Universidad Católica de Santa María

Escuela de Postgrado

Doctorado en Matemática Aplicada

IMPLEMENTACIÓN COMPUTACIONAL DE MÉTODOS PARA LA RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN LINEAL ENTERA

Tesis presentada por la Maestra:
Chirinos Avilés, Yanet Silvana
Para Optar el Grado Académico de:
Doctor en Matemática Aplicada
Asesor:
Dr. Gómez Valdez, Badhin

AREQUIPA – PERÚ

2019

DICTAMEN DE BORRADOR DE TESIS PARA OPTAR EL GRADO ACADÉMICO DE DOCTOR EN MATEMATICA APLICADA

Título : "IMPLEMENTACION COMPUTACIONAL DE METODOS PARA LA RESOLUCION DE PROBLEMAS DE PROGRAMACION LINEAL ENTERA"
Graduando : Chirinos Avilés, Yanet Silvana
Fecha : Arequipa, 2019 enero 08

Visto el Borrador de Tesis presentado y no teniendo observaciones, nuestro dictamen es **PROCEDENTE.**

Atentamente,

EPG Universidad Católica de Santa María

Dr. Héctor Raúl Velarde Bedregal
Docente Principal de la UCSM
Dictaminador

DICTAMEN DE JURADO DICTAMINADOR DE BORRADOR DE TESIS DE DOCTORADO EN MATEMATICA APLICADA

DIRIGIDO A: Dr. José A. Villanueva Salas, Director de la Escuela de Postgrado

DOCTORANDO: YANET SILVANA CHIRINOS AVILES

EXPEDIENTE: 20180000053714

TESIS: "IMPLEMENTACIÓN COMPUTACIONAL DE MÉTODOS PARA LA RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN LÍNEAL ENTERA"

FECHA: 08 de enero del 2019

Dictamen:

Me dirijo a usted en mi calidad de jurado evaluador del borrador de tesis *"Implementación Computacional de Métodos para la Resolución de Problemas de Programación Lineal Entera"*, luego de haber realizado la lectura exhaustiva con las correspondientes observaciones, y el doctorando habiéndolas subsanado, es que doy MI APROBACIÓN al escrito de la presente tesis doctoral.

Por lo expuesto, considero que el trabajo está listo para la defensa de tesis.

Atentamente,

Prof. Badhin Gómez Valdez, Ph.D.
Profesor Investigador
Código 2632

Universidad Católica
de Santa María

Escuela Profesional de
Mecánica, Mecánica Eléctrica
y Mecatrónica

INFORME

DICTAMEN DE JURADO DICTAMINADOR

A:	Dr. José A. Villanueva Salas Director de la Escuela de Postgrado
DE:	Dr. Hermann E. Alcázar Rojas Docente de la E.P de Ingeniería Mecánica, Mecánica –Eléctrica y Mecatrónica
FECHA:	27 de diciembre de 2018
REFERENCIA:	**Expediente N° 20180000053714, Magister CHIRINOS AVILES YANET SILVANA**
ASUNTO:	**Dictamen de Borrador de Tesis para el Grado Académico de Doctor**

Es grato dirigirme a usted para saludarlo y presentarle el dictamen referido al expediente N° 20180000053714 presentado por la **Magister CHIRINOS AVILES YANET SILVANA** estudiante del Doctorado en Matemática Aplicada, del Proyecto de Tesis para obtener el Grado de Doctor titulado "Implementación computacional de métodos para la resolución de problemas de programación lineal entera".

Una vez revisado el proyecto de tesis, el dictamen es el siguiente:

FAVORABLE, para que proceda con la presentación de la Tesis.

Sin otro particular, me despido reiterándole mis saludos.

Atentamente,

Dr. Hermann E. Alcázar Rojas
1634

27 DIC. 2018

A mis Padres:

Alberto Chirinos Janco

Y

Rosa Avilés de Chirinos

Por su apoyo, dedicación, confianza, seguridad y aliento siempre en seguir adelante, porque sin vuestra ayuda no hubiera conseguido nada de lo que hoy soy.

A mi hermana Maritza y familia,

Por sus atenciones, alegrias y comprensión.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a mi asesor de tesis de Doctorado en Matemática Aplicada, Dr. Badhin Gómez Valdez, por la orientación y ayuda a lo largo de todos estos meses. Asi como al Laboratorio Computacional de Bioinformática – CIIM – UCSM, por proporcionarme información y resultados para la realización de este trabajo de investigación.

También agradecer a la Universidad Católica de Santa María y a la Escuela de Posgrado, por darme la oportunidad de realizar mis estudios de Doctorado en Matemática Aplicada, como también concluir el trabajo de investigación de la tesis.

Introducción

El Problema de Programación Lineal tiene sus limitaciones, debido a que las soluciones, las cuales son vectores en $\Re^n$, pueden tener componentes fraccionarios. Es así que una solución óptima en Programación Lineal puede considerar como válida respuestas como por ejemplo "cómprese 18½ autos". Pero en la vida real no podemos aceptar ese tipo de solución, haciéndose necesaria la búsqueda de técnicas que nos muestren resultados cuyas componentes sean enteras.

Por otro lado, el Problema de Programación Lineal Entera no es más fácil de resolverse que el Problema de Programación Lineal clásico, esto porque la región de factibilidad de programación lineal entera no es convexa, lo cual es primordial la construcción de métodos eficientes en optimización.

La experiencia que se tuvo en la Tesis cuyo tema fue: "Programación Lineal Entera Vía El Método de Puntos Interiores Primal Dual" se pudo ver que el problema de Programación Lineal Entera no es nada fácil de resolver, es por eso que en la mayoría de libros de investigación de operaciones apenas muestran la esencia del método, y los algoritmos no son incluidos por tratarse de procedimientos extremadamente complicados. Con esto decimos que la Programación Lineal Entera es uno de los más difíciles de tratar porque no existe método eficiente para su solución, cualquier método tomara un tiempo extremadamente grande desde el punto de vista computacional, y esto empeora a medida que el problema aumente en el número de variables y restricciones.

Además, incluiremos en esta tesis otra versión del método de puntos interiores, el cual usa técnicas de predicción y corrección, lo llamamos el Método de Puntos Interiores Predictor-Corrector, este método también resuelve problemas de programación lineal (de grandes dimensiones). Ahora, nuestro trabajo consiste en crear un nuevo método que resulta atractivo para estos tipos de problemas de programación lineal entera, lo que haremos es fusionar el método de puntos interiores predictor-corrector con el método de ramificación y acotamiento R&A.

En este trabajo, estamos interesados en la manera de cómo resolver el problema de programación lineal entera, más aún, en obtener un método más eficiente que los clásicos.

En el primer capítulo se presentara el problema de programación lineal entera y lo diferenciaremos del problema de programación lineal, también analizaremos los métodos clásicos existentes para su resolución: el método de ramificación y acotamiento y el método del plano cortante.

Luego, introduciremos el método de puntos interiores en sus dos versiones primal-dual y predictor-corrector para programación lineal. Aquí específicamente analizaremos el método de Penalización Interna o también conocido como el método de Barrera para la construcción de un método de puntos interiores acondicionado a la solución del problema de programación lineal entera.

En el segundo capítulo se mostrara que el problema de programación lineal entera es un problema difícil de resolver, para esto se ha introducido parámetros, que nos permitan comparar algoritmos y clasificar los problemas de acuerdo al nivel de esfuerzo que demanda resolverlo.

El tercer capítulo trata del tema principal de esta tesis: "Implementación Computacional de Métodos para la Resolución de Problemas de Programación Lineal Entera", presentaremos los métodos que resultan ser la combinación del método de Puntos Interiores, en sus dos versiones Primal-Dual y Predictor-Corrector para programación lineal con el método de Ramificación y Acotamiento, además mostraremos los algoritmos, los programas y los experimentos computacionales para la resolución de problemas de programación lineal entera, al primer método lo llamaremos "Programación Lineal Entera Vía el Método de Puntos Interiores Primal-Dual" *PLEPIPD* y al segundo método, el cual nos centraremos en nuestro trabajo de investigación lo llamaremos "Programación Lineal Entera Vía el Método de Puntos Interiores Predictor-Corrector" *PLEPIPC*.

Resaltamos que el objetivo principal de nuestro trabajo de investigación consiste en:

Comprobar por medio de la implementación que el método de Puntos Interiores en sus dos versiones Primal- Dual y Predictor-Corrector, cada una de ellas fusionadas con el Método de ramificación y acotamiento, resuelven el problema de programación lineal entera.

También presentaremos en este trabajo los objetivos secundarios, tales como:

- Reconocer las técnicas más usuales para abordar el Problema de Programación Lineal Entera.
- Estudio del Problema de Programación Lineal Entera
- Analizar los métodos clásicos existentes
- Exponer un nuevo método, el cual resultará de una adaptación de la tesis: "Programación lineal entera vía el método de puntos interiores primal-dual"
- Exponer otro método nuevo que resulta de la fusión del método de puntos interiores predictor-corrector con el método de ramificación y acotamiento.

- Dar a conocer la implementación computacional de los métodos de Programación Lineal Entera Vía el Método de Puntos Interiores Primal Dual (*PLEPIPD*) y Programación Lineal Entera Vía el Método de Puntos Interiores Predictor Corrector (*PLEPIPD*).

RESUMEN

La presente investigación trata sobre el estudio y la implementación de dos métodos, Programación lineal Entera Vía El Método de Puntos Interiores Primal-Dual y del método de Programación Lineal Entera Vía El Método de Puntos Interiores Predictor-Corrector, estos métodos resuelven problemas de Programación Lineal Entera, además se basan en el uso de los métodos de Puntos Interiores Primal-Dual y del método de Puntos Interiores Predictor-Corrector, son eficientes para problemas de grandes dimensiones, esto se debe a la complejidad de los algoritmos, ya que son de orden polinomial. Estos métodos son fusionados con el método de Ramificación y Acotamiento. Ambos métodos se resumen en algoritmos, el cual los llamamos PLEPIPD y PLEPIPC, son recursivos y fueron implementados y sometidos a diversos experimentos para confirmar su validez.

Palabras clave: *Programación Lineal Entera (PLE), El Método de Ramificación y Acotamiento, el Método de Puntos Interiores en sus dos veciones Primal-Dual y Predictor-Corrector, Complejidad computacional.*

ABSTRACT

This research deals with the study and implementation of two methods, Integer Linear Programming via Primal-Dual Inner Point Method and the Integer Linear Programming via Predictor-Corrector Inner Point Method. These methods solve Integer Linear Programming problems, they are based on the use of the primal-dual internal points and the predictor-corrector internal point's method they are efficient for large-scale problems, this is due to the complexity of the algorithms, since they are of polynomial order. These methods they are fused with the method Branching and bounding. Both methods are summarized in algorithms, which we call PLEPIPD and PLEPIPC, are recursive and were implemented and subjected to various experiments to confirm their validity.

Keywords: *Integer Linear Programming (PLE), The Branching and bounding Method, The Method of Interior Points in its two versions Primal-Dual and Predictor-Corrector, Computational Complexity.*

Índice

Índice de Gráficas

Capítulo 1

El Problema de Programación Lineal Entera

En la primera sección de este capítulo presentaremos el Problema de Programación Lineal Entera y los métodos más conocidos para su resolución, tales como el Método de Ramificación y Acotamiento y el Método del Plano Cortante.

Comenzaremos definiendo el Problema de Programación Lineal (PL), a quien denominaremos en este texto Problema de Programación Lineal en Variable Continua, corresponde a la siguiente formulación matemática

$$\begin{aligned} &\text{Minimizar } c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{1.1}$$

donde $A \in \Re^{m\times n}$, $b \in \Re^{m}$, $c \in \Re^{n}$. Resolver este problema consiste en encontrar un vector $x \in \Re^{n}$ tal que satisfaga las restricciones $Ax = b$, las condiciones de no negatividad $x \geq 0$, y que haga mínima la función objetivo $c^{\mathrm{T}}x$. Decimos que el problema está en variable continua, porque la solución óptima puede tener componentes fraccionarias.

Ahora con respecto a la DUALIDAD EN PROGRAMACIÓN LINEAL propuesto por (Bazaraa, Jarvis y Sherali, 1990, p.243), resulta ser la relación entre Primal-Dual. Es decir, asociada a cada problema lineal existe otro problema de programación lineal denominado PROBLEMA DUAL, que posee importantes propiedades y relaciones notables con

respecto al problema lineal original, problema que para diferencia del dual se denomina entonces como PROBLEMA PRIMAL.

La dualidad y las condiciones de optimalidad de un problema de programación lineal son usadas para la construcción del método primal-dual.

Al hablar del enlace entre el primal y el dual, es muy importante conocer lo siguiente:

El problema dual tiene tantas variables como restricciones tiene el problema primal.

El problema dual tiene tantas restricciones como variables tiene el problema primal.

Los coeficientes de la función objetivo del problema dual son los términos independientes de las restricciones del problema primal.

Los términos independientes de las restricciones del dual son los coeficientes de la función objetivo del problema primal.

La matriz de coeficientes técnicos del problema dual es la transpuesta de la matriz técnica del problema primal.

El sentido de las desigualdades de las restricciones del problema dual y el signo de las variables del mismo problema, depende la forma de que tenga el signo de las variables del problema primal y del sentido de las restricciones del mismo problema.

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

El problema dual de un problema dual es el problema primal original.

Los problemas duales son los que se obtienen de un problema primal en forma canónica y normalizada, es decir, cuando llevan desigualdades de la forma mayor o igual en los problemas de minimización, y desigualdades menores o iguales para los problemas de maximización.

Aquí también intervienen las condiciones de KARUSH-KUHN-TUKER (KKT) en los problemas lineales (primales y duales). Consideremos el siguiente problema lineal, que denominaremos PRIMAL

$$\begin{aligned} &\text{Minimizar } Z(x) = c^T x \\ &\text{Sujeto a : } Ax \geq b \\ &\qquad x \geq 0 \end{aligned} \tag{1.2}$$

Si agregamos la variable de exceso a cada restricción de (1.2), obtenemos:

$$\begin{aligned} &\text{Minimizar } Z(x) = c^T x \\ &\text{Sujeto a :} \\ &\qquad Ax - y = b \\ &\qquad x \geq 0 \\ &\qquad y \geq 0 \end{aligned}$$

Este problema tiene sólo restricciones de igualdad y variables no negativas. Ahora las restricciones de igualdad se pasan a la función objetivo (Lagrangeano):

$$\begin{aligned} &\text{Minimizar } L(x,\lambda) = c^T x + \lambda^T (Ax - y - b) \\ &\text{Sujeto a :} \\ &\qquad x \geq 0 \\ &\qquad y \geq 0 \end{aligned}$$

Ahora este problema sólo tiene restricciones de no-negatividad.

Aquí, $\lambda = (\lambda_1, \lambda_2, ..., \lambda_m)$ representa el vector de los multiplicadores de Lagrange asociados a las restricciones.Las condiciones de optimalidad (Condiciones de KKT) presentado por Seebach (2006) para este problema con respecto a las variables, son:

$$\frac{\partial L}{\partial x}(\bar{x}, \bar{y}, \bar{\lambda}) \geq 0 \qquad \frac{\partial L}{\partial y}(\bar{x}, \bar{y}, \bar{\lambda}) \geq 0$$

$$x\frac{\partial L}{\partial x}(\bar{x}, \bar{y}, \bar{\lambda}) = 0 \qquad y\frac{\partial L}{\partial y}(\bar{x}, \bar{y}, \bar{\lambda}) = 0$$

$$x \geq 0 \qquad y \geq 0$$

$$\frac{\partial L}{\partial \lambda}(\bar{x}, \bar{y}, \bar{\lambda}) = 0$$

Reordenando

$$\frac{\partial}{\partial x}(cx) + \lambda \frac{\partial}{\partial x}(-Ax) \geq 0$$
$$x\left[\frac{\partial L}{\partial x}(cx) + \lambda \frac{\partial}{\partial x}(-Ax)\right] = 0$$
$$x \geq 0$$
$$\lambda \geq 0$$
$$y\lambda = 0$$
$$y \geq 0$$
$$b - Ax + y = 0$$

Despejando y de estas últimas restricciones y reemplazando en las demás tenemos:

$$\frac{\partial}{\partial x}(cx) + \lambda \frac{\partial}{\partial x}(-Ax) \geq 0$$
$$x\left[\frac{\partial L}{\partial x}(cx) + \lambda \frac{\partial}{\partial x}(-Ax)\right] = 0$$
$$x \geq 0$$
$$\lambda \geq 0$$
$$(Ax - b)\lambda = 0$$
$$y \geq 0$$
$$Ax - b \geq 0$$

En resumen las condiciones (necesarias de primer orden) de KKT se pueden derivar de una función:

$$L(x, \lambda) = c^T x + \lambda^T (b - Ax)$$

$$\frac{\partial L}{\partial x} = \frac{\partial}{\partial x} c^T x + \lambda^T \frac{\partial}{\partial x}(b - Ax) \geq 0$$
$$x\frac{\partial L}{\partial x} = x\left(\frac{\partial}{\partial x} c^T x + \lambda^T \frac{\partial}{\partial x}(b - Ax)\right) = 0$$
$$x \geq 0$$
$$\frac{\partial L}{\partial \lambda} = b - Ax \leq 0$$
$$\lambda \frac{\partial L}{\partial \lambda} = \lambda (b - Ax) = 0$$
$$\lambda \geq 0$$

Asociados a este problema primal tenemos otro problema lineal denominado DUAL (posteriormente explicaremos la relación entre ambos):

$$\begin{aligned} &\text{Maximizar } G(\lambda) = \lambda b \\ &\text{Suejto a :} \\ &\lambda A \le c \\ &\lambda \ge 0 \end{aligned} \tag{1.3}$$

La función Lagrangiana de este problema será:

$$L(\lambda, x) = \lambda b + (c - \lambda A)x$$

en donde el vector $x = (x_1, x_2, ..., x_n)$ representa a los multiplicadores asociados a las restricciones del dual.

