonces ir al** paso 2.

Teorema 4.2. *Para instancias métricas I, el algoritmo de* JV *resuelve el problema con un factor de aproximación de* 3 *y puede ejecutarse para correr en un tiempo de* $O(m \log_2(m))$*, donde* $m = |\mathcal{F}||\mathcal{D}|$.

Demostración: Para empezar, el algoritmo entrega una solución factible, por lo tanto:

$$\sum_{j\in\mathcal{D}} \nu_j \leq OPT(I)$$

.

Por otro lado, los costos de servicio y de apertura de las plantas se definen de la siguiente manera:

$$c_F(X) := \sum_{i \in X} f_i \qquad \text{y} \qquad c_S(X) := \sum_{j \in D} c_{\sigma(j)j}$$

.

Se demostrará entonces que:

$$c_F(X) + c_S(X) \leq 3OPT(I)$$

,

o dicho de otra manera

$$\sum_{i \in X} f_i + \sum_{j \in D} c_{\sigma(j)j} \leq 3OPT(I) \tag{4.3}$$

Para cada zona i, todas las ciudades j con $w_{ij} > 0$, están conectadas a la planta construida en i, o dicho de otro modo, $\sigma(j) = i$, además se tiene que $f_i = \sum_{j \in \mathcal{D}} w_{ij}$.

Entonces, la primera suma de la ecuación (4.3) queda:

$$\sum_{i \in X} f_i = \sum_{i \in X} \sum_{j \in \mathcal{D}} w_{ij} \leq 3 \sum_{i \in X} \sum_{j \in \mathcal{D}} w_{ij}$$

Para la segunda suma de (4.3), se propone que el costo de servicio para cada cliente j es a lo más $3(\nu_j - w_{\sigma(j)j})$, es decir:

$$\sum_{j \in D} c_{\sigma(j)j} \leq \sum_{j \in \mathcal{D}} 3(\nu_j - w_{\sigma(j)j}) = 3 \sum_{j \in \mathcal{D}} \nu_j - 3 \sum_{j \in \mathcal{D}} w_{\sigma(j)j}$$

Para demostrarlo, se analizarán dos casos. Si $c_{\sigma(j)j} = \nu_j - w_{\sigma(j)j}$, es trivial. Por otro lado, $c_{\sigma(j)j} > \nu_j$ y $w_{\sigma(j)j} = 0$, lo que significa que $\varphi(i) \neq i$ cuando j se conectó a $\varphi(i)$ en el paso 3 o 4 del algoritmo, entonces hay una zona $i \in \mathcal{F} \backslash (Y \cup X)$ que no tiene plantas con $c_{ij} \leq \nu_j$ y un cliente j' con $w_{ij'} > 0$ y $w_{\sigma(j)j'} > 0$, y por lo tanto $c_{ij'} = \nu_{j'} - w_{ij'} < \nu_{j'}$ y $c_{\sigma(j)j'} = \nu_{j'} - w_{\sigma(j)j'} < \nu_{j'}$. Nótese que $\nu_{j'} \leq \nu_j$, pues j' se conectó a $\sigma(j)$ antes que j. Se concluye que:

$$c_{\sigma(j)j} \leq c_{\sigma(j)j'} + c_{ij'} + c_{ij} < \nu_{j'} + \nu_{j'} + \nu_j \leq 3\nu_j = 3(\nu_j - w_{\sigma(j)j})$$

.

Sumando ahora los dos costos, de servicio y de apretura de plantas, se tiene:

$$\begin{aligned} c_F(X) + c_S(X) = & \sum_{i \in X} f_i \quad + \sum_{j \in D} c_{\sigma(j)j} \\ \leq & 3 \sum_{i \in X} \sum_{j \in \mathcal{D}} w_{ij} \; + 3 \sum_{j \in \mathcal{D}} (\nu_j - w_{\sigma(j)j}) \\ = & 3 \sum_{j \in \mathcal{D}} \big(\sum_{i \in X} w_{ij} + \nu_j - w_{\sigma(j)j} \big) \end{aligned}$$

Luego, $\sum_{i \in X} w_{ij} = w_{\sigma(j)j}$, puesto que $w_{ij} = 0$ siempre que el cliente j no esté asignado a la planta construida en i (es decir $i \neq \sigma(j)$), por lo que se sigue que:

$$\begin{aligned} 3 \sum_{j \in \mathcal{D}} \big(\sum_{i \in X} w_{ij} + \nu_j - w_{\sigma(j)j} \big) = & 3 \sum_{j \in \mathcal{D}} \big(w_{\sigma(j)j} + \nu_j - w_{\sigma(j)j} \big) \\ = & 3 \sum_{j \in \mathcal{D}} \nu_j \\ \leq & 3OPT(I) \end{aligned}$$

Para medir el tiempo de ejecución del algoritmo, se analizará por separado cada uno de los pasos del algoritmo.