Obteniendo las condiciones de KKT respecto de las variables son:

$$\frac{\partial L}{\partial \lambda} = \frac{\partial}{\partial \lambda}\lambda b + x\frac{\partial}{\partial \lambda}(c - \lambda A) \le 0$$
$$\lambda \frac{\partial L}{\partial \lambda} = \lambda\left(\frac{\partial}{\partial \lambda}\lambda b + x\frac{\partial}{\partial \lambda}(c - \lambda A)\right) = 0$$
$$\lambda \ge 0$$
$$\frac{\partial L}{\partial x} = c - \lambda A \ge 0$$
$$x\frac{\partial L}{\partial x} = x(c - \lambda A) = 0$$
$$x \ge 0$$

Como se puede observar, ambas condiciones de optimalidad son las mismas para los problemas. A la misma consideración se puede llegar sin más que comparar la función de Lagrange de los dos problemas y ver que son iguales:

$$L(x, \lambda) = cx + \lambda(b - Ax)$$
$$L(\lambda, x) = \lambda b + (c - \lambda A)x = \lambda b + cx - \lambda Ax = cx + \lambda(b - Ax) = L(x, \lambda)$$

Por lo tanto, asociado a todo problema de programación lineal existe otro problema de programación lineal denominado PROGRAMA DUAL que tiene importantes relaciones con el problema original denominado PROGRAMA PRIMAL.

De lo anterior, podemos decir que el programa dual de un programa dual proporciona el programa primal original.

Por otro lado, las condiciones necesarias y suficientes para que un problema de programación lineal tenga solución, es que ambos problemas primal dual sean factibles. Además, la condición necesaria y suficiente para que exista solución óptima del primal (x^*), es que exista una solución óptima para el dual (λ^*) y que el valor de la función objetivo de ambos programas sea igual $Z(x^*) = G(\lambda^*)$.

Ahora para que (x^*, λ^*) sean soluciones óptimas del programa primal y dual es que satisfagan las condiciones de holgura complementaria:

$$\begin{aligned} (c - \lambda^* A)x^* &= 0 \\ \lambda^*(b - Ax^*) &= 0 \end{aligned} \tag{1.4}$$

A continuación daremos algunos conceptos sobre la programación lineal y la dualidad.

Por otro lado, el Problema de Programación Lineal Entera (PLE) consiste en la siguiente formulación matemática

Minimizar $c^{\mathrm{T}}x$

Sujeto a:

$$\begin{aligned} Ax &= b \\ x &\geq 0 \\ x &\text{ entero} \end{aligned} \tag{1.5}$$

donde $A \in \Re^{m \times n}$, $b \in \Re^m$ y $c \in \Re^n$.

Resolver este problema consiste en encontrar un vector $x \in Z^n$ que satisfaga las restricciones, las condiciones de no negatividad, y haga mínima la función objetivo. La gran diferencia radica en que ahora se requiere que todas las componentes del vector óptimo x sean enteras.

El problema PLE no es más fácil de resolver que el problema PL, por el contrario, en los siguientes capítulos mostraremos que PLE está en la clase donde están los problemas de más difícil solución. Difícil en el sentido computacional, es decir, un problema se considera difícil si los métodos existentes para resolverlo demandan un tiempo extremadamente grande, cuando el tamaño (intuitivamente: el número de variables y restricciones) del problema aumenta.

1.1 El Problema de Programación Lineal Entera (PLE)

En esta sección veremos dos métodos clásicos para resolver PLE : el Método de Ramificación y Acotamiento y el Método del Plano Cortante. Ambos métodos presentan la siguiente característica: comienzan calculando un punto óptimo x resolviendo el problema PLE como si fuera un problema PL, si x no es un vector entero, entonces se aplica un proceso por el cual se añaden restricciones especiales que modifican iterativamente la región de factibilidad, con el propósito de encontrar una solución óptima entera.

La característica de estos dos métodos es generar restricciones especiales, aumentando dramáticamente el tamaño del problema, ese inconveniente tiene un alto costo: el tiempo total de ejecución crece.

Las investigaciones (Taha. H, 2012) mostraron que el Método de Ramificación y Acotamiento es más efectivo que el Método del Plano Cortante. La intención en esta sección no es comparar estos métodos, apenas se analizara, debido a que se combinara uno de ellos con el Método de Puntos Interiores Primal-Dual, como también se combinara con el Método de Puntos Interiores Predictor-Corrector.

El Problema de Programación Lineal Entera definido en (1.5) exige que todas las variables (componentes) del vector óptimo tengan valores enteros. Pero a veces se requiere que sólo algunas lo sean, posteriormente lo comentaremos en uno de los capítulos.

Definición 1.1. El problema PLE sin la condición que x tenga componentes enteras lo denominaremos Relajación de PLE (Yanet y Percy, 2003).

Es decir, la relajación de PLE es un problema de programación lineal en variable continua, nosotros usaremos esta notación frecuentemente en lo sucesivo.

1.1.1 Método de Ramificación y Acotamiento

Un grafo $G = \langle N, A \rangle$ es un conjunto N cuyos elementos se llaman nodos y un conjunto A cuyos elementos se llaman aristas. En otras palabras, un grafo es un conjunto de nodos unidos por un conjunto de aristas (flechas o líneas), existen dos tipos distintos de grafos: grafos dirigidos o di-grafos y grafos no-dirigidos.

En el caso de di-grafos, los nodos son unidos por líneas con orientación (flechas) llamadas aristas. En los grafos no-dirigidos los nodos son unidos apenas por líneas.

El Método de Ramificación y Acotamiento es una técnica para realizar búsquedas en un grafo dirigido dado en forma implícita, este grafo usualmente es acíclico o un árbol. En un árbol los nodos representan las hojas y las aristas las ramas (figura 1.1). La finalidad es buscar la solución óptima de algún problema. En cada nodo calculamos una cota para los posibles valores de las demás soluciones aún no investigadas y las cuales ocupan lugares más distantes. Si la cota demuestra que cualquier solución es peor que la mejor solución que se ha encontrado, entonces no necesitamos seguir explorando más esta parte del grafo. (Taha, 2012, p.336)

En la versión más simple, el cálculo de esa cota se combina con una búsqueda de profundidad uno, y sirve solamente, para la bifurcación del árbol o para cerrar un cierto trayecto de búsqueda dentro de un grafo. El cálculo de esta acotación no sólo sirve para recortar el proceso de búsqueda de la solución óptima, sino que otorga una estrategia de señalar rutas más promisorias.

En términos generales, cuando se aplica la técnica de ramificación y acotamiento para la solución de algún problema de optimización, la implementación computacional no trabaja directamente en el orden de creación del árbol, sino que son usadas técnicas recursivas que trabajan en el sentido inverso a su creación.

Un ejemplo ilustra la técnica de Ramificación y Acotamiento:

Supongamos que S es un conjunto de personas, deseamos determinar la persona de mayor edad tal que ella cumple años hoy día, donde por algún procedimiento es posible apenas entrevistar a la persona de mayor edad.

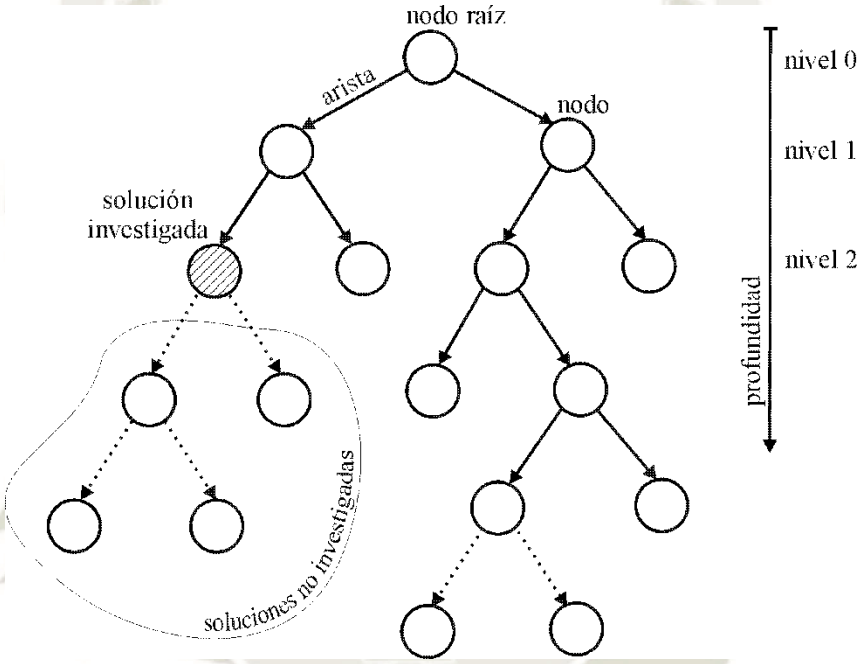


Figura 1.1. Grafo: árbol de búsqueda en un grafo dirigido

Comenzamos determinando la persona de mayor edad e_0 dentro de S, si ésta cumple años hoy día entonces hemos resuelto el problema. Caso contrario, tal persona es retirada de S y formamos dos subconjuntos (*ramificamos*) S_I y S_D, tal que

$$S - \{e_0\} = S_I \cup S_D .$$

Posteriormente determinamos la persona de mayor edad e_I en S_I, si ésta cumple años hoy día, entonces el problema está resuelto con respecto a S_I y decimos que el problema está acotado con respecto a S_I. Pero aún no está resuelto el problema principal, pues falta comparar ese resultado con el que obtendremos en S_D. Si en S_D la persona de mayor

edad e_D es tal que $\text{edad}(e_D) \leq \text{edad}(e_I)$, entonces no tiene sentido seguir buscando en S_D una persona de mayor edad que cumpla años hoy día, pues

$$\text{edad}(e_D) \geq \max\{\text{edad}(e) : e \in S_D,\ e \text{ cumpleaños hoy día}\}$$

En estas circunstancias, decimos que el problema con respecto a S_D también es acotado. Por lo tanto, la solución buscada para el problema principal con respecto a S será e_I.

Note que si $\text{edad}(e_D) > \text{edad}(e_I)$, no podemos concluir nada aún, y tenemos que seguir repitiendo el procedimiento (ramificando) hasta que algún subproblema se considere acotado. Debemos aclarar que algún subproblema también se considera acotado si es infactible, es decir, si no tiene solución.

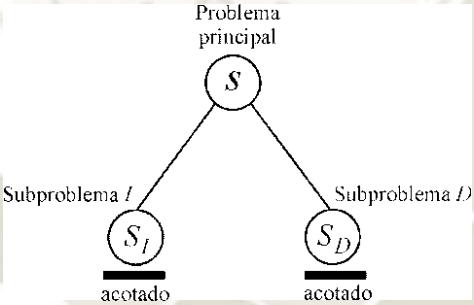


Figura 1.2.

La anterior técnica se denomina *ramificación y acotamiento*, cada conjunto se subdivide en dos subconjuntos, ocasionando que problemas se subdividan en dos subproblemas. La figura 1.2 ilustra la situación del ejemplo que acabamos de considerar y la figura 1.3 es una situación más general de esta técnica.

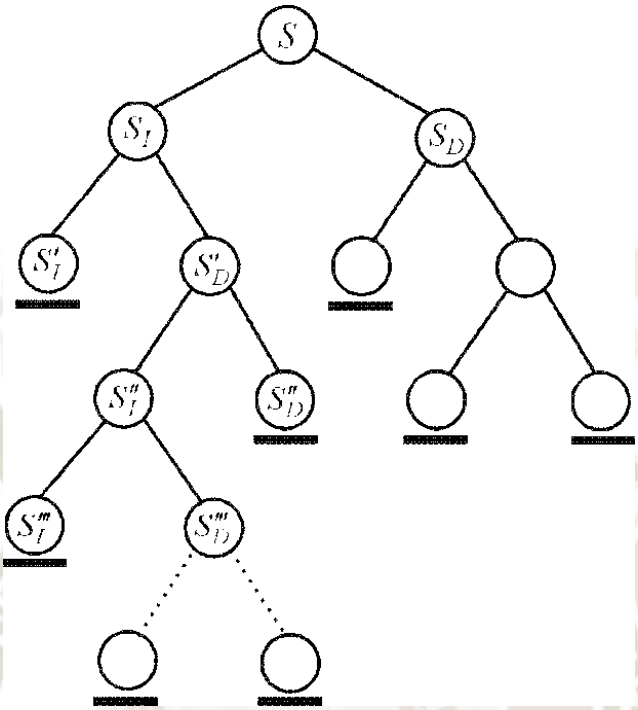


Figura 1.3.

1.1.2 Método del Plano Cortante

El Método del Plano Cortante es uno de varios métodos existentes para resolver el Problema de Programación Lineal Entera (Taha, 2012, p.344). La manera cómo procede este método es crear, en cada iteración, cortes en forma de restricciones adicionales, de tal manera que la región de factibilidad sea modificada y los puntos enteros factibles no sean desechados. En esta subsección se hablará de la manera cómo se generan tales cortes denominados *Cortes de Gomory*.

El inconveniente de este método surge cuando se trabaja con problemas cuyo número de restricciones y variables es grande, pues solamente trabaja con problemas de dimensión pequeña. Si lo comparamos con el Método de Ramificación y Acotamiento, está comprobado que el Método del Plano Cortante resulta ser menos eficiente cuando es sometido a un problema de "grandes" dimensiones[1]. Cuando el Método del Plano Cortante fue publi-

[1] Un número grande de variables y restricciones, por ejemplo, 100 ó 200 variables bastan para notar el mal comportamiento.

cado, se le consideró uno de los precursores dentro de la Programación Lineal Entera, esto dio origen a la creación de nuevos métodos de resolución en este campo.

Considere el siguiente problema PLE

$$\text{Minimizar } c^{\mathrm{T}}x$$

Sujeto a:

$$Ax = b$$

$$x \geq 0$$

$$x \text{ entero}$$

Una manera de resolver este problema es ignorar la última condición, x entero, y resolver el problema como si fuese un problema de programación lineal en variable continua (PL). En condiciones de optimalidad, si todas las variables tienen valores enteros, entonces se tiene la solución óptima del problema de PLE original, de otro modo consideramos la adición de una restricción o *corte* a la región de factibilidad del problema PL .

El procedimiento de adición de restricciones es repetido hasta hallar una solución entera óptima, la infactibilidad del problema, o hasta detectar que el problema posee soluciones ilimitadas.

El método del plano cortante procede del siguiente modo: Primeramente resuelve la relajación de PLE y encuentra una solución óptima x^*, si el vector x^* contiene todas sus componentes enteras, entonces el problema está resuelto. Caso contrario, es añadida una nueva restricción (corte) modificando de este modo la región de factibilidad. Para resolver la relajación el método hace una llamada a una subrutina, la cual es justamente el Algoritmo Simplex, en realidad es una variación de este método denominado Dual-Simplex. Debemos resaltar que el corte sólo es posible cuando es conocida una solución básica.

En general podemos necesitar cualquier número finito de cortes para llegar al punto óptimo entero. De hecho, el número de cortes necesarios para producir la solución entera parece ser independiente de las dimensiones del problema.

Seguidamente veremos cómo se construye el *corte*. Consideremos el problema PLE

Minimizar $c^{\mathrm{T}}x$

Sujeto a:

$$\begin{aligned} Ax &= b \\ x &\geq 0, \quad \text{entero} \end{aligned} \tag{1.6}$$

Supongamos que A, b y c tengan componentes enteras, y asumamos que $\text{rango}(A) = m$. Si estos datos no cumplen esta característica, entonces es posible transformarlos fácilmente mediante la introducción de constantes multiplicativas, este procedimiento se denomina *escalonamiento del problema*.

Supongamos que resolvemos la relajación de PLE mediante el Algoritmo Simplex, y obtenemos la solución básica factible óptima $x*$, entonces existe una submatriz básica B de A asociada a $x*$, note B es inversible desde que A tiene rango completo. Si las columnas de A y las componentes de x son reordenadas de modo que la siguiente partición en bloques es permitida: $A = [B \quad N]$ y $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$, reemplazando en $Ax = b$ de (1.6) tenemos

$$\begin{aligned} x_B &= B^{-1}b - B^{-1}N\,x_N \\ x_B &= \bar{b} - \sum_{j \in R} B^{-1}a_j x_j \\ x_B &= \bar{b} - \sum_{j \in R} y_j x_j \end{aligned} \tag{1.7}$$

donde R es el conjunto de los índices de las variables no básicas, $y_j = B^{-1}a_j$, a_j es la j -columna de la matriz no básica N, y $\bar{b} = B^{-1}b$. Recuerde que haciendo $x_N = 0$ el vector $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$ es definido como una solución básica, un extremo de la región de factibilidad.

Sea $\bar{b}_r$ la r-componente no entera del vector $\bar{b}$, asumamos por comodidad que las variables no básicas están indexadas ordenadamente $j = m+1,..., n$. Desde (1.7) tenemos la ecuación básica asociada con $\bar{b}_r$:

$$x_r + \sum_{j=m+1}^{n} y_{rj} x_j = \bar{b}_r \tag{1.8}$$

Notemos que x_r es la r-componente de x_B, mientras que y_{rj} es la r-componente de la j-columna de la matriz $B^{-1}N$. Sea i_{rj} el mayor entero menor o igual que y_{rj} (i_{rj} es llamado la parte entera de y_{rj}). Similarmente, sea i_r la parte entera de $\bar{b}_r$. Definimos f_{rj} y f_r como las respectivas partes fraccionarias:

$$f_{rj} = y_{rj} - i_{rj} \qquad \text{y} \qquad f_r = \bar{b}_r - i_r$$

Entonces, $0 \le f_{rj} < 1$ y $0 < f_r < 1$. Usando esta condición, podemos rescribir la ecuación básica para x_r como

$$x_r + \sum_{j=m+1}^{n} (i_{rj} + f_{rj}) x_j = i_r + f_r$$

Reordenando términos, tenemos

$$x_r + \sum_{j=m+1}^{n} i_{rj} x_j - i_r = f_r - \sum_{j=m+1}^{n} f_{rj} x_j$$

Ahora, el lado izquierdo de esta ecuación será entero para cualquier solución factible entera, es así que al reemplazar una solución factible cuyas componentes sean enteras en el lado izquierdo de la ecuación anterior obtendremos un valor entero. Además, el lado derecho debe ser estrictamente menor que uno, desde que $f_r < 1$, $f_{rj} \ge 0$, y $x_j \ge 0$. Pero por la igualdad, el lado derecho también debe ser entero, como no existe un entero mayor

que cero y menor que uno, podemos concluir que esta ecuación debe ser menor o igual que cero. Así podemos escribir

$$f_r - \sum_{j=m+1}^{n} f_{rj} x_j \leq 0 \tag{1.9}$$

La inecuación en (1.9) se denomina *Corte de Gomory.* Este corte, en forma de restricción, es entonces adicionada a las restricciones del problema original, cortando de este modo la región de factibilidad. Algunos resultados interesantes son formalizados en el teorema 1.1, próximamente.

Podemos ver la desigualdad (1.9) de otro modo: sabemos que x en (1.6) está restringido a ser no negativo, así

$$\sum_{j=m+1}^{n} i_{rj} x_j \leq \sum_{j=m+1}^{n} y_{rj} x_j \tag{1.10}$$

Luego, adicionamos x_r en ambos miembros

$$x_r + \sum_{j=m+1}^{n} i_{rj} x_j \leq x_r + \sum_{j=m+1}^{n} y_{rj} x_j$$

por (1.8) tenemos la siguiente ecuación

$$x_r + \sum_{j=m+1}^{n} i_{rj} x_j \leq \bar{b}_r \tag{1.11}$$

En programación lineal entera, la variable x es restringida a tener componentes enteras, y así el lado izquierdo de la ecuación (1.11) es entero. El lado derecho puede ser reemplazado por el valor entero sin alterar la relación produciendo lo siguiente

$$x_r + \sum_{j=m+1}^{n} i_{rj} x_j \leq i_r \tag{1.12}$$

Sustrayendo (1.8) a (1.12) obtenemos lo siguiente

$$\sum_{j=m+1}^{n}(y_{rj}-i_{rj})x_j \geq \bar{b}_r - i_r \qquad (1.13)$$

Como $f_{rj} = y_{rj} - i_{rj}$, $f_r = \bar{b}_r - i_r$, $0 \leq f_{rj} < 1$ y $0 < f_r < 1$, reemplazando f_{rj} y f_r en (1.13), tenemos

$\sum_{j=m+1}^{n} f_{rj}x_j \geq f_r$, por lo tanto $f_r - \sum_{j=m+1}^{n} f_{rj}x_j \leq 0$.

En condiciones de optimalidad, es decir cuando $x_j = 0$ para $j \in R$ y $f_r > 0$, la solución actual x^* del problema de programación lineal (solución cuyas componentes no son todas enteras) no satisface esta restricción adicional. Esto se debe a que al reemplazar x_j en (1.9) tenemos $f_r \leq 0$, lo cual es una contradicción a $f_r > 0$. En otras palabras, la restricción cortaría la región incluyendo la actual solución óptima x^* no entera.

Una vez adicionada la nueva restricción (corte) dada en (1.9) al problema PLE dado en (1.6), se puede aplicar nuevamente el Método Simplex para resolver la relajación respectiva, y así por delante. Por lo general, se usa el tableado Simplex, en estas condiciones, es preferible aplicar una variación denominada Método Dual-Simplex. Este procedimiento en conjunto se conoce como el *Método del Plano Cortante.* El corte dado en (1.9) es llamado también *Corte Fraccional de Gomory*.

Teorema 1.1. Dado el Corte de Gomory de (1.9)

$$f_r - \sum_{j=m+1}^{n} f_{rj}x_j \leq 0$$

a) El punto no entero $x^* = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} x_B \\ 0 \end{bmatrix}$, solución óptima del problema de programación lineal en variable continua, no cumple (1.9).

Es decir, x^* no es un punto factible para el nuevo problema de relajación del problema PLE.

b) Ningún punto factible entero de la nueva región fue eliminado con el corte (1.9).

Demostración.

Para demostrar la parte (a) lo que hacemos es reemplazar la solución básica factible óptima $x*$ en (1.9), como $x_j = 0$ para $j = m+1,..., n$, entonces $f_r \leq 0$, lo cual es una contradicción a $f_r > 0$. Por lo tanto $x*$ no es un punto factible para el nuevo problema de programación lineal entera.

Ahora para mostrar la parte (b), empezamos con el problema original PLE dado en (1.6)

$$\begin{aligned} &\text{Minimizar } c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &Ax = b \\ &x \geq 0, \\ &x \text{ entero} \end{aligned} \tag{1.14}$$

Dada una base B asociada a una solución básica (no necesariamente óptima) entonces el problema PLE (1.14) es equivalente al siguiente problema

$$\begin{aligned} &\text{Minimizar } c_B^{\mathrm{T}}x_B + c_N^{\mathrm{T}}x_N \\ &\text{Sujeto a:} \\ &Bx_B + Nx_N = b \\ &x_B, x_N \geq 0 \\ &x_B, x_N \text{ enteros} \end{aligned} \tag{1.15}$$

Analicemos ahora que pasa cuando x es una solución factible entera, es decir, las componentes de x_B y de x_N son enteras, considerando las restricciones

$$Bx_B + Nx_N = b$$

del problema dado en (1.15), despejando x_B tenemos

$$x_B = B^{-1}b - B^{-1}N\,x_N = \bar{b} - \sum_{m+1}^{n} y_j x_j \tag{1.16}$$

Ahora, sin pérdida de generalidad, en el sistema de ecuaciones dado en (1.16) consideremos la t-ecuación

$$x_t = \bar{b}_t - \sum_{m+1}^{n} y_{tj} x_j$$

Luego hacemos

$$x_t + \sum_{m+1}^{n} y_{tj} x_j = \bar{b}_t \tag{1.17}$$

Usando las definiciones de f_{rj} y f_r vistas anteriormente, en este caso para $t = r$, tenemos

$$f_{tj} = y_{tj} - i_{tj} \tag{1.18}$$

$$f_t = \bar{b}_t - i_t \tag{1.19}$$

donde $0 \le f_{tj} < 1$ y $0 < f_t < 1$. Ahora de (1.18) y (1.19) despejamos y_{tj} y $\bar{b}_t$

$$y_{tj} = f_{tj} + i_{tj} \tag{1.20}$$

$$\bar{b}_t = f_t + i_t \tag{1.21}$$

Ahora reemplazamos (1.20) y (1.21) en (1.17) y obtenemos

$$x_t + \sum_{j=m+1}^{n} (f_{tj} + i_{tj})\, x_j = f_t + i_t$$

$$x_t + \sum_{j=m+1}^{n} f_{tj}\, x_j + \sum_{j=m+1}^{n} i_{tj}\, x_j = f_t + i_t$$

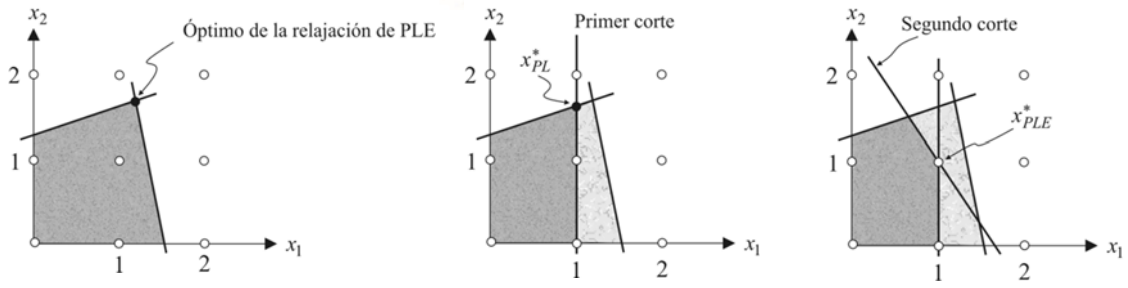
$$x_t + \sum_{j=m+1}^{n} i_{tj}\, x_j - i_t = f_t - \sum_{j=m+1}^{n} f_{tj} x_j \tag{1.22}$$

Por el mismo razonamiento anterior, debido a que x_t es la componente de x_B (entera) y x_j la componente de x_N (enteros para $j = m+1,\ldots, n$), entonces (1.22) es menor o igual que cero. Es decir, como $f_{tj} \geq 0$ y $f_r < 1$, el lado derecho de la ecuación en (1.22) es menor que 1, pero sabemos que i_{tj} es la parte entera de y_{tj} y i_t la parte entera de $\bar{b}_t$, y además x_t, x_j son enteros, luego el lado izquierdo de (1.22) es entero. De este razonamiento, como no existe un número entero entre cero y uno, se tiene

$$f_t - \sum_{j=m+1}^{n} f_{tj} x_j \leq 0 \tag{1.23}$$

Esto significa que una solución factible entera arbitraria de (1.14), o sino de (1.15), continúa siendo factible para el nuevo problema PLE, es decir, no es eliminada por el corte.

Las siguienres figuras 1.4 (a), (b) y (c), muestran cómo opera el Algoritmo del Plano Cortante en el problema de relajación de PLE.



(a) (b) (c)

Figura 1.4. Idea gráfica de como actua el Algoritmo del Plano Cortante en un problema de relajación de PLE

- En la figura 1.4 (a) se tiene el óptimo de la relajación del problema PLE.

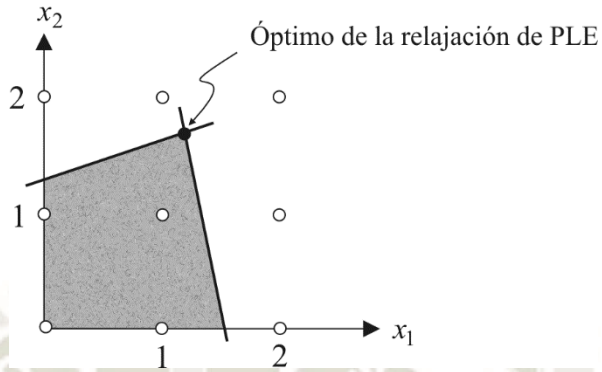


Figura 1.4. (a)

- En la figura 1.4 (b) se tiene la nueva solución óptima x^*_{PL} de la relajación de PLE después de un corte.

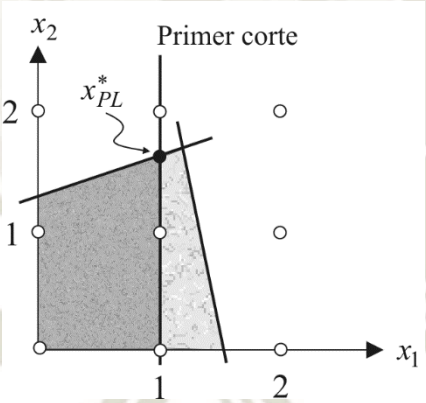


Figura 1.4. (b)

- La figura 1.4 (c) muestra la solución óptima x^*_{PLE} de la relajación de PLE después de dos cortes. Note que x^*_{PLE} constituye ya la solución óptima entera del problema PLE original.

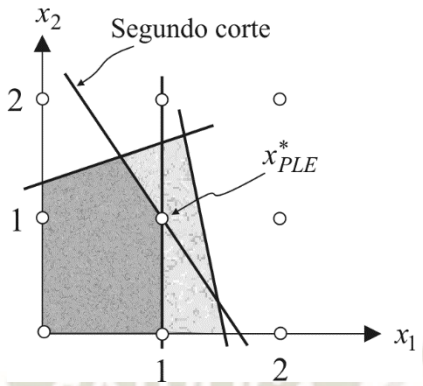


Figura 1.4. (c)

1.2 El Método de Puntos Interiores para Programación Lineal

En 1947 George Dantzig publicó el Método Simplex para programación lineal, fue el primer método formalmente establecido. Este método resultó muy útil en la práctica, pero desde el punto de vista teórico no es eficiente. Tal afirmación se debe a la existencia de una clase especial de problemas para los cuales el método puede tomar un tiempo excesivo en resolverlos, pues el número de iteraciones crece en forma exponencial respecto al número de variables en tales problemas.

En 1984, Karmarkar propuso un algoritmo cuya complejidad computacional es polinomial y que resulto altamente competitivo frente al método Simplex, para resolver problemas con gran tamaño.

El algoritmo de Karmarkar originó una multitud de trabajos alrededor de su idea original que ha sido mejorada en muchos aspectos. Una de las más fructíferas variantes es el algoritmo de Barrera Primal-Dual introducido por Meggido (1986). El método de Barrera es también conocido como el método de Penalización Interna trabaja siempre con puntos interiores del conjunto factible. Existe un parámetro no negativo que acompaña al término de penalización determinando el grado de penalidad a ser impuesto a medida que este parámetro crece (o equivalentemente tiende a cero), la aproximación entre las soluciones de los problemas originales y los penalizados mejoran. En este sentido, los problemas originales pueden ser resueltos a través de una sucesión de problemas penalizados, monitorizándose el parámetro de penalización.