① Como se puede ver en el algoritmo, en este paso sólo se le asignan valor a 3 variables, lo cual se ejecuta en tiempo constante al que llamaremos c_1, por lo que el tiempo total del paso 1 es $T_1 = c_1$.

② Este paso se compone de tres partes, en las cuales se calcula el valor de t_1, t_2 y t respectivamente:

- $t_1 = \text{mín}\{c_{ij} | i \in V, j \in U\}$, y la manera más rápida para calcularlo es simplemente hacer a lo más $m = |\mathcal{F}||\mathcal{D}|$ comparaciones para conservar en cada paso el valor más chico. Sin embargo por conveniencia,

se ordenan primero a las $c'_{ij}s$ de menor a mayor con el algoritmo *Merge-Sort*, lo cual lleva un tiempo de ejecución de $O(m \log_2 m)$, y como encontrar el valor mínimo en un conjunto ordenado es constante, entonces el tiempo total para calcular t_1 es $O(m \log_2 m)$.

- $t_2 = \text{mín}\{\tau | \exists i \in Y | \omega(i, \tau) = f_i\}$, sin embargo viéndolo de la siguiente forma es más fácil calcularlo:

$$t_2 = \text{mín}\left\{ \frac{t_2^i}{|U_i|} | i \in Y \right\}$$

en donde:

$$\begin{aligned} U_i &= \{j \in U | c_{ij} \leq t\} \\ t_2^i &= f_i + \sum_{j \in \mathcal{D} \setminus U : \nu_j > c_{ij}} (c_{ij} - \nu_j) + \sum_{j \in U_i} c_{ij} \end{aligned}$$

Una vez calculados, t_2^i y U_i se mantienen igual en cada iteración, lo que nos indica que t_2 se mantiene igual hasta que alguno de sus componentes cambie y esto sucede cuando algún cliente es conectado o cuando $t = c_{ij}$ para algún $j \in U$. Como esto se hace a lo más para cada $i \in Y$ y $j \in U$, entonces actualizar los valores de t_2, t_2^i y $|U_i|$ lleva un tiempo de $O(m)$. Sólo hay que notar que para que esto sea posible, hay que cambiar la definición de t_1 en el paso 2 como: $t_1 = \text{mín}\{c_{ij} | i \in Y, j \in U\}$.

- Calcular el valor de t lleva tiempo constante.

El tiempo total de ejecución del paso 2 es por lo tanto:

$$T_2 = O(m \log_2(m))$$

.

③ El paso 3 se ejecuta sólo si $t_2 = t$ y la condición *si-entonces* se ejecuta en $O(|\mathcal{D}|)$ ya que $i' \in X$, $j \in \mathcal{D} \setminus U$ y $\nu_j > c_{i'j}$ implica que $\sigma(j) = i'$. Por lo tanto el tiempo de ejecución del paso 4 es: $T_4 = O(|\mathcal{D}|)$.

④ El paso 4 sólo se ejecuta si $t_1 = t$ por lo que su tiempo de ejecución es constante $T_3 = c_3$.

⑤ Finalmente, el paso 5 se ejecuta en tiempo constante:

$T_5 = O(1)$.

El tiempo estimado de ejecución del algoritmo es entonces:

$$T_1 + T_2 + T_3 + T_4 + T_5 = O(m \log_2(m))$$

□

4.4.2. Descripción del algoritmo de Ajuste Dual

En el año 2003, Jain, Mahdain, Markakais, Sabery y Vazirani [17] propusieron un mejor algoritmo en cuanto al tiempo de ejecución, al cual se llamará algoritmo de *Ajuste-Dual*.