En esta sección trataremos brevemente el Método de Puntos Interiores Primal-Dual para Programación Lineal, aquí incluiremos el método de Penalización Interna (o Barrera), donde al ser aplicado a problemas de programación lineal vía puntos interiores tuvo mucho realce e importancia comparada a otros métodos clásicos, como el método Simplex.

Posteriormente, en el capítulo 3 mostraremos la fusión del método de Puntos Interiores Primal-Dual con el método de Ramificación y Acotamiento, como también la fusión del método de Puntos Interiores Predictor-Corrector con el método de Ramificación y Acotamiento, el cual constituyen herramientas fundamentales para construir nuevos métodos para el Problema de *PLE* .

1.2.1 Penalización Interna

Los métodos de penalización interna o de barrera fueron originalmente propuestos para trabajar con restricciones no lineales de desigualdad, pues en ese caso el conjunto factible tiene interior no vacío. Esto surgió en los años 50, hasta 1984 los métodos de penalización interna conservaron los principios originales de su creación, siendo considerados de poco prestigio. A partir de 1984, cuando aplicados a restricciones lineales, los métodos de barrera tuvieron su prestigio renovado, principalmente en programación lineal vía puntos interiores (algoritmos polinomiales: Karmarkar (1984))

1.2.2 Construcción del Método de Penalización Interna

El método de Penalización Interna (o Barrera) es usado para la construcción del método de Puntos Interiores Primal-Dual, el cual resuelve problemas que pueden ser considerados difíciles de resolver, generalmente con restricciones de igualdad y desigualdad, en problemas relativamente fáciles de resolver, el cual existen técnicas para su resolución como el método de Penalización Interna (o Barrera) que consiste en un funcional que tiende hacia el infinito, cuando la variable tiende hacia la frontera de la región de factibilidad.(Bazaraa, Sherali y Shetty,1993)

Para describirnos la idea fundamental de penalización interna en problemas de programación no lineales con restricciones de igualdad y desigualdad vamos a considerar el siguiente problema de optimización.

$$\begin{aligned} &\text{Minimizar} \quad f(x) \\ &\text{Sujeto a:} \quad g(x) \le 0 \\ &\qquad\qquad x \in X \end{aligned} \tag{1.24}$$

donde $X \subset \Re^n$, $g : \Re^n \to \Re^m$, $f, g \in C^o(\Re^n)$, $\Omega = \{x \in X : g(x) \le 0\}$ tiene interior no vacío, es decir, el interior de Ω, denotado por Ω^o, es definido por $\Omega^o = \{x \in X : g(x) < 0\}$. Ahora, el método consiste en transformar (1.24) en un problema irrestricto con función objetivo

$$f(x) + \mu B(x) \tag{1.25 a}$$

donde $\mu > 0$ es un escalar denominado *parámetro de penalización.*

Otra forma de ver el problema de optimización es

$$\begin{aligned} &\text{Minimice} \quad \theta(\mu) \\ &\text{Sujeto a:} \quad \mu \ge 0 \end{aligned} \tag{1.25 b}$$

donde $\theta(\mu) = \inf\{f(x) + \mu B(x) : g(x) < 0, x \in X\}$. La función barrera B en ambos casos (1.25 a) y (1.25 b) es no negativa y continua sobre la región Ω^o.

Si $\{x^{(k)}\} \subset \Omega, g(x^{(k)}) < 0$ para todo $k \in N$, y $\lim_{k\to\infty} g_i(x^{(k)}) = 0$ para algún $i \in \{1,...,m\}$, entonces $\lim_{k\to\infty} B(x^{(k)}) = \infty$.

Bazaraa, Sherali y Shetty (1993), presenta la función barrera de la siguiente forma

Función Barrera Inversa

$$B(x) = \sum_{i=1}^{m} \frac{-1}{g_i(x)} \tag{1.26}$$

Función Barrera Logaritmo

$$B(x) = -\sum_{i=1}^{m} \log(g_i(x)) \tag{1.27}$$

Función Barrera Logarítmica de Frisch´s

$$B(x) = -\sum_{i=1}^{m} \ln[-g_i(x)] \tag{1.28}$$

Puede ser que la solución óptima del problema (1.24) se encuentre en la frontera de la región de factibilidad, pero el método de penalización interna impide aproximarse a la frontera de la región de factibilidad, esto por el parámetro de penalización, pues valores pequeños de μ equilibran el efecto del valor inmenso de $B(x)$ en las proximidades de la frontera.

Ahora, mostraremos a través de un lema y un teorema cuando $\mu \to 0$, que resolver el problema penalizado (1.25) es equivalente a resolver el problema original (1.24).

Algoritmo (Penalización Interna)

Dados $\mu_1 > 0$, $x^{(0)} \in \Omega^o, k = 1$

Paso 1: Calcular $x^{(k)}$, la solución global del subproblema:

$$\text{Minimizar } f(x) + \mu_k B(x)$$
$$x \in \Omega^o \tag{1.29}$$

Paso 2: Elegir μ_{k+1} tal que $0 < \mu_{k+1} < \mu_k$

Paso 3: Hacer $k = k + 1$ y volver al paso 1.

En el algoritmo el parámetro de penalización μ_k debe aproximarse iterativamente hacia cero, un modo clásico consiste en hacer $\mu_1 = 1$ y $\mu_{k+1} = \mu_k / 10$ para $k = 1, 2, ...$

Ahora, notemos que en el algoritmo se exige que el parámetro de penalización μ tienda a cero, la razón es que bajo esta hipótesis es posible mostrar que el algoritmo posee la propiedad de otorgar un minimizador global para el problema (1.24). (Chirinos y Ticona, 2003). El siguiente Lema 1.1 ratifica lo mencionado.

Lema 1.1. Sea $\left\{x^{(k)}\right\}$ la sucesión generada por el algoritmo (*Penalización Interna*), entonces

$(i)\ f(x^{(k+1)}) + \mu_{k+1} B(x^{(k+1)}) \leq f(x^{(k)}) + \mu_k B(x^{(k)})$
$(ii)\ B(x^{(k)}) \leq B(x^{(k+1)})$
$(iii)\ f(x^{(k+1)}) \leq f(x^{(k)})$

Prueba. Observemos que los parámetros penalizados $\{\mu_k\}$ son estrictamente decrecientes, como la función barrera es no negativa y debido a que $\left\{x^{(k)}\right\}$ es una sucesión de minimizadores globales del problema (1.29), se tiene

$$\begin{aligned} f(x^{(k+1)}) + \mu_{k+1} B(x^{(k+1)}) &\leq f(x^{(k)}) + \mu_{k+1} B(x^{(k)}) \\ &\leq f(x^{(k)}) + \mu_k B(x^{(k)}) \end{aligned} \tag{1.30}$$

Para mostrar (*ii*) vemos además que

$$f(x^{(k)}) + \mu_k B(x^{(k)}) \leq f(x^{(k+1)}) + \mu_k B(x^{(k+1)}) \tag{1.31}$$

La afirmación en (1.30) se debe a que $x^{(k)}$ es un minimizador global de (1.29) con respecto al parámetro μ_k. Ahora restamos (1.31) de (1.30) tenemos

$$(\mu_{k+1} - \mu_k) B(x^{(k+1)}) \leq (\mu_{k+1} - \mu_k) B(x^{(k)})$$

Como $\mu_{k+1} - \mu_k < 0$, se tiene que $B(x^{(k)}) \leq B(x^{(k+1)})$. Por otro lado, para mostrar (*iii*) usamos el hecho que $x^{(k+1)}$ es minimizador del problema (1.29) con respecto al problema de parámetro μ_{k+1}, además del resultado obtenido en (*ii*):

$$\begin{aligned} f(x^{(k+1)}) + \mu_{k+1} B(x^{(k+1)}) &\leq f(x^{(k)}) + \mu_{k+1} B(x^{(k)}) \\ &\leq f(x^{(k)}) + \mu_{k+1} B(x^{(k+1)}) \end{aligned}$$

luego, $f\left(x^{(k+1)}\right) \leq f\left(x^{(k)}\right)$.

La importancia del algoritmo, es que nos proporciona un minimizador global del problema original (1.24), además un factor importante para tal resultado es que el sub-problema (1.29), se está asumiendo que cada $x^{(k)}$ es un minimizador global de dicho subproblema, lo cual no siempre es posible asegurar en tiempo finito.

Teorema 1.2. Sea $f : E_n \to E_1$ y $g : E_n \to E_m$ funciones continuas, y sea X un conjunto cerrado no vacío en E_n. Supóngase que el conjunto $\{x \in X : g(x) < 0\}$ es no vacío. Además, supóngase que el problema primal minimiza $f(x)$ sujeto a $g(x) \leq 0,\ x \in X$ tiene una solución óptima $\bar{x}$ con la siguiente propiedad. Dado alguna vecindad N alrededor de $\bar{x}$, existe un $x \in X \cap N$ tal que $g(x) < 0$. Entonces

$$mín\{ f(x) : g(x) \leq 0, x \in X \} = \lim_{\mu \to 0^+} \theta(\mu) = \inf_{\mu \geq 0} \theta(\mu)$$

Permitiendo $\theta(\mu) = f(x_\mu) + \mu B(x_\mu)$, donde $x_\mu \in X$ y $g(x_\mu) < 0$,[2] entonces el límite de alguna subsucesión convergente de $\{x_\mu\}$ es una solución óptima del problema primal, y $\mu B(x_\mu) \to 0$ cuando $\mu \to 0^+$.

Prueba. Bazaraa, Sherali y Shetty (1993, p.389).

La representación gráfica del teorema 1.2 se muestra en la figura 1.5.

[2] Asumamos que tal punto x_μ existe y son dados por el Lema 9.4.2. (Bazaraa, Sherali y Shetty, 1993, p.387)

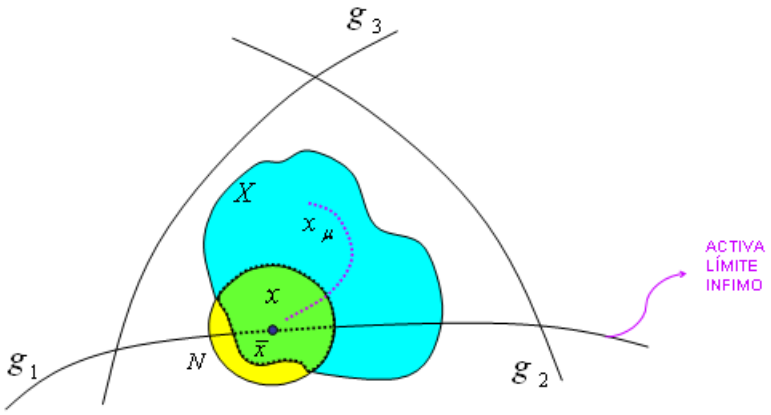


Figura 1.5. Representación Gráfica del Método de Puntos Interiores (Teorema 1.2)

Ahora, en lo que sigue vamos analizar el problema de minimización con restricciones lineales, desde el punto de vista de la penalización interna. Sea

$$\begin{aligned} &\text{Minimizar} && f(x) \\ &\text{Sujeto a :} && Ax = 0 \\ & && x \geq 0 \end{aligned} \tag{1.32}$$

donde $A \in \Re^{m \times n}$, $m \leq n$ y donde $rango\,(A) = m$.

Utilizando la función barrera logarítmica, tenemos el siguiente subproblema, apenas con restricciones lineales de igualdad:

$$\begin{aligned} &\text{Minimizar} && f(x) - \mu \sum_{i=1}^{n} \log(x_i) \\ & && A\,x = b \end{aligned} \tag{1.33}$$

Donde el minimizador de (1.33) necesariamente debe cumplir (1.34), pero el retorno de (1.33) a (1.32) no necesariamente se cumple.

Las condiciones de optimalidad de (1.33) corresponden a un sistema no lineal con $n+m$ restricciones

$$\nabla f(x) - \mu \begin{bmatrix} \frac{1}{x_1} \\ \vdots \\ \frac{1}{x_n} \end{bmatrix} + A^T y = 0$$

$$A x - b = 0 \tag{1.34}$$

La matriz Jacobiana del sistema (1.34) es dada por

$$\begin{bmatrix} \nabla^2 f(x) + \mu X^{-2} & A^T \\ A & 0 \end{bmatrix} \tag{1.35}$$

donde $X = \text{diag}(x_1, \ldots, x_n)$. El número de condición de esta matriz crece cuando $\mu \to 0$ y alguna componente x_i se aproxima a cero.

El mal condicionamiento del método de barrera puede ser controlado escalonando el sistema haciendo

$$z_i = \frac{\mu}{x_i}, \quad i = 1, \ldots, n$$

Entonces (1.34) puede ser reescrito como

$$\begin{aligned} \nabla f(x) - z + A^T y &= 0 \\ Ax &= b \\ x_i z_i - \mu &= 0, \quad i = 1, \ldots, n \end{aligned} \tag{1.36}$$

El sistema aumentado (1.36), con $2n + m$ ecuaciones y $2n + m$ incógnitas, tiene el siguiente Jacobiano

$$J = \begin{bmatrix} \nabla^2 f(x) & A^T & -I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \tag{1.37}$$

donde $Z = \text{diag}(z_1, \ldots, z_n)$.

Note que en este caso (1.37) es independiente de μ. Si tuviéramos complementariedad estricta, esto es: $x_i z_i = 0$ con $x_i \neq 0$ o $z_i \neq 0$, entonces (1.37) tiene rango completo, es decir $J.J^T$ es no singular. Ahora, el sistema (1.36) sólo será mal condicionado si el problema original (1.32) lo fuera. Así, si en vez de trabajar con (1.33), resolvemos (1.36) cuando $\mu = 0$, tenemos las condiciones de Karush Kuhn Tucker (KKT) del problema original (1.32).

Para obtener la minimización del problema (1.32) una vez calculada la matriz (1.37), se utilizara el método de Newton para cada valor de μ fijado, de donde una sucesión de subproblemas será obtenida mediante lo siguiente:

$$x_{k+1} = x_k - J^{-1}(x_k)F(x_k) \tag{1.38}$$

Note que requerimos un punto inicial x_0, $F(x) = 0$ representa el sistema de ecuaciones dado en (1.36) y $J(x)$ su matriz Jacobiana en el punto *x*. Para el caso del problema (1.32) tenemos

$$F(x, y, z) = \begin{bmatrix} \nabla f(x) - z + A^T y \\ Ax - b \\ z_i x_i - \mu \\ \vdots \\ z_n x_n - \mu \end{bmatrix} = 0$$

Veamos una aplicación orientada a la programación lineal.

Consideremos el siguiente problema

$$\text{Minimice} \quad f(x) = c^T x$$

Sujeto a:

$$A\,x \le b$$

$$x \ge 0$$

donde $A = [1\ \ 1]$, $b = [10]$, $c = [-1\ -1]^T$, $x_0 = [1\ \ 1]^T$. Implementaremos el algoritmo de penalización interna (pag. 24) en Matlab, el programa está conformado por las siguientes funciones auxiliares:

La función gradiente $\nabla f(x) = \begin{bmatrix} -1 + \frac{\mu}{10 - x_1 - x_2} - \frac{\mu}{x_1} \\ -1 + \frac{\mu}{10 - x_1 - x_2} - \frac{\mu}{x_2} \end{bmatrix}$

```
function fx=gradiente(x,c,mu)
fx=[c(1)+mu/(10-x(1)-x(2))-mu/x(1)
   c(2)+mu/(10-x(1)-x(2))-mu/x(2)];
```

La función Jacobiano $\nabla f(x) = \begin{bmatrix} \frac{\mu}{(10 - x_1 - x_2)^2} + \frac{\mu}{x_1^2} & \frac{\mu}{(10 - x_1 - x_2)^2} \\ \frac{\mu}{(10 - x_1 - x_2)^2} & \frac{\mu}{(10 - x_1 - x_2)^2} + \frac{\mu}{x_2^2} \end{bmatrix}$

```
function fx=jacobiano(x,mu)
fx=[mu/(10-x(1)-x(2))^2+mu/x(1)^2     mu/(10-x(1)-x(2))^2
     mu/(10-x(1)-x(2))^2              mu/(10-x(1)-x(2))^2+mu/x(2)^2 ];
```

La function Newton

```
function fx=newton(x,c,mu,epsilon,A,b)
i=0;
while norm(gradiente(x,c,mu))>epsilon
   x=x-inv(jacobiano(x,mu))*gradiente(x,c,mu); i=i+1;
   if  i>1000
      error('newton: Paso de límite de iteraciones permitido');
   end
end
```

```
fx=x;
```

El programa penalización interna

```
% PROGRAMA PENALIZACION INTERNA
mu=100;
epsilon=0.001;
i=0;
while mu>0.001
   x=newton(x,c,mu,epsilon,A,b);
   mu=mu/2; i=i+1;
end
disp('x*=');disp(x);
disp('cx*=');disp(c'*x);
disp('iter='); disp(i);
```

Ejecutamos el programa y obtuvimos una buena aproximación a la solución óptima:

$$x^* = [\,4.9992 \quad 4.9992]^T$$
$$c^T x^* = -9.9985$$
$$iter = 17$$

El Método de Penalización Interna (o Barrera) es usado para la construcción del Método de Puntos Interiores, el cual será acondicionado a la solución del problema de programación lineal entera, esto lo veremos en lo que sigue, iniciando con el método de puntos interiores Primal-Dual, posteriormente con el método de puntos interiores Predictor-Corrector.

1.2.3 El Método de Puntos Interiores Primal-Dual

El procedimiento que tiene el método de Puntos Interiores Primal-Dual y el método de Puntos Interiores Predictor-Corrector es que ellos trabajan sobre un problema de programación lineal en la forma estándar, la estrategia de tales métodos consiste en convertir este problema en un problema sin restricciones, el cual no necesariamente es lineal, esto se consigue aplicando el Método de Lagrange a las restricciones de igualdad y el Método

de Fiacco & McCormick (1968) a las condiciones de no negatividad. Finalmente, los procedimientos anteriores llevan a un sistema de ecuaciones no lineal, el cual es resuelto mediante el Método de Newton.

Considere el siguiente problema de programación lineal en la forma estándar

$$\begin{aligned} &\text{Minimizar } c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &\qquad Ax = b \\ &\qquad x \geq 0 \end{aligned} \tag{1.39}$$

donde $A \in R^{mxn}, b \in R^m, c \in R^n$ y $\text{rango}(A) = m$. Considere además su problema dual asociado

$$\begin{aligned} &\text{Maximizar } b^{\mathrm{T}}y \\ &\text{Sujeto a:} \\ &\qquad A^{\mathrm{T}}y + z = c \\ &\qquad z \geq 0 \end{aligned} \tag{1.40}$$

Mediante las técnicas anteriormente mencionadas, estos problemas con restricciones, son transformados en los siguientes problemas irrestrictos:

$$\text{Minimizar } L_P(x,y) = c^{\mathrm{T}}x - \mu\sum_{j=1}^{n}\log(x_j) - y^{\mathrm{T}}(Ax-b) \tag{1.41}$$

$$\text{Maximizar } L_D(x,y,z) = b^{\mathrm{T}}y + \mu\sum_{j=1}^{n}\log(z_j) - x^{\mathrm{T}}(A^{\mathrm{T}}y+z-c) \tag{1.42}$$

donde $\mu > 0$ es un parámetro real de penalización. Ahora, para minimizar la función L_P y maximizar la función L_D se calculan las derivadas parciales

$$\frac{\partial L_D}{\partial x}, \frac{\partial L_D}{\partial y}, \frac{\partial L_D}{\partial z}, \frac{\partial L_P}{\partial x}, \frac{\partial L_P}{\partial y}$$

Igualando a cero se obtiene el siguiente sistema de ecuaciones:

$$Ax = b \tag{1.43}$$

$$A^{\mathrm{T}} y + z = c \tag{1.44}$$

$$x_j z_j = \mu,\ j = 1,...,n \tag{1.45}$$

Si definimos $X = \operatorname{diag}(x_1,...,x_n)$, $Z = \operatorname{diag}(z_1,...,z_n)$, y $e = [1 \ \ldots \ 1]^{\mathrm{T}}$ el vector de unos n-dimensional, entonces obtenemos el sistema de $(2n+m)\times(2n+m)$

$$\begin{aligned} Ax - b &= 0 \\ A^{\mathrm{T}} y + z - c &= 0 \\ XZe - \mu e &= 0 \end{aligned}$$

Si es posible encontrar vectores[3] $x > 0$, $z > 0$ y y, tal que satisfagan el anterior sistema de ecuaciones, y desde que $\mu \to 0$, entonces los vectores x y y serían las soluciones óptimas de los problemas primal y dual, respectivamente. Observe que si (x, y, z) satisface el anterior sistema, entonces se están verificando las condiciones necesarias y suficientes de Karush-Kuhn-Tucker para programación lineal (Bazaraa, Sherali y Shetty, 1993).

El sistema anterior es resuelto aplicando el método de Newton, para eso necesitamos fijar un parámetro inicial $\mu = \mu_0$, y establecer una solución inicial compuesta de $x^{(0)}$, $y^{(0)}$ y $z^{(0)}$, sean estos

$$x^{(0)} = \begin{bmatrix} x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix},\ z^{(0)} = \begin{bmatrix} z_1^{(0)} \\ \vdots \\ z_n^{(0)} \end{bmatrix} \text{ y } y^{(0)} = \begin{bmatrix} y_1^{(0)} \\ \vdots \\ y_m^{(0)} \end{bmatrix}$$

El único requerimiento es que $x^{(0)}$ y $z^{(0)}$ tengan todas sus componentes positivas, mientras que las componentes de $y^{(0)}$ no tienen restricción de signo, por otro lado μ_0 debe ser inicialmente grande.

[3] La relación de orden es componente a componente, debido a que $\Re^n$ no es un campo ordenado.

Definiendo $d_P = b - Ax^{(0)}$ y $d_D = A^{\mathrm{T}} y^{(0)} + z^{(0)} - c$, y ejecutando una iteración Newton, obtenemos los vectores componentes de un *paso newton*

$$dy = (AZ^{-1}XA^{\mathrm{T}})^{-1}(b - \mu AZ^{-1}e - AZ^{-1}Xd_D)$$
$$dz = -d_D - A^{\mathrm{T}}(AZ^{-1}XA^{\mathrm{T}})^{-1}(b - \mu AZ^{-1}e - AZ^{-1}Xd_D)$$
$$dx = Z^{-1}[\mu_0 e - XZe - X(-d_D - A^{\mathrm{T}}(AZ^{-1}XA^{\mathrm{T}})^{-1}(b - \mu AZ^{-1}e - AZ^{-1}Xd_D))]$$

Teniendo calculado el *paso de newton* (dx, dy, dz) podemos usar dx como una dirección en el $x - espacio$ y (dy, dz) como una dirección en el $(y, z) - espacio$. Luego, debemos elegir la longitud de los pasos en cada espacio de modo que se preserve $x^{(1)} > 0$ y $z^{(1)} > 0$, esto se consigue mediante el *criterio de la razón*: el cual consiste en calcular

$$\alpha_P = 0.995 \min_j \left\{ \frac{-x_j^{(0)}}{dx_j} \;\middle|\; dx_j < 0 \right\}$$
$$\alpha_D = 0.995 \min_j \left\{ \frac{-z_j^{(0)}}{dz_j} \;\middle|\; dz_j < 0 \right\}$$

donde el factor 0.995 nos mantiene en el interior del poliedro de factibilidad y al margen de la frontera.

Entonces, para evitar que $x^{(1)}$ y $z^{(1)}$ tengan componentes no positivas, simplemente nos desplazamos α_P veces en la dirección del vector dx en el x-espacio, α_D veces en la dirección (dy, dz) en el (y, z)-espacio. Por lo tanto, los nuevos puntos estarán dados por

$$x^{(1)} = x^{(0)} + \alpha_P dx \tag{1.46}$$
$$y^{(1)} = y^{(0)} + \alpha_D dy \tag{1.47}$$
$$z^{(1)} = z^{(0)} + \alpha_D dz \tag{1.48}$$

Todo eso constituye una iteración del método. Para la siguiente iteración $x^{(1)}$, $y^{(1)}$ y $z^{(1)}$ son considerados iniciales y el valor de μ_1 es escogido tal que $0 < \mu_1 < \mu_0$.

El método de Puntos Interiores Primal-Dual construye una sucesión convergente $\left\{\left(x^{(k)}, y^{(k)}, z^{(k)}\right)\right\}_{k\in N}$ tal que $\left\{x^{(k)}\right\}_{k\in N}$ converge a una solución óptima para el Problema de Programación Lineal en la forma estándar. Debemos aclarar que a diferencia del Método Simplex, cuando existen soluciones alternativas óptimas, el punto óptimo límite de la sucesión no necesariamente es un extremo del poliedro de factibilidad.

Algoritmo de Puntos Interiores Primal-Dual para Programación Lineal

Inicio: Dados los vectores iniciales $x, z \in \Re^n$ con $x, z > 0$, y el vector inicial $y \in \Re^m$. Definir la precisión requerida $\varepsilon > 0$, el n-vector columna $e = [1 \quad 1 \quad \cdots \quad 1]^T$ y el valor inicial de μ.

Paso 1: Si $\left|\dfrac{c^T x - b^T y}{1+\left|b^T y\right|}\right| < \varepsilon$ parar el algoritmo, la aproximación a la solución óptima para PL es x. Caso contrario ir al Paso 2.

Paso 2: Defina las matrices diagonales $X = \text{diag}(x)$ y $Z = \text{diag}(z)$.

Paso 3: Calcule $d_P = b - Ax$ y $d_D = A^T y + z - c$.

Paso 4: Halle dy desde el sistema $\left(AZ^{-1}XA^T\right)dy = b - \mu AZ^{-1}e - AZ^{-1}Xd_D$

Paso 5: Calcule $dz = -d_D - A^T dy$

Paso 6: Halle dx resolviendo el sistema $Zdx = \mu e - XZe - Xdz$

Paso 7: Calcule $\alpha_P = 0.995 \min_j \left\{\dfrac{-x_j}{dx_j} \;\middle|\; dx_j < 0\right\}$

Paso 8: Calcule $\alpha_D = 0.995 \min_j \left\{\dfrac{-z_j}{dz_j} \;\middle|\; dz_j < 0\right\}$

Paso 9: Hacer $\begin{cases} x \leftarrow x + \alpha_P dx \\ y \leftarrow y + \alpha_D dy \\ z \leftarrow z + \alpha_D dz \end{cases}$

Paso 10: Hacer[4] $\mu = \frac{c^T x - b^T y}{\Theta(n)}$ y volver al Paso 1.

Debemos recalcar que el número de iteraciones que el Algoritmo de Puntos Interiores Primal Dual realiza es aproximadamente un múltiplo constante de nL, donde n es el número de variables y L una "constante" que aumenta, lamentablemente, cuando las dimensiones del problema crecen. Cuando esto sucede se dice que el algoritmo es del orden polinomial y se le representa por $O(nL)$, es por este motivo que este algoritmo es considerado uno de los más eficientes para problemas de programación lineal.

Note que el Simplex requiere aproximadamente un número de iteraciones el cual es un múltiplo constante de 2^n para ciertos tipos de problemas PL, lo que lo hace teóricamente ineficiente, en este caso se dice que el Simplex es de orden exponencial, y se representa por $O(2^n)$. En el capítulo 2 trataremos sobre la eficiencia de algoritmos.

El Método de Newton

Presentaremos brevemente el método de Newton, el método resuelve un sistema de ecuaciones no lineales, el cual, consiste en determinar simultáneamente, el cero de un conjunto de funciones de *n* variable. La solución para este problema puede ser obtenida por el método de Newton-Raphson, que es una generalización del método de Newton para el cálculo del cero de la función unidimensional.

Sea $F : R^n \to R^n$ un vector de funciones no lineales F_i definidas $R^n \to R$, con las siguientes propiedades:

(i) Existe $v^* \in R^n$, tal que $F(v^*) = 0$;

[4] Observe que $\Theta(n) = n^2$ cuando $n \leq 5000$, mientras que $\Theta(n) = n\sqrt{n}$ cuando $n > 5000$, n es el número de variables que intervienen en el problema.

(ii) La función F es continua y diferenciable en la vecindad de v^*;

(iii) Para cualquier v diferente de v^*, las matrices de las derivadas parciales de F, denominada matriz Jacobiana (J) es no singular.

El método de Newton para determinar $F(v^*)=0$ puede ser derivado usando la serie de Taylor para expandir F en las vecindades de un punto v^k,

$$F(v^k+\Delta v^k)=F(v^k)+J(v^k)\Delta v^k+O(\Delta v^k)^2.$$

Eliminando los términos de orden dos superior y haciendo $F(v^k+\Delta v^k)=0$, se tiene:

$$J(v^k)\Delta v^k=-F(v^k)$$

La solución de este sistema de ecuaciones lineales proporciona una dirección de traslado, simultáneamente, cada función más próxima de un cero. Las ecuaciones de este sistema son conocidas como ecuaciones de Newton y Δv^k como dirección de Newton.

El nuevo punto es calculado por $v^{k+1}=v^k+\Delta v^k$.

El proceso finaliza en el siguiente criterio de parada

$$\sum_{i=1}^{n}\| F_i \|<\varepsilon_1 \quad \text{o} \quad \sum_{i=1}^{n}\left|\Delta v^k\right|<\varepsilon_2 .$$

donde $\varepsilon_1, \varepsilon_2>0$ son las tolerancias (o errores) pre- establecidas.

La Figura 1.6 representa el algoritmo de Newton-Raphson para calcular la solución de un sistema de ecuaciones no lineales.

Algoritmo Newton_Raphson

{Objetivo : Resolver sistemas no lineales.}

parámetros de entrada v^0, *toler*, F

{solución inicial, tolerancia, vector de funciones no lineales}

parámetros de salida v {solución}

$k \leftarrow 0$

mientras que *critério de convergencia* > *toler* hacer

Calcule $J(v^k)$, $J_{i,j} = \left.\frac{\partial F_i}{\partial v_j}\right|_{v_j^k}$, $i, j = 1, \ldots, n$

Determine Δv^k tal que $J(v^k)\Delta v^k = -F(v^k)$

$v^{k+1} \leftarrow v^k + \Delta v^k$

$k \leftarrow k + 1$

fin de mientras

$v \leftarrow v^{k+1}$

fin del algoritmo

Figura 1.6. Algoritmo de Newton – Raphson

1.2.4 El Método de Puntos Interiores Predictor-Corrector

El método de Puntos Interiores Predictor-Corrector (MPIPC) es una variante del prototipo original del algoritmo del MPI. Esta variante se adapta a elegir el parámetro μ (que lo veremos posteriormente), forzando al algoritmo en las iteraciones en realizar muchos pequeños pasos en una cierta vecindad de la trayectoria central (o dirección de centrado), que es esencial a la prueba del algoritmo polinomial. En estos pasos incluye el método escalado afín, el cual determina la dirección.

Además Salahi, Peng y Terlaky (2005), comprueban que el algoritmo puede terminar después de a lo más de

$$O\left(n^2 \log\left(x^0\right)^T z^0 / \varepsilon \right)$$

iteraciones, pero modificando ligeramente el sistema de Newton en el paso corrector, se reduce la complejidad de la iteración en

$$O\left(n \log \left(x^0\right)^T z^0 / \varepsilon \right).$$

Algoritmo Primal-Dual Clásico

La desventaja del método Escalado Afín son sus variables (*x, z*) que se aproximan a sus límites muy rápidamente. Consecuentemente las direcciones son muy distorsionadas y el método converge lentamente, pues $x_i z_i \approx 0$. Para evitar que esto ocurra se añadirá una perturbación μ en la condición de complementariedad $x_i z_i = 0$. En su lugar consideramos $x_i z_i = \mu$, es decir el método primal-dual resuelve el sistema de ecuaciones no lineales (1.49).

$$F_\mu(x, y, z) = \begin{bmatrix} Ax - b \\ A^T y + z - c \\ XZ\, e - \mu e \end{bmatrix} = 0, \quad (x, z) \geq 0 \tag{1.49}$$

Chvátal (1983) confirma que el algoritmo de puntos interiores primal-dual clásico, pueden ser vistos como una aplicación del método de Newton para calcular aproximaciones de la solución de una sucesión de sistemas no lineales (1.50) perturbados por un parámetro μ, originado de la utilización de la función barrera para relajar las restricciones de no negatividad.