El algoritmo de Ajuste-Dual es prácticamente el mismo algoritmo JV pero en este caso las aportaciones de las ciudades a las plantas difieren un poco del algoritmo JV:

- Si la ciudad j está asignada a alguna planta ubicada en la zona i', entonces ofrecerá como aportación para la apertura de una planta en i (cerrada) lo que se ahorraría por cambiar de planta (máx$\{0, c_{i'j} - c_{ij}\}$).
- Por el contrario, si la ciudad aún no ha sido asignada entonces aporta lo mismo que en el algoritmo JV (máx$\{0, \nu_j - c_{ij}\}$).

Inicialmente todas las ciudades están no asignadas ($U = \mathcal{D}$) y todas las plantas están cerradas ($X = \emptyset$) y el tiempo t y todas las variables duales valen cero.

Luego mientras que $U \neq \emptyset$ se incrementa el valor ν_j para todo $j \in U$ y el valor de t al mismo tiempo y en la misma proporción; es decir, $\nu_j = t$ para todo $j \in U$ hasta que alguno de los siguiente eventos ocurra:

1. $\nu_j = c_{ij}$ para alguna $j \in U$ y alguna $i \in X$. Si esto ocurre, se asigna j a i y se congela el valor de ν_j.
2. La suma de las aportaciones de las ciudades asignadas y no asignadas para alguna planta en $i \in \mathcal{F}$ cerrada es igual a su costo de apertura. Si esto sucede se abre la planta en i y todas las ciudades asignadas y no asignadas que tengan una aportación no negativa a esa planta son asignadas a ella.

Formalmente el algoritmo queda así:

Algoritmo de Ajuste-Dual

Entrada : Una instancia $(\mathcal{D}, \mathcal{F}, (f_i)_{i\in\mathcal{F}}, (c_{ij})_{i\in\mathcal{F}, j\in\mathcal{D}})$ del PUPsC.

Salida : Una solución $X \subseteq \mathcal{F}$ y $\sigma : \mathcal{D} \to X$.

① **Inicializar** $X = \emptyset$, $U = \mathcal{D}$.

② **Definir** $t_1 = \text{mín}\{c_{ij} | i \in X, j \in U\}$.
Definir $t_2 = \text{mín}\{\tau | \exists i \in \mathcal{F}\backslash X | \omega(i,\tau) = f_i\}$ donde
$\omega(i,\tau) = \sum_{j\in U} \text{máx}\{0,\ \tau - c_{ij}\} + \sum_{j\in\mathcal{D}\backslash U} \text{máx}\{0,\ c_{\sigma(j)j} - c_{ij}\} = f_i$.
Definir $t = \text{mín}\{t_1, t_2\}$

③ **Para** $i \in \mathcal{F}\backslash X$ con $\omega(i,t) = f_i$ **Hacer**:
$X = X \cup \{i\}$.
Para $j \in \mathcal{D}\backslash U$ con $c_{ij} < c_{\sigma(j)j}$ **Hacer**: $\sigma(j) = i$.
Para $j \in U$ con $c_{ij} < t$ **Hacer**: $\sigma(j) := i$.

④ **Para** $i \in X$, $j \in U$ con $c_{ij} \leq t$ **Hacer**:
$\sigma(j) = i$, $\nu_j = t$ y $U = U\backslash\{j\}$.

⑤ **Si** $U \neq \emptyset$ **entonces ir al** paso 2.

Teorema 4.3. *El algoritmo de arriba calcula una solución dual y una solución factible primal y puede ejecutarse en un tiempo de* $O(|\mathcal{F}|^2|\mathcal{D}|)$ *[19]*

Para entender como funciona el algoritmo se resolverá con él un ejemplo numérico que consta de 4 zonas para construir plantas y 7 clientes que atender.

La matriz C de costos de asignación es la siguiente:

$$\mathbf{C} = \begin{pmatrix} 2 & 2 & 1 & 5 \\ 2{,}5 & 6 & 5{,}5 & 8 \\ 4 & 3{,}5 & 10 & 4 \\ 8 & 3 & 7 & 3 \\ 4 & 7 & 3 & 2 \\ 5{,}5 & 5 & 5 & 9 \\ 7 & 1 & 4 & 4 \end{pmatrix}$$

y los costos de apertura de cada zona son:

$$\mathbf{F} = \begin{pmatrix} 30 & 25 & 18 & 35 \end{pmatrix}$$

El primer paso del algoritmo es hacer $t = 0$, $X = \emptyset$ y $\mathcal{D} = U$, luego como lo indica el procedimiento, se necesita calcular el valor de t y para esto se necesitan calcular, t_1 y t_2:

$$t_1 = \text{mín}\{c_{ij} | i \in X, j \in U\}$$

en este caso se asume que $t_1 = \infty$ pues $X = \emptyset$.

Por otro lado, para t_2:

$$t_2 = \text{mín}\{\tau | \exists i \in \mathcal{F} \backslash X | \omega(i, \tau) = f_i\}$$

donde

$$\omega(i, \tau) = \sum_{j \in U} \text{máx}\{0,\ \tau - c_{ij}\} + \sum_{j \in \mathcal{D} \backslash U} \text{máx}\{0,\ c_{\sigma(j)j} - c_{ij}\} = f_i$$

Al contrario de t_1, como $X = \emptyset$ entonces se tiene que encontrar al valor de τ necesario para abrir una planta en cada una de las 4 zonas de $\mathcal{F}$.

Para la zona 1, $\tau = \frac{60}{7}$, para la segunda zona τ=7.5, en la tercera τ=7.25 y para la cuarta, τ=10. El valor mínimo de τ es 7.25, lo cual quiere decir que t_2 =7.25 y por lo tanto $t = t_2$.

El paso 3 del algoritmo indica que hay que abrir una planta en la zona 3, es decir $X = X \cup \{3\}$. Además todos los clientes $j \in U$ que hicieron una aportación a la apertura de la planta 3 son asignados a ella, es decir todos aquellos clientes tales que $c_{3j} < t$.

Como se puede ver en la matriz de costos $\mathbf{C}$, todos los costos de la columna 3 son menores a 7.25 a excepción de $c_{33} = 10$, por lo tanto:

$$\sigma(j) = 3 \text{ para } j \in \{1, 2, 4, 5, 6, 7\}$$

El paso 4 indica que $\nu_j = t$ =7.25 para $j \in \{1, 2, 4, 5, 6, 7\}$ y que $U = U \backslash \{1, 2, 3, 4, 6, 7\}$.

Como lo indica el paso 5, $U \neq \emptyset$ por lo que se regresa al paso 2.

De nuevo en el paso 2, $t_1 = c_{33} = 10$ y t_2 es el mínimo valor de τ necesario para abrir una planta en las zonas 1, 2 y 4.

Para la zona 1, $\tau = 29$, para la zona 2, τ =21.5 y para la 3, $\tau = 34$, por lo que t_2 =21.5 y por último $t = \text{mín}\{t_1, t_2\} = 10$.

El paso 3 no aplica y al paso 4 indica que $\sigma(3) = 3$, $\nu_3 = 10$ y $U = U \backslash \{3\}$.

Como $U = \emptyset$ entonces el algoritmo concluye con que se debe abrir la planta en la zona 3 y que todos los clientes deben ser asignados a ella.

La solución primal es entonces:

$$x_{31} = x_{32} = x_{33} = x_{34} = x_{35} = x_{36} = x_{37} = 1$$

$$y_3 = 1$$

y todas las demás variables valen 0, la cual tiene un valor en la función objetivo de:

$$c_{31} + c_{32} + c_{33} + c_{34} + c_{35} + c_{36} + c_{37} + f_3 = 53{,}5$$

4.5. ¿Qué tanto se puede mejorar?

El algoritmo de *Ajuste-Dual* entrega una solución dual que por lo regular no es factible, esto se debe a que al abrir una planta se dejan de considerar las aportaciones de las ciudades a esa planta, lo que puede ocasionar que al final se sobrepase el valor de apertura de la planta.

Sin embargo en el año 2003, Jain et al, encontraron que el vector $(\frac{\nu_j}{\gamma}, j \in \mathcal{D})$

resulta ser una solución dual factible y además demostraron que el número γ es el factor de aproximación del algoritmo de *Ajuste-Dual* [17].

Para encontrar el valor de γ, se utilizará la siguiente definición:

Definición 4.1. *Dado ν_j con $j \in (1, \ldots |\mathcal{D}|)$, una planta en la zona i es llamada* a lo más limitada por γ *si y sólo si:*

$$\sum_j \text{máx}\{\nu_j/\gamma - c_{ij}, 0\} \leq f_i \tag{4.4}$$

Usando la definición anterior, es fácil ver que la solución es factible si y sólo si todas las plantas son a lo más limitadas por γ. Nótese que en la suma (4.4) sólo se necesitan ciudades tales que $\nu_j/\gamma - c_{ij} > 0$.