Observemos que las dos primeras ecuaciones son lineales y forman una viabilidad primal y dual. La tercera ecuación es no lineal y tiene la condición de complementariedad cuando $\mu = 0$ entonces los problemas (1.50) y (1.49) son equivalentes. Luego, la solución del problema (1.49) se aproxima a la solución del problema primal-dual cuando $\mu \to 0$.

$$\begin{aligned} Ax - b &= 0 \\ A^T y + z &= c \\ XZe &= 0 \\ (x, z) &\geq 0 \end{aligned} \tag{1.50}$$

Un algoritmo primal-dual obtiene una solución aproximada para el problema generando una secuencia de puntos (x^k, y^k, z^k) y de parámetro μ_k. El conjunto de puntos (x^k, y^k, z^k) satisface (1.49) para todo $\mu > 0$ es denominado trayectoria central y toda solución (*x,y,z*) no negativa es llamada centro analítico. Dada una solución y un parámetro μ asociado, se puede medir el error en la complementariedad calculando el llamado *gap de complentariedad,* dado por

$$g = x^T z$$

Si la solución fuera viable, ese valor sería reducido al usual *gap de dualidad de* $(g = \mu e^T e = n\mu)$.

A cada iteración del algoritmo es aplicado un paso del método de Newton para resolver el sistema (1.49), con un parámetro dado μ_k.

La dirección de Newton es obtenida de la solución del siguiente sistema de ecuaciones lineales:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta z^k \end{bmatrix} = \begin{bmatrix} b - Ax^k \\ c - A^T y^k - z^k \\ \mu_k e - X^k Z^k e \end{bmatrix} = \begin{bmatrix} r_p \\ r_d \\ r_{\mu_k} \end{bmatrix} \qquad (1.51)$$

El nuevo punto es calculado haciendo,

$$(x^{k+1}, y^{k+1}, z^{k+1}) = (x^k, y^k, z^k) + (\alpha_P \Delta x^k, \alpha_D \Delta y^k, \alpha_D \Delta z^k),$$

siendo α_P y α_D los pasos primal y dual, respectivamente. Esos valores son determinados para preservar la no negatividad de las variables *x* y *z*.

El parámetro μ_k es decreciente y el proceso se repite hasta que sea encontrada una solución suficientemente próxima de la solución óptima o sea detectada la inviabilidad del problema.

En la Figura 1.7 se presenta un algoritmo donde el parámetro de barrera o el cálculo de perturbación es

$$\mu_k = \lambda \frac{(x^k)^T z^k}{n}, \text{ siendo } \lambda \in (0,1),$$

el valor $\frac{(x^k)^T z^k}{n}$ será una "medida" de la distancia media (promedio) de un punto a una solución óptima, y los tamaños máximos para los pasos son determinados por

$$\alpha_P = máx\left\{\alpha \in [0,1] / x^k + \alpha \Delta x^k \geq 0\right\}, \ \alpha_D = máx\left\{\alpha \in [0,1] / z^k + \alpha \Delta z^k \geq 0\right\}.$$

Este algoritmo es conocido como primal-dual inviable. Las iteraciones generadas son interiores mas no satisfacen, necesariamente, las restricciones de igualdad.

A cada iteración del algoritmo es necesario resolver un sistema de ecuaciones lineales para determinar la dirección de Newton. Algunas variantes de métodos de puntos interiores resuelven a cada iteración varios sistemas con la misma matriz de los coeficientes y con eso determinan una mejor dirección, reduciendo el número total de iteraciones y, consecuentemente, el número de factorizaciones.

Algoritmo Predictor-Corrector

El algoritmo predictor-corrector propuesto por Mehrotra (1992) difiere del presentado en la sección anterior, en el cálculo de las direcciones de Newton. La dirección $\Delta = (\Delta x, \Delta y, \Delta z)$ es descompuesta en dos partes $\Delta = \Delta_a + \Delta_c$.

El término Δ_a es obtenido resolviendo el sistema (1.51) para $\mu = 0$, o sea,

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta_a x^k \\ \Delta_a y^k \\ \Delta_a z^k \end{bmatrix} = \begin{bmatrix} r_p \\ r_d \\ -X^k Z^k e \end{bmatrix} \qquad (1.52)$$

La componente Δ_a, es conocida como dirección Escalado Afín (o vectores de desplazamiento del Escalado Afín), posteriormente mostraremos su obtención (p.45).

La función de Δ_a cumple un rol muy importante que es, el de ser responsable de determinar una mejor predicción para el parámetro de la Barrera. Este parámetro es escogido (elegido) usando la siguiente heurística:

$$\mu_k = \left(\frac{\left(x^k + \alpha_{pa}\Delta_a x^k\right)^T \left(z^k + \alpha_{da}\Delta_a z^k\right)}{\left(x^k\right)^T z^k} \right)^2 \left(\frac{\left(x^k + \alpha_{pa}\Delta_a x^k\right)^T \left(z^k + \alpha_{da}\Delta_a z^k\right)}{n} \right)$$

siendo α_{pa} y α_{da} los pasos que preservan la no negatividad de las variables *x* y *z*.

La dirección correctora, Δ_c es dada por,

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta_c x^k \\ \Delta_c y^k \\ \Delta_c z^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mu_k e - \Delta_a X^k \Delta_a Z^k e - XZe \end{bmatrix} \tag{1.53}$$

donde, $\Delta_a X^k = diag(\Delta_a x^k)$, $\Delta_a Z^k = diag(\Delta_a z^k)$.

El término Δ_c tiene por objetivo hacer que la nueva iteración sea la más centrada posible. Notemos que, el vector de términos independientes depende de la solución del sistema (1.52). La obtención de la dirección correctora Δ_c también se mostrara posteriormente (p.46).

El aumento de trabajo ocurrido por resolver dos sistemas lineales a cada iteración es compensado considerando la reducción significativa en el número de iteraciones como lo muestra el Algoritmo Predictor-Corrector en la Figura 1.8.

Algoritmo Primal - Dual Clásico

Parámetros de entrada $\left(x^0, y^0, z^0\right)$, $toler$

$\{$solución inicial, tolerancia$\}$

parámetros de salida (x, y, z), $\{$solución$\}$

$k \leftarrow 0$

Determine λ tal que $\lambda \in (0,1)$

$\mu^k \leftarrow \lambda\left(x^0\right)^T z^0 / n$

Mientras $criterio\ de\ convergencia > toler$ hacer

Calcule $\left(\Delta x^k, \Delta y^k, \Delta z^k\right)$ resolviendo el sistema (1.51)

Determine α_P, α_D

$\left(x^{k+1}, y^{k+1}, z^{k+1}\right) \leftarrow \left(x^k, y^k, z^k\right) + \left(\alpha_P \Delta x^k, \alpha_D \Delta y^k, \alpha_D \Delta z^k\right)$

$\mu^{k+1} \leftarrow \lambda\left(x^{k+1}\right)^T z^{k+1} / n$

$k \leftarrow k+1$

fin de mientras

$(x, y, z) \leftarrow \left(x^{k+1}, y^{k+1}, z^{k+1}\right)$

Fin del algoritmo

Figura 1.7. Algoritmo Primal-Dual Clásico

Algoritmo Predicctor - Corrector

{Objetivo: Resolver PPL usando algoritmo predictor - corrector}

parámetros de entrada $\left(x^0, y^0, z^0\right)$, *toler*

{solución inicial, tolerancia}

parámetros de salida (x, y, z), {solución}

$iter \leftarrow 0$

Mientras *criterio de convergencia* $>$ *toler* hacer

PREDICTOR

Resolver (1.52) calculando $\left(\Delta_a x^k,\ \Delta_a y^k,\ \Delta_a z^k\right)$

Determine $\alpha_P,\ \alpha_D$

Obtener el resultado del paso PREDICTOR

$$\left(x^{k+1}, y^{k+1}, z^{k+1}\right) \leftarrow \left(x^k, y^k, z^k\right) + \left(\alpha_P \Delta_a x^k, \alpha_D \Delta_a y^k, \alpha_D \Delta_a z^k\right)$$

CORRECTOR

Calcular el Parámetro de centrado

$$\mu_{k+1} = \left(\frac{\left(x^k + \alpha_{pa}\Delta_a x^k\right)^T \left(z^k + \alpha_{da}\Delta_a z^k\right)}{\left(x^k\right)^T z^k} \right)^2 \left(\frac{\left(x^k + \alpha_{pa}\Delta_a x^k\right)^T \left(z^k + \alpha_{da}\Delta_a z^k\right)}{n} \right)$$

Calcule $\left(\Delta_c x^k,\ \Delta_c y^k,\ \Delta_c z^k\right)$ resolviendo el sistema (1.53) con μ_{k+1}

Determine $\alpha_P,\ \alpha_D$

Obtener el resulrado del paso CORRECTOR

$$\left(x^{k+1}, y^{k+1}, z^{k+1}\right) \leftarrow \left(x^k, y^k, z^k\right) + \left(\alpha_P \Delta_c x^k, \alpha_D \Delta_c y^k, \alpha_D \Delta_c z^k\right)$$

$k \leftarrow k + 1$

fin de mientras

$(x, y, z) \leftarrow \left(x^{k+1},\ y^{k+1},\ z^{k+1}\right)$

Fin del algoritmo

Figura 1.8. Algoritmo Predictor-Corrector

OBTENCIÓN DE LA DIRECCIÓN DEL ESCALADO AFÍN (Δ_a)

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta_a x^k \\ \Delta_a y^k \\ \Delta_a z^k \end{bmatrix} = \begin{bmatrix} r_p \\ r_d \\ r_k \end{bmatrix} \tag{1.52}$$

definimos

$$r_p = b - Ax^k$$
$$r_d = c - A^T y^k - z^k$$
$$r_k = -X^k Z^k e$$

Considerando $X = X^k$, $Z = Z^k$, $x = x^k$, $y = y^k$, $z = z^k$, $\mu = \mu_k$, tenemos que $X = diag\,(x)$, $Z = diag(z)$

El sistema (1.32) podemos verlo como

$$A\Delta_a x = r_p \tag{1.52_a}$$
$$A^T \Delta_a y + \Delta_a z = r_d \tag{1.52_b}$$
$$Z\Delta_a x + X\Delta_a z = r_k = -XZe \tag{1.52_c}$$

Ahora multiplicamos a ambos miembros de (1.52_b) por $AZ^{-1}X$ tenemos

$$AZ^{-1}X\left(A^T \Delta_a y\right) + AZ^{-1}X\,\Delta_a z = AZ^{-1}X\,r_d$$

Tomando en cuenta la ecuación (1.52_c), además notando que $AXe = Ax$ y $Ax = b - r_p$

$$
\begin{aligned}
\left(AZ^{-1}XA^T\right)\Delta_a y &= -AZ^{-1}\left(X\Delta_a z\right)+AZ^{-1}Xr_d \\
&= -AZ^{-1}\left(-XZe-Z\Delta_a x\right)+AZ^{-1}Xr_d \\
&= AZ^{-1}XZe+AZ^{-1}Z\,\Delta_a x+AZ^{-1}Xr_d \\
&= AXe+A\Delta_a x+AZ^{-1}Xr_d \\
&= Ax+r_p+AZ^{-1}Xr_d \\
&= (b-r_p)+r_p+AZ^{-1}Xr_d \\
&= b+AZ^{-1}Xr_d
\end{aligned}
$$

Entonces para obtener $\Delta_a y$ resolvemos el sistema lineal

$$\left(AZ^{-1}XA^T\right)\Delta_a y = b+AZ^{-1}Xr_d \qquad (1.52\_d)$$

Usando (1.52_b), (1.52_c) y $\Delta_a y$ de (1.52_d), hallamos también $\Delta_a z$ y $\Delta_a x$ mediante

$$
\begin{aligned}
\Delta_a z &= r_d - A^T\Delta_a y \\
\Delta_a x &= Z^{-1}\left(-XZe-X\,\Delta_a z\right)
\end{aligned}
$$

OBTENCIÓN DE LA DIRECCIÓN CORRECTORA (Δ_c)

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{bmatrix}\begin{bmatrix} \Delta_c x^k \\ \Delta_c y^k \\ \Delta_c z^k \end{bmatrix} = \begin{bmatrix} r_p \\ r_d \\ r_{\mu k} \end{bmatrix} \qquad (1.53)$$

definimos

$$
\begin{aligned}
r_p &= b-Ax^k \\
r_d &= c-A^T y^k - z^k \\
r_{\mu k} &= \mu_k e-\Delta_a X^k\,\Delta_a Z^k e-X^k Z^k e
\end{aligned}
$$

Considerando $X = X^k$, $Z = Z^k$, $x = x^k$, $y = y^k$, $z = z^k$, $\mu = \mu_k$, tenemos que

$\Delta_a X = diag(\Delta_a x), \Delta_a Z = diag(\Delta_a z)$.

El sistema (1.53) podemos verlo de la siguiente forma

$$A\Delta_c x = r_p \qquad (1.53\_a)$$

$$A^T \Delta_c y + \Delta_c z = r_d \qquad (1.53\_b)$$

$$Z\Delta_c x + X\Delta_c z = r_{\mu k} = \mu e - \Delta_a X \, \Delta_a Z \, e - XZe \qquad (1.53\_c)$$

Aplicando los mismos pasos que vimos anteriormente en la dirección del Escalado Afín multiplicando a ambos miembros de (1.53_b) por $AZ^{-1}X$ conseguimos

$$\left(AZ^{-1}XA^T\right)\Delta_c y = b - \mu \, AZ^{-1}e + AZ^{-1}\Delta_a X \, \Delta_a Z \, e + AZ^{-1}Xr_d \qquad (1.53\_d)$$

Además obtenemos $\Delta_c z$ y $\Delta_c x$ donde

$$\Delta_c z = r_d - A^T \Delta_c y$$

$$\Delta_c x = Z^{-1}\left(\mu e - \Delta_a X \, \Delta_a Z \, e - XZe - X\Delta_c z\right)$$

Capítulo 2

Complejidad Computacional

2.1 Introducción

En optimización frecuentemente se trabaja con métodos especializados en la solución de determinados problemas, tales métodos son resumidos en algoritmos. En la primera sección trataremos sobre “el algoritmo más eficiente”, donde conoceremos los parámetros que nos permitan comparar y clasificar a los algoritmos de acuerdo a su eficiencia.

Por ejemplo, el Algoritmo Simplex para resolver el Problema de Programación Lineal es considerado ineficiente desde el punto de vista teórico, aunque en la práctica está probado tener un buen comportamiento cuando es sometido a problemas con un número razonable de variables y restricciones. En realidad, el Simplex es considerado ineficiente debido a que el número de iteraciones está vinculado a una cantidad exponencial. Por otro lado, después que en 1978 fue publicado el primer algoritmo teóricamente eficiente, muchos nuevos algoritmos surgieron, cada uno mejor que el anterior.

En la primera sección estableceremos los parámetros con los que diferenciaremos un algoritmo de tiempo polinomial de un algoritmo de tiempo exponencial, tales criterios que permiten esta comparación son cruciales para la caracterización de un algoritmo como bueno o malo.

La segunda sección está destinada a estudiar la complejidad computacional de problemas, consiste en clasificar los problemas de acuerdo al grado de dificultad que presentan en su resolución, teniendo en cuenta la existencia o no de algoritmos eficientes para su solución.

La complejidad computacional de problemas es un campo que camina paralelamente con los algoritmos, allí estudiaremos la “dificultad” inherente del problema. Así por ejemplo, la clase P es la clase donde pertenecen los problemas fáciles de resolver. La clase NP contiene los problemas cumpliendo determinadas hipótesis, con lo cual es posible esta-

blecer si un problema está en la clase NP -completo, si esto sucediera, entonces tal problema es catalogado como difícil de resolver, debido a que requiere un tiempo y espacio extremadamente grande cuando el problema crece en dimensión.

En resumen, en este capítulo mostraremos que el Problema de Programación Lineal Entera es un problema NP -completo, el cual marca la diferencia con el Problema de Programación Lineal clásico, debido a que este último es considerado de fácil resolución.

2.2 Eficiencia de Algoritmos

Un algoritmo es un conjunto de instrucciones para resolver un problema, por lo general, los algoritmos resumen métodos. Por otro lado, un método es un procedimiento, con las sustentaciones matemáticas necesarias, por el cual se resuelve un determinado problema. De este modo, cuando se refiere a la construcción de un programa, nos estamos refiriendo a la construcción de un algoritmo. El algoritmo no es un concepto proveniente del campo de la computación, sino que es un término matemático (Victor Valenzuela, 2003).