Gráficamente, una estrella es un conjunto de vértices, en los que existe un vértice llamado *centro* tal que todos los demás vértices se conectan a él, pero los demás vértices no se conectan entre sí.

Considérese la estrella S que consta de alguna zona F con costo de apertura f, y d ciudades que cumplen con que $\nu_j/\gamma - c_{ij} > 0$. Sea d_j el costo de servicio entre la zona F y la ciudad j, y ν_j el presupuesto de la ciudad j al final de la ejecución del algoritmo que sin pérdida de generalidad cumplen con:

$$\nu_1 \leq \nu_2 \leq \ldots \leq \nu_d$$

.

Se necesitan más variables para poder representar la ejecución del algoritmo. Para esto, se identificará la *situación* de cada ciudad k ($1 \leq k \leq d$) antes de ser asignada por primera vez, es decir, cuando $t = \nu_k - \epsilon$ para una ϵ muy pequeña.

Sea $j < k$ para la ciudad j, hay dos situaciones posibles justo antes de que la ciudad k sea asignada por primera vez:

1. La ciudad j ya estaba asignada a alguna planta con un costo de d_j.
2. La ciudad j aún no ha sido asignada a ninguna planta, entonces $\nu_j = \nu_k$

Con esta información se puede definir una variable que nos indique la situación de las ciudades $1, 2, \ldots, j, \ldots, k-1$ al tiempo $t = \nu_k - \epsilon$.

$$r_{j,k} = \begin{cases} d_j & \text{Si } j \text{ ya estaba conectada a la planta construida en la zona } F \in \mathcal{F} \\ \nu_k & \text{en otro caso, i.e. si } \nu_j = \nu_k \end{cases}$$

Se describen ahora desigualdades válidas para esas variables. Primero, para $j = 1, \ldots, d-2$,

$$r_{j,j+1} \geq r_{j,j+2} \geq \ldots \geq r_{j,d} \tag{4.5}$$

porque el costo de servicio decrece si los ciudades son re-conectados.

Luego en el tiempo $t = \nu_k - \epsilon$, la aportación que la ciudad j otorga para que la planta se construya en F es igual a:

$$\begin{array}{rl} \text{máx}\{r_{j,k} - d_j, 0\} & \text{si } j < k, \text{ y} \\ \text{máx}\{t - d_j, 0\} & \text{si } j \geq k. \end{array}$$

Para obtener una solución factible, entonces se debe cumplir que:

$$\sum_{j=1}^{k-1} \text{máx}\{0, r_{j,k} - d_j\} + \sum_{l=k}^{d} \text{máx}\{0, \nu_k - d_j\} \leq f. \tag{4.6}$$

Para que la instancia sea métrica, se considera las siguiente restricción:

$$r_{j,k} + d_j + d_k \geq \nu_k \qquad \text{para} \qquad 1 \leq j < k \leq d. \tag{4.7}$$

Por último, se necesita que:

$$\sum_{j=1}^{d} \nu_j/\gamma - d_j \leq f$$

.

Despejando γ se obtiene:

$$\gamma \geq \frac{\sum_{j=1}^{d} \nu_j}{f + \sum_{j=1}^{d} d_j}.$$

Con lo anterior se puede generar el siguiente modelo llamado *Programa Lineal revelador de factores de aproximación*:

$$\text{maximizar } \frac{\sum_{j=1}^{d} \nu_j}{f + \sum_{j=1}^{d} d_j}$$

sujeto a

$$\begin{aligned} \nu_j &\leq \nu_{j+1} && (1 \leq j < d) \\ r_{j,k} &\geq r_{j,k+1} && (1 \leq j < k < d) \\ r_{j,k} + d_j + d_k &\geq \nu_k && (1 \leq j < k \leq d) \end{aligned}$$

$\sum_{j=1}^{k-1} \text{máx}\{0, r_{j,k} - d_j\} +$

$$\sum_{l=k}^{d} \text{máx}\{0, \nu_k - d_l\} \leq f \quad (1 \leq k \leq d)$$

$$\begin{aligned} \sum_{j=1}^{d} d_j &> 0 \\ f &\geq 0 \\ \nu_j, d_j &\geq 0 \quad (1 \leq j \leq d) \\ r_{j,k} &\geq 0 \quad (1 \leq j < k \leq d) \end{aligned}$$

El modelo puede ser fácilmente modificado para transformarlo en uno lineal, añadiendo la siguiente restricción:

$$f + \sum_{j=1}^{d} d_j \leq 1$$

El valor de γ se calcula numéricamente utilizando diversos valores para d. Jain y Vazirani aproximaron el valor de γ a 1.61.