El objetivo de esta sección es clasificar a los algoritmos de acuerdo a su eficiencia, para lo cual estableceremos los formalismos necesarios. Existen dos criterios para medir la eficiencia de un algoritmo respecto a un determinado problema:

1. El criterio empírico: el cual consiste en implementar el algoritmo en algún lenguaje de programación y con la ayuda de un computador, someterlo a diversos casos del problema para determinar su comportamiento. Claramente, con este criterio no tenemos garantía del comportamiento del algoritmo para cualquier caso de este problema, debido a que el número de casos experimentados es finito.

2. El criterio teórico: consiste en determinar matemáticamente la cantidad de recursos necesarios, el tiempo de ejecución y el espacio necesario de memoria de computador para guardar los datos. Estos recursos deben ser expresados en función del tamaño del problema (en número de restricciones y variables). Una de las ventajas en este criterio es que no depende del computador que se esté usando, del lenguaje de programación, de las habilidades del programador, etc. Otra ventaja es que nos permite estudiar la eficiencia de un algoritmo con relación a problemas de cualquier tamaño.

Al analizar la eficiencia de un algoritmo con el criterio teórico, el tiempo puede estar expresado justamente en unidades tiempo, pero también puede estar expresado en término del número de operaciones elementales (asignaciones, comparaciones, sumas y multiplicaciones), y algunas veces en términos del número de iteraciones que realiza el algoritmo.

Asumiremos en lo sucesivo que una operación elemental demanda una unidad de tiempo para ejecutarse.

Definición 2.1. Un ejemplar de un problema es un miembro particular de éste, donde se conocen los datos numéricos de modo explícito

Por ejemplo, considere el problema que consiste en multiplicar dos números enteros a y b, entonces el ejemplar asociado a este problema consiste en ingresar los datos (números) $a = 8$ y $b = 18$.

Un Problema Combinatorio es aquel donde el conjunto de las posibles soluciones es finito, aunque algunas veces se exige que dicho conjunto sea al menos infinito numerable. A continuación presentaremos brevemente algunos de los muchos problemas clásicos de tipo combinatorio, ellos serán frecuentemente mencionados cuando se analice algoritmos y se estudie la complejidad de un problema.

El Problema de Programación Lineal (*PL*)

El problema de PL es utilizado ampliamente en el primer capítulo puede ser caracterizado por un conjunto G y una función de costo c, de modo que

$$G = \left\{ x \in \Re^n \mid Ax = b, x \geq 0 \right\}$$
$$c : x \mapsto c^{\mathrm{T}} x$$

donde $b \in \Re^m$, $c \in \Re^n$, $A \in \Re^{mxn}$ y m,n enteros positivos, se trata de minimizar c sobre G. Entendemos por un ejemplar del problema de PL a la entrada de los datos suficientes del problema de modo que este bien determinado. Esto es, ingresar numéricamente las componentes de la matriz A, el vector b y el vector c.

El Problema de la Factibilidad Booleana (*SAT*)

El problema *SAT* es el problema central de la lógica matemática y la teoría de la computación. *SAT* es fundamental para la solución de varios problemas en razonamiento automático, diseño y manufactura asistida por computadoras, planificación, visión computacional, base de datos, robótica, diseño de circuitos integrados, de arquitectura de computadora y de redes de computadora, entre otros.

El problema *SAT* es el siguiente: consideremos las variables del tipo booleano *x*, esto es, variables que pueden asumir valores "verdad" o "falso", sean los conectivos lógicos de disyunción "$\vee$", conjunción "$\wedge$" y negación "~", los cuales denotamos por "+", ".", y "$\bar{x}$", respectivamente. (Papadimitriou y Kenneth, 1982).

Una fórmula booleana es una expresión del tipo:

$$(\bar{x}_1 + \bar{x}_2)\cdot(x_2 + x_3)\cdot(x_1)$$

Las fórmulas entre paréntesis son llamadas "clausulas". Si damos un valor para cada variable booleana, entonces podemos evaluar la fórmula completa, si esta tuviera un valor *verdadero*, entonces diremos que la fórmula es "satisfecha". Cuando el valor total de la fórmula es "*falso*", entonces diremos que la fórmula es "*no-satisfecha*".

De un modo general, el problema de "Factibilidad Booleana" es el siguiente:

Sean *m* clausulas $c_1, c_2, \ldots, c_m$ con respecto a las variables, $x_1, x_2, \ldots, x_n$, ¿la fórmula $c_1, c_2, \ldots, c_m$ es satisfecha?

El Problema de Programación Lineal Entera (PLE)

Este problema es uno de los mayores representantes dentro del mundo de la optimización combinatoria.

Sea $A \in Z^{mxn}$ una matriz, $x \in Z^n$, $c \in Z^n$ y $b \in Z^m$, el problema es

$$\text{Minimizar} \quad c^{\mathrm{T}} x$$

Sujeto a:

$$Ax = b$$

$$x \geq 0$$

$$x \quad \text{entero}$$

Notemos que estos y otros problemas populares en el mundo de la optimización combinatoria son del tipo solucionables, esto es, que existe un algoritmo para cada uno de ellos capaz de obtener una solución. (Papadimitriou y Kenneth, 1982).

2.2.1 El Principio de Invariabilidad

Un algoritmo está correctamente definido si éste resuelve el problema para todos sus ejemplares asociados. Para demostrar que un algoritmo es incorrecto, lo que tenemos que hacer es encontrar un ejemplar para el cual no es posible encontrar una respuesta correcta. Por otro lado, mostrar que un algoritmo es correcto no es tarea fácil y escapa al objetivo de este texto. Podemos decir entonces que es más fácil mostrar que un algoritmo es incorrecto que correcto.

Supongamos que tenemos tres computadoras C_1, C_2 y C_3 en las cuales ejecutamos una implementación del mismo algoritmo Λ. Si el sistema C_2 es dos veces más veloz que C_1, entonces probablemente C_2 ejecute el algoritmo en la mitad del tiempo que C_1. Lo mismo pasaría si usamos la computadora C_3 que es tres veces más rápida que C_1, ejecutaría el algoritmo en una tercera parte del tiempo usado por C_1 y dos terceras partes del tiempo usado por C_2. Como podemos observar, el tiempo de ejecución de estas tres implementaciones difiere sólo por una constante multiplicativa.

Algo similar ocurriría si usáramos dos lenguajes de programación diferentes. Si por un lado, implementamos nuestro algoritmo con el lenguaje C++, y por el otro, implementamos en MatLab. Claramente el tiempo de ejecución de la implementación hecha en C++ será mucho menor que el tiempo usado al ejecutar nuestro algoritmo usando MatLab. Lo mismo ocurre si la implementación depende de la habilidad y técnica del programador.

Debido a que no existe una computadora estándar dónde comparar los tiempos de ejecución, ni formas únicas de programar, es que se adoptó el *principio de invariabilidad*. Mediante el principio de invariabilidad dos implementaciones del mismo algoritmo no se diferencian en eficiencia por más de constante multiplicativa. Es decir, si dos implementaciones del mismo algoritmo toman $t_1(n)$ y $t_2(n)$ segundos, respectivamente, para resolver un ejemplar de un determinado problema de tamaño n, entonces debe existir una constante positiva c tal que $t_1(n) \leq c \cdot t_2(n)$ para $n \geq n_0$ suficientemente grande. Con esto, podemos medir la eficiencia del algoritmo usando $t_1(n)$ o $t_2(n)$, pues para nuestros principios teóricos, las dos implementaciones del algoritmo son prácticamente iguales diferenciándose apenas por una constante multiplicativa.

En resumen, un cambio de máquina sólo nos permitirá resolver un ejemplar de un problema 10 ó 20 veces más rápido, pero sólo un cambio de algoritmo nos dará una mejora substancial a medida que el tamaño del ejemplar aumente**.**

Decimos que un algoritmo toma un tiempo $O(t(n))$, y leemos *orden de* $t(n)$, donde t es una función y n representa la medida del ejemplar, si existe una constante positiva c y una implementación del algoritmo capaz de resolver todo ejemplar de tamaño n en un tiempo limitado superiormente por $c \cdot t(n)$ segundos. Debemos notar que el uso segundos no es indispensable, pues pudimos haber usado horas como unidad de tiempo, bastando sólo cambiar la constante para limitar el tiempo por $c_1 \cdot t(n)$ horas, donde $c_1 = \frac{c}{3600}$. O pudimos usar minutos, variando el límite superior para $c_2 \cdot t(n)$ minutos.

Como vemos, el uso de segundos o cualquiera otra unidad de tiempo no hacen diferencia para nuestros principios teóricos, pues quedan absorbidos por el principio de invariabilidad.

En realidad, el uso de alguna unidad de tiempo no es indispensable para analizar el comportamiento de un algoritmo, pues basta conocer por ejemplo el número de operaciones elementales o el número de iteraciones. Digamos que $r(n)$ sea el número de operaciones elementales realizadas por el algoritmo, si conocemos la frecuencia con que opera un sistema de cómputo, digamos $c = 2 \times 10^9$ operaciones elementales por segundo, entonces es posible obtener el tiempo que tomará el algoritmo para resolver este ejemplar: $\frac{r(n)}{c}$

segundos. Este hecho nos permite analizar algoritmos sin considerar la unidad de tiempo, lo cual torna el trabajo mucho más cómodo.

2.2.2 Notación Asintótica

Anteriormente mencionamos el *orden* de tiempo que un algoritmo toma para resolver el ejemplar de algún problema, ahora formalizaremos este concepto.

Definición 2.2. Sean N e $\Re$ los conjuntos de los números naturales y reales, respectivamente, sea $\Re^+$ el conjunto de números reales positivos, y $f : N \mapsto \Re^+$ una función arbitraria. Nosotros definimos el conjunto

$$O(f(n)) = \left\{ t : N \mapsto \Re^+ \mid (\exists c \in \Re^+)(\exists n_0 \in N)(\forall n \geq n_0)\left[t(n) \leq cf(n)\right] \right\}. \tag{2.1}$$

Y lo llamamos "orden $f(n)$". El conjunto $O(f(n))$ representa todas las funciones reales t acotadas superiormente por un múltiplo real positivo de f, para n suficientemente grande $(n > n_0)$. (Brassard y Bratley, 1988, p.37)

Cabe resaltar que existen otras notaciones para el *orden*, tales como:

$$\Omega(f(n)) = \left\{ t : N \mapsto \Re^+ \mid (\exists c \in \Re^+)(\exists n_0 \in N)(\forall n \geq n_0)\left[t(n) \geq cf(n)\right] \right\}. \tag{2.2}$$

$\Omega(f(n))$ es el conjunto de todas las funciones t acotadas inferiormente por un múltiplo positivo real de f, para n suficientemente grande.

Tiempos de Ejecución

Una medida que suele ser conveniente conocer es el tiempo de ejecución de un algoritmo en función de *n*, lo que denominamos *T*(*n*). Esta función se puede medir físicamente (ejecutando el programa, reloj en mano), o calcularse sobre el código contando instrucciones a ejecutar y multiplicando por el tiempo requerido por cada instrucción.

Por ejemplo, un fragmento de programa como:

A1; FOR *i:=1* TO *n* DO *A2* END;

requiere: $T(n) = t_1 + t_2 * n$, siendo t_1 el tiempo que lleve ejecutar la serie "*A1*" de sentencias, y t_2 el que lleve la serie "*A2*".

Ejemplo 2.1

Consideremos el problema de ordenamiento de un arreglo $T[1..n]$ en forma creciente.

Existen varios algoritmos que resuelven este problema, nosotros analizaremos ellos con respecto al número de iteraciones realizadas:

- El algoritmo llamado *insertsort* requiere un número de iteraciones en el orden n^2, es decir $O(n^2)$, para resolver el peor ejemplar de este problema. Significa que *insertsort* requiere a lo más $c \cdot n^2$ iteraciones para resolver cualquier ejemplar del problema. En virtud de las observaciones últimas, por lo general decimos simplemente que "el algoritmo es $O(n^2)$".

- Por otro lado, el algoritmo denominado *merguesort* requiere apenas un número de iteraciones en el orden $n \log n$, es decir $O(n \log n)$. Esto significa que a lo más requiere $c \cdot n \cdot \log n$ iteraciones para resolver el peor ejemplar del problema.

En base a esto, podemos decir entonces que el algoritmo *merguesort* es mejor que *insertsort*. La utilización del "orden de" nos permite clasificar los algoritmos de una manera organizada. (Brassard y Bratley, 1988, p.7)

Ejemplo 2.2

Si un algoritmo requiere un tiempo exacto $T(n)$ segundos, donde n es el tamaño del ejemplar asociado a un determinado problema, tal que T está definido por

$$T(n) = n^3 + 7n^2 - n + 4$$

Entonces, por medio de la definición 2.2 el tiempo de ejecución de este algoritmo está en $O(n^3)$. En este caso decimos que $T(n)$ es del orden n^3.

La introducción de estos conceptos nos permite clasificar a los algoritmos como buenos o malos. Sean dos algoritmos Λ_1 y Λ_2, si Λ_1 es $O(n^3)$ y Λ_2 es $O(n^2 \log n)$, donde n representa el tamaño del ejemplar considerado, entonces ahora sabemos que el algoritmo Λ_2 es más eficiente que Λ_1. En estos casos, si un algoritmo es $O(2^n)$, entonces éste será catalogado como teóricamente malo.

Dado $O(T(n))$, si graficamos T para varios tamaños de entrada n, se puede observar su comportamiento. Esto nos permite visualizar el crecimiento de T a medida que n aumenta. Analizar el crecimiento de T cuando n crece asintóticamente nos permite a su vez ver cómo crece el tiempo de ejecución de los algoritmos que son de orden $T(n)$.

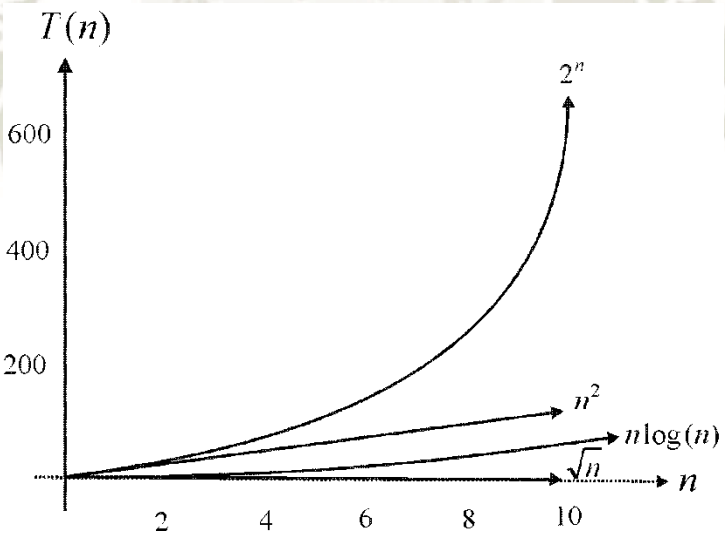


Figura 2.1. Interpretación gráfica del ($T(n)$) tiempo de ejecución para diferentes algoritmos

Dado $O(T(n))$, la figura 2.1 muestra diferentes casos cuando $T(n) = n^2$, $T(n) = n \log n$, $T(n) = \sqrt{n}$ y $T(n) = 2^n$. Observe que $T(n) = 2^n$ presenta la mayor rapidez de crecimiento, mientras que $T(n) = \sqrt{n}$ es la menor. El eje vertical representa $T(n)$, el tiempo de ejecución; y el eje horizontal representa n, el tamaño del ejemplar considerado.

Por último, se dice que un algoritmo es polinomial, o para ser más precisos, el tiempo de ejecución tomado por una implementación de este algoritmo está en el orden $p(n)$, si p es un polinomio en función del tamaño del ejemplar n. Análogamente, un algoritmo se dice exponencial, si su implementación requiere un tiempo en $O(a^n)$ para resolver cualquier ejemplar de tamaño n, donde $a > 1$.

Del anterior ejemplo 2.1, el algoritmo *insertsort* es entonces un algoritmo de tiempo polinomial, en virtud que éste es $O(n^2)$. Mientras que el Algoritmo Simplex que resuelve problemas de programación lineal, es un algoritmo exponencial debido a que su tiempo de ejecución está en $O(2^n)$, lo cual lo hace teóricamente ineficiente. Cabe mencionar que el algoritmo de puntos interiores primal-dual para programación lineal, visto al final del primer capítulo, es un algoritmo polinomial, debido a que éste está en $O(nL)$, donde n es el número de variables y L es una constante[5] que representa el tamaño del ejemplar, lo cual lo caracteriza como uno de los más eficientes dentro del campo de la optimización lineal.