Sin embargo, también se necesitan encontrar factores de aproximación para diferentes valores de f y d_j. Esto se logra con el siguiente modelo:

$$\text{maximizar } \frac{\sum_{j=1}^{d} \nu_j - \gamma_F f}{\sum_{j=1}^{d} d_j}$$

sujeto a

$$\begin{aligned} \nu_j \leq \quad & \nu_{j+1} \quad (1 \leq j < d) \\ r_{j,k} \geq r_{j,k+1} \quad & (1 \leq j < k < d) \\ r_{j,k} + d_j + d_k \geq \quad & \nu_k \quad (1 \leq j < k \leq d) \end{aligned}$$

$\sum_{j=1}^{k-1} \text{máx}\{0, r_{j,k} - d_j\} +$

$$\sum_{l=k}^{d} \text{máx}\{0, \nu_k - d_l\} \leq f \quad (1 \leq k \leq d)$$

$$\sum_{j=1}^{d} d_j > 0$$

$$\begin{aligned} f &\geq 0 \\ \nu_j, d_j &\geq 0 \quad (1 \leq j \leq d) \\ r_{j,k} &\geq 0 \quad (1 \leq j < k \leq d) \end{aligned}$$

El siguiente lema garantiza que la solución óptima del modelo es en realidad el factor de aproximación del algoritmo *Ajuste-Dual*.

Lema 4.1. *Sea $\gamma_F \geq 1$ y sea γ_S el supremo de los valores óptimos del modelo de arriba para todas las $d \in \mathbb{N}$. Sea $X^* \subseteq \mathcal{F}$ cualquier solución de I una instancia dada, entonces el costo de la solución generada por el algoritmo* Ajuste-Dual *en la instancia I es a lo más $\gamma_F c_F(X^*) + \gamma_S c_S(X^*)$ [31].*

Utilizando el lema anterior, se encontraron tres valores para γ_S diferentes dependiendo del valor de γ_F, dando lugar al siguiente lema:

Lema 4.2. *Consideremos el problema lineal* revelador de factores *para alguna $d \in \mathbb{N}$*

1. *(Vygen 2005 [31]) Para $\gamma_F = 1$, $\gamma_S = 2$*
2. *(Jain 2003 [17]) Para γ_F=1.61, γ_S=1.61*
3. *(Mahdain, Ye y Zhang [23]) Para γ_F=1.11, γ_S=1.78*

Mezclando los dos lemas anteriores, se obtiene el siguiente corolario:

Corolario 4.1. *Sea* $(\gamma_F, \gamma_S) \in \{$*(1,2),(1.61,1.61),(1.11,1.78)*$\}$*, sea I una instancia del problema métrico de ubicación de plantas sin límite de capacidad y sea* $X^* \subseteq \mathcal{F}$ *cualquier solución. Entonces el costo producido por el algoritmo de Ajuste-Dual es a lo más* $\gamma_F c_F(X^*) + \gamma_S c_S(X^*)$

Ahora se definirán dos técnicas útiles para mejorar el factor de aproximación de los algoritmos descritos arriba.

GREEDY AUGMENTATION

La técnica *GA* o *Greedy Augmantation* de un conjunto de zonas $X \in \mathcal{F}$ consiste en seleccionar iterativamente un elemento $i \in \mathcal{F}$ maximizando el cociente $\frac{g_X(i)}{f_i}$ y luego hacer $X = X \cup \{i\}$ hasta que $g_X(i) \leq f_i$ para toda $i \in \mathcal{F}$. En donde:

$$g_X(i) = c_S(X) - c_S(X \cup \{i\})$$

para una solución $X \subseteq \mathcal{F}$ y una planta $i \in \mathcal{F}$.

SCALING

La técnica S o *Scaling* consiste en multiplicar todas las plantas $i \in \mathcal{F}$ por un valor $\delta > 0$ para luego aplicar el algoritmo en cuestión a la instancia modificada y después multiplicar los costos por $\frac{1}{\delta}$.

El siguiente lema y teorema servirán para mejorar el factor de aproximación de los algoritmos mencionados en este capítulo.

Lema 4.3. *(Charikar y Guha [3]) Sea $\emptyset \neq X, X^* \subseteq \mathcal{F}$. Aplicar la técnica* GA *a X para obtener un conjunto $Y \supseteq X$. Entonces:*

$$c_F(Y)+c_S(Y) \leq c_F(X) \quad +c_F(X^*) \ln \left(\text{máx} \left\{ 1, \frac{c_S(X) - c_S(X^*)}{c_F(X^*)} \right\} \right) + c_F(X^*) + c_S(X^*)$$

Teorema 4.4. *Supóngase que existen constantes positivas $\beta, \gamma_S, \gamma_F$ y un algoritmo A, el cual para cada instancia calcula una solución X tal que:*

$$\beta c_F(X) + c_S(X) \leq \gamma_F c_F(X^*) + \gamma_S(X^*)$$

para cada $\emptyset \neq X^* \subseteq \mathcal{F}$ y sea $\delta \geq \frac{1}{\beta}$.

Entonces, aplicar la técnica S con valor δ y luego aplicar la técnica GA al resultado para obtener una solución con un costo a lo más de:

$$\text{máx} \left\{ \frac{\gamma_F}{\beta} + \ln(\beta\delta), 1 + \frac{\gamma_S - 1}{\beta\delta} \right\}$$

veces el óptimo

Usando el corolario 4.1 se puede aplicar el teorema 4.4 al algoritmo *Ajuste-Dual* con $\beta = \gamma_F = 1$, $\gamma_S = 2$ y δ=1.76 obteniendo así una factor de aproximación de 1.57. El siguiente corolario indica que puede mejorarse aún más:

Corolario 4.2. *Si se aplican las técnicas* GA *y* S *con* δ*=1.504,* $\beta = 1$*,* γ_F*=1.11 y* γ_S*=1.78 al algoritmo* Ajuste-Dual *entonces el factor de aproximación es de 1.52 [23]*

El factor de aproximación 1.52, es el mejor conocido hasta la fecha para el problema métrico de ubicación de plantas sin límite de capacidad, sin embargo es interesante saber qué tanto se puede disminuir este factor.

Sea α la solución de la ecuación $\alpha + 1 = \ln(\frac{2}{\alpha})$, entonces $\alpha \approx 0{,}46305$, si restringimos a que los costos de servicio estén en el intervalo [1,3] entonces los siguientes teoremas sugieren que el problema *PUPsC* tiene un algoritmo con factor de aproximación de $1 + \alpha$.

Teorema 4.5. *Consideremos el problema* PUPsC *(no métrico) restringido a costos de servicio en el intervalo [1,3] entonces el problema tiene un algoritmo con factor de aproximación* $1 + \alpha + \epsilon$ *con* $0 < \epsilon < 1$ *[19]*

Teorema 4.6. *Si existe algún* $\epsilon > 0$ *y un algoritmo de aproximación con factor de* $1 + \alpha - \epsilon$ *para el* PUPsC métrico *entonces P=NP [31]*

Conclusiones

El problema de localización de plantas, pertenece a la clase **NP-Duro**, lo que quiere decir que si se requiere una solución exacta, los algoritmos existentes tardarían mucho tiempo para instancias grandes, por lo que si se aplica a una situación real la decisión debe ser tomada con mayor rapidez.

Para solucionar esta problemática, se cuenta con el algoritmo de ajuste dual que tiene un margen de error de 1.52 y que además es el mejor conocido a la fecha.

La pregunta es, ¿tiene sentido seguir investigando para encontrar un mejor algoritmo?, la respuesta es sí, ya que si en algún momento se encuentra uno con un factor de aproximación menor a 1.49 entonces $P = NP$ lo que incluso implicaría que existe un algoritmo óptimo de tiempo polinomial para todos los problemas $NP - Duros$.

Bibliografía

[1] M. Andrews and L. Zang. The access network design problem. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, 1998.

[2] G. Italiano Deng X B. Li, M. Golin and K. Soharby. On the optimal placement of web proxies in the internet. In *Proceedings of IEEE INFOCOM'99, págs 1282-1290*, 1999.

[3] M. Charikar and S. Guha. *Improved combinatorial algorithms for the facility location and k-median problems.* SIAM Journal on Computing vol. 34, págs 803-824, 2005.

[4] F.A. Chudak and D.B. Shmoys. *Improved approximation algorithms for the uncapacited facility location problem.* SIAM Journal on Computing vol. 33, págs 1-25, 2003.

[5] S.A. Cook. *The complexity of theorem proving procedures.* Proceedings of the 3rd Annual ACM Symposium on the theory of Computing. págs 151-158, 1971.

[6] R.J. Dakin. *A tree-search algorithm for mixed integer programming problems.* Mathematical and Phisical Sciences, Computer Journal, Volume 8, 1965.

[7] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[8] É. Tardos D.B. Shmoys and K. Aardal. On Approximation algorithms for facility location problems. In *Proceedings of the 29th annual ACM Symposium in theory of Computing*, 1997, págs 265-274.

[9] L.R. Ford and Fulkerson D.R. *Maximal flow though a network.* Canadian Journal of Mathematics 8, 1956.

[10] Robert S. Garfinkel. *Integer Programming.* Wiley-Interscience, 1972.

[11] Nemhauser G.L. and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons Inc., 1990.

[12] R Gomory. *An Algorithm for Integer Solutions to Lineal Programming*. Princeton IBM Mathematical Research Report, 1958.

[13] S. Guha and S. Khuller. *Greedy strikes back: Improved facility locations algorithms*. Journal of algorithms vol. 31, págs 228-248, 1999.

[14] D.S. Houchbaum. *Heuristics for the fixed cost median problem*. Mathematical Programming 22, págs 148-162, 1982.

[15] M. Svirindeko. On An improved approximation algorithm for the metric uncapacited facility location problem. In *Proceedings of 9th Internacional IPCO Conference*, 2002, págs 240-257.

[16] K. Jain and V.V. Vazirani. *Approximation algorithms for metric facility locations and k-median problems using primal-dual schema and Lagrangian relaxation*. Journal of the ACM 48, págs 274-296, 2001.

[17] Mahdian M. Markakis E. Saberi A. y Vazirani V.V. Jain, K. *Greedy facility location algorithms analized using dual fitting with factor-revealing LP*. págs 795-824. Journal of the ACM 50, 2003.

[18] R.M. Karp. *Reductibility among combinatorial problems*. págs 85-103. Plenum Press, New York, 1972.

[19] Bernhard Korte and Jens Vygen. *Combinatorial Optimization, Theory and Algorithms*. Number 21 in Algorithms and Combinatorics. Springer, págs 561-598, third edition, 2005.

[20] H.W. Kuhn and A.W. Tucker. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, 1951.

[21] A.H. Land and Doig A.G. *An automatic model of solving discrete programming problems*. Econometrica 28, 1960.

[22] Venkata N. Padmanabhan Lili Qiu and Geoffrey M. Volker. On the placement of web servers replicas. In *Proceedings of IEEE INFOCOM'01*, 2001.

[23] Y. Ye M., Mahdian and J. Zhang. Approximation algorithms for metric facility location problems. 2006.

[24] Plus Magazine. Agner Krauro Erlang (1878-1929), 30/04/1997, http://plus.maths.org/content/os/issue2/erlang/index, (25/05/2012).

[25] Garey M.R. and D.S. Johnson. *Computers and Intractability A guide to the theory of NP-Completeness*. 1979.

[26] G. Cornuejols G.L. Nemhauser and L.A. Wolsey. *Discrete Location Theory*. págs 119-171. John Wiley and Sons Inc., 1990.

[27] Bellman R. and S. Dreyfus. *Applied Dynamic Programming.* Princeton University Press, 1962.

[28] Harvey M. Salkin. *Integer Programming.* Addison-Wesley Publishing Company, 1975.

[29] K.J. Arrow Scarf, H.E. and S. Karlin. *Approximation Solutions to a Simple Multi-Echelon Inventory Problem.* Standford University Press, 1962.

[30] Hamdy A. Taha. *Integer Programming, Theory, Applications and Computations.* Academic Press, 1975.

[31] Jens Vygen. *Approximation Algorithms for Facility Location Problems.* Research Institute for Discrete Mathematics, 2005.

[32] Wikipedia. Francois Quesnay, 19/01/2007, http://es.wikipedia.org/wiki/quesnay, (11/02/2013).

[33] Averill M. Law y W.David Keltan. *Simulation modeling and analysis.* McGraw-Hill, Tuscon Arizona, 1991.