Ejemplo 2.3

Considere el problema de las Torres de Hanoi: Se trata de N argollas, todas de diámetros diferentes, y 3 varas de sección circular A, B y C, donde se pueden apilar las argollas. En un principio, las N argollas forman una sola torre, apiladas de mayor a menor en A (el más pequeño arriba y el más grande abajo).

El problema consiste en pasar la torre del sitio A al sitio C, utilizando B como sitio auxiliar (también llamado pivote), observando las siguientes reglas:

a) Sólo puede pasarse un disco a la vez;

b) Ningún disco puede apoyarse sobre otro de menor tamaño.

[5] Constante que lamentablemente crece a medida que el ejemplar aumenta de tamaño.

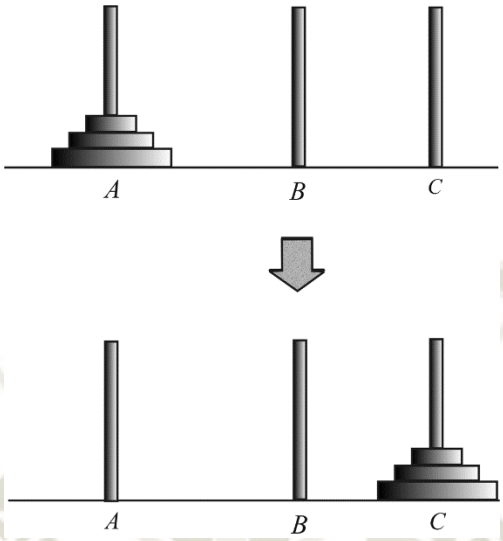


Figura 2.2.

El algoritmo asociado a este problema es de tiempo exponencial del orden 2^n, es decir $O(2^n)$, y es de naturaleza recursiva. Note que el tiempo de ejecución exacto está dado por $t(n) = 2t(n-1) + \text{const}$, donde n representa el tamaño del ejemplar y const es una constante. Para mostrar que este algoritmo es efectivamente de orden exponencial, vemos que

$$
\begin{aligned}
t(n) &= 2t(n-1) + \text{const} \\
&= 2\big(2t(n-2) + \text{const}\big) + \text{const} = 2^2 t(n-2) + (2+1)\text{const} \\
&= 2^2(2t(n-3) + \text{const}) + \text{const} = 2^3 t(n-3) + (2^2+1)\text{const} \\
&= 2^3(2t(n-4) + \text{const}) + \text{const} = 2^4 t(n-4) + (2^3+1)\text{const} \\
&\vdots \\
&= 2^{n-1}(2t(n-(n-2)) + \text{const}) + \text{const} = 2^n t(2) + (2^{n-1}+1)\text{const} \\
&= 2^n(2t(n-(n-1)) + \text{const}) + \text{const} = 2^{n+1} t(1) + (2^n+1)\text{const} \\
&= 2^n\big(2t(1) + \text{const}\big) + \text{const} \\
t(n) &= k_1 2^n + k_2
\end{aligned}
$$

Si definimos $t(1) = 1$, haciendo $k_1 = 2t(1) + \text{const}$ y $k_2 = \text{const}$, y considerando la definición 2.2, concluimos que $t(n) \in O(2^n)$. De esta forma podemos ver que al aumentar el

tamaño del ejemplar, el tiempo crece en forma exponencial, lo cual torna este problema difícil de resolver.

2.3 Representación de Estrategias de Algoritmos

Los científicos de la computación, han identificado diversas técnicas generales que a menudo producen algoritmos eficientes para la resolución de muchas clases de problemas. En esta parte presentamos algunas técnicas que están relacionadas con el tema central de la tesis, me refiero de la solución del problema de programación lineal entera, el cual usa técnicas de ramificación y acotamiento. La representación de las técnicas que mostraremos a continuación son: recursión y programación dinámica. (Valenzuela, 2003)

Se debe tener en cuenta que hay algunos problemas como los NP-Completos, el cual ni éstas ni otras técnicas conocidas producirán soluciones eficientes. Cuando se presenta un problema de este tipo, suele ser útil determinar si las entradas al problema tiene características especiales que se puedan explotar en la búsqueda de una solución, o si puede usarse alguna solución aproximada sencilla, en vez de la solución exacta, difícil de calcular.

2.3.1 Recursión

Una técnica fundamental de la recursión en algoritmos eficientes, está basada en la solución de versiones más pequeñas del problema, para obtener la solución general del mismo. Un ejemplar del problema se soluciona según la solución de una o más ejemplares diferentes y más pequeñas que ella.

La recursión, es una herramienta poderosa que sirve para resolver cierto tipo de problemas, consiste en que una función o procedimiento se llama a sí mismo.

La recursividad es una alternativa a la iteración o repetición, y aunque en tiempo de computadora y en ocupación en memoria es la solución recursiva menos eficiente que la solución iterativa, existen numerosas situaciones en las que la recursividad es una solución simple y natural a un problema que en caso contrario ser difícil de resolver. Por esta razón se puede decir que la recursividad es una herramienta potente y útil en la resolución de problemas que tengan naturaleza recursiva y, por ende, en la programación.

Tipos de recursividad

- **Directa o simple:** un subprograma se llama a si mismo uno o más veces directamente.
- **Indirecta o mutua:** un subprograma *A* llama a otro subprograma *B* y éste a su vez llama al subprograma *A*.

Ejemplos:

Factorial

$n!=1\times 2\times 3\ldots\times(n-2)\times(n-1)\times n$

$0!=1$

$1!=1$

$2!=1\times 2$

$3!=1\times 2\times 3$

…

$n!=1\times 2\times 3\ldots\times(n-2)\times(n-1)\times n \quad \Rightarrow n!=n\times(n-1)!$

$Si \quad n=0 \quad entonces \quad n!=1$

$Si \quad n>0 \quad entonces \quad n!=n\times(n-1)!$

```
Function factorial(n:integer):longint;
begin
    if (n=0) then
            factorial:=1
    else
            factorial:=n*factorial(n-1);
end;
```

Observación de la ejecución:

Si n=5

Nivel

1) Factorial(5)=5*facorial(4)=120
2) Factorial(4)=4*facorial(3)=24
3) Factorial(3)=3*facorial(2)=6
4) Factorial(2)=2*facorial(1)=2
5) Factorial(1)=1*facorial(0)=1
6) Factorial(0)=1

La profundidad de un procedimiento recursivo es el número de veces que se llama a sí mismo.

Serie de Fibonacci

$0,1,1,2,3,5,8,\ldots$

$F_0 = 0 \;;\; F_1 = 1$

$F_2 = F_1 + F_0 = 1$

$F_3 = F_2 + F_1 = 2$

$F_4 = F_3 + F_2 = 3$

$\ldots$

$F_n = F_{n-1} + F_{n-2}$

Si n=0,1 $F_n = n$

Si $n > 1$ $F_n = F_{n-1} + F_{n-2}$

```
Procedure fibo(var fib:longint; n:integer);
var
      fibA, fibB:integer;
begin
      if (n=0) or (n=1) then fib:=n
      else begin
          fibo(fibA,n-1);
          fibo(fibB,n-2);
```

```
        fib:=fibA+fib;
    end;
end;
```

```
Function fibo(n : integer) : integer;
begin
    if  (n = 0) or (n = 1) then
            fibo:= n
    else
            fibo:= fibo(n – 1) + fibo(n – 2);
    end;
end;
```

Cambiar la recursión es complicado cuando existe más de una llamada recursiva, pero una vez hecha ésta, la versión del algoritmo es siempre más eficiente

2.3.2 Programación Dinámica

En un problema complejo a menudo se resuelve dividiendo este problema en subproblemas más pequeños, resolver estos últimos (recurriendo posiblemente a nuevas subdivisiones) y combinar las soluciones obtenidas para calcular la solución del problema inicial.

Puede ser que en ciertos problemas la división natural conduzca a un gran número de ejemplares idénticos. Si se resuelve cada uno de ellos sin tener en cuenta las posibles repeticiones, resulta un algoritmo ineficiente; en cambio si se resuelve cada ejemplar distinto una sola vez y se conserva el resultado, el algoritmo obtenido es mucho mejor.

Esta es la idea de la programación dinámica: no calcular dos veces lo mismo y utilizar una tabla de resultados que se va rellenando a medida que se resuelven los subejemplares. (Valenzuela, 2003)

Varias estrategias HEURÍSTICAS, frecuentemente dependientes del problema, establecen las reglas para decidir en qué orden se tratan las tareas mediante las que se realiza la

ramificación del árbol de búsqueda, y qué funciones se utilizan para producir la acotación del árbol.

En optimización combinatoria, el problema de la selección óptima, trata de seleccionar una solución por sus componentes, de forma que respetando una serie de restricciones, de lugar a la solución que optimiza una función objetivo que se evalúa al construirla. El árbol de búsqueda permite recorrer todas las soluciones para quedarse con la mejor de entre las que sean factibles. Las estrategias HEURÍSTICAS orientan la exploración por las ramas más prometedoras y la acotación en aquellas que no permiten alcanzar una solución factible o mejorar la mejor solución factible alcanzada hasta el momento.

Método de Branch and Bound

Branch and bound (o también conocido como Ramificación y acotamiento), su diseño se basa en el análisis del árbol de búsqueda de soluciones a un problema. Sin embargo, no utiliza la búsqueda en profundidad, y solamente se aplica en problemas de optimización.

Los algoritmos generados por esta técnica son normalmente de orden exponencial o peor en su peor caso, pero su aplicación ante instancias muy grandes, ha demostrado ser eficiente.

Se debe decidir de qué manera se conforma el árbol de búsqueda de soluciones. Sobre el árbol, se aplica una búsqueda en anchura, pero considerando prioridades en los nodos que se visitan. El criterio de nodos, se basa en un valor óptimo posible (bound), con el que se toman decisiones para hacer las acotaciones en el árbol.

Modo de proceder de los algoritmos Branch and Bound:

- Nodo vivo: nodo con posibilidades de ser ramificado, es decir, no se ha realizado una acotación.
- No se pueden comenzar las acotaciones hasta que no se haya descendido a una hoja del árbol
- Se aplicara la acotación a los nodos con una cota peor al valor de una solución dada, entonces a los nodos acotados a los que se les ha aplicado una acotación se les conoce como nodos muertos

- Se ramificará hasta que no queden nodos vivos para expandir
- Los nodos vivos han de estar almacenados en algún sitio, en una lista de nodos, de modo que sepamos qué nodo expandir

2.4 Complejidad de un Problema

En esta sección hablaremos sobre un campo que camina paralelamente con los algoritmos, la complejidad computacional. Dentro de la evolución de la teoría de la complejidad se han encontrado problemas con características similares, los cuales pueden ser agrupados en categorías para su estudio.

La complejidad por lo general está hecha en función de problemas de reconocimiento (o decisión), un problema de reconocimiento es aquel donde se busca una respuesta del tipo "si" o "no". Los problemas más interesantes están caracterizados porque los algoritmos que los resuelven requieren tiempo de ejecución finita, y por lo tanto, pueden ser resueltos.

En este capítulo los algoritmos han sido clasificados como buenos o malos algoritmos. Un buen algoritmo es aquel para el cual existe un algoritmo polinomial determinístico que lo resuelva. También se acepta que un mal algoritmo es aquel para el cual dicho algoritmo simplemente no existe. Un problema se dice intratable, si es muy difícil que un algoritmo de tiempo no polinomial lo resuelva. No obstante esta clasificación de algoritmos en buenos y malos puede resultar a veces engañosa, ya que se podría pensar que los algoritmos exponenciales no son de utilidad práctica y que habrá que utilizar solamente algoritmos polinomiales. Sin embargo se tiene el caso de los métodos simplex y Branch and Bound, los cuales son eficientes para muchos problemas prácticos. Desafortunadamente ejemplos como estos dos son raros, de modo que es preferible seguir empleando como regla de clasificación en buenos y malos algoritmos, la que lo hace dependiendo de si son o no polinomiales.

Ahora, los algoritmos se pueden clasificar de acuerdo al tipo de cálculo que estos ejecutan: determinísticos y no-determinísticos.

Un algoritmo determinístico, se caracteriza porque en cada iteración, todos los resultados obtenidos están completamente determinados por las condiciones iniciales. Una propie-

dad importante es que para una misma entrada (ingreso de datos que construyen el ejemplar), el resultado de aplicar repetidas veces el algoritmo siempre es el mismo.

Un algoritmo no-determinístico tiene dos fases:

- Fase no-determinística (predicción): alguna cadena de caracteres, *c*, completamente arbitrarias es escrita a partir de algún lugar de memoria designado. Cada vez que el algoritmo corre, la cadena escrita puede diferir (la cadena puede ser vista como una adivinación de la solución para el problema, por lo que a esta fase se le da el nombre de fase de adivinación, pero *c* también podría ser complicado o sin sentido).

- Fase determinística (verificación): Un algoritmo determinístico (es decir ordinario) siendo ejecutado. Además de la entrada del problema de decisión, el algoritmo puede leer *c*, o puede ignorarla. Eventualmente éste para con una salida de si o no, o puede entrar en un ciclo infinito y nunca parar (también se le conoce como la fase de chequear, el algoritmo determinístico verifica *c* para ver si ésta es una solución para la entrada del problema de decisión).

El número de pasos llevados a cabo durante la ejecución de un algoritmo no-determinístico es definido como la suma de los pasos en ambas fases; esto es, el número de pasos tomados para escribir *c* (simplemente el número de caracteres en *c*) más el número de pasos ejecutados por la segunda fase determinística.

Además, se dice que un algoritmo no-determinístico es polinomialmente acotado si hay un polinomio *p* tal que para cada entrada de tamaño *n* para la cual la respuesta es si, hay alguna ejecución del algoritmo que produce una salida si en cuanto mucho *p*(*n*) pasos.

Por lo anterior, se puede decir que: NP es la clase de problemas de decisión para los cuales hay un algoritmo no-determinístico acotado polinomialmente (el nombre de NP viene de no determinístico polinomialmente acotado). (Valenzuela, 2003)

En esta parte, la complejidad de problemas está basada en la caracterización de problemas en clases, nosotros veremos las clases más importantes: clase P, clase NP y la clase NP -Completo.

Un problema en la clase NP-Completo es un problema computacional que es tan difícil de resolverlo como cualquier problema razonable, problemas que están dirigidos a la determinación de trazos y diseños óptimos de objetos tales como rutas, conjuntos de nodos, particiones, etc., todos estos conceptos sujetos a una formulación precisa. Usualmente, cuando un problema está en la clase NP-Completo, simplemente se dice que el problema es NP-Completo. Seguidamente tenemos dos interesantes propiedades que caracterizan un problema NP-Completo:

- Ningún problema NP-Completo puede ser resuelto por algún algoritmo de tiempo polinomial conocido.

- Si existe un algoritmo de tiempo polinomial para algún problema NP-Completo, entonces existen algoritmos polinomiales para todos los problemas NP-Completos.

Computacionalmente hablando, los problemas NP-Completos son poco tratables, si existe un algoritmo capaz de obtener una solución para un determinado ejemplar de un problema de este tipo, este presentará un tiempo de orden exponencial en el peor de los casos, de allí que son impracticables para todos los ejemplares. Ahora definiremos un ejemplar de un problema de optimización en forma general.

Definición 2.3. Un ejemplar de un problema de optimización es un par (G,c), donde G es cualquier conjunto (o domino de puntos factibles o soluciones factibles), c es una función de costo

$$c : G \mapsto \Re$$

El problema consiste en hallar $f \in G$ tal que $c(f) \leq c(y), \forall y \in G$, f es llamada solución global óptima.

2.4.1 Clase P

La clase P está conformada por todos aquellos problemas de reconocimiento (decisión) para los cuales se tiene un algoritmo de solución que se ejecuta en tiempo polinomial. (Papadimitriou y Kenneth, 1982)

En otras palabras, P es la clase de problemas de reconocimiento relativamente fáciles de resolver, algunos problemas de la clase P son por ejemplo: Programación Lineal, Emparejamiento Máximo, Árbol generador Mínimo (AGM), Conectividad en Grafos.

Ejemplo 2.4

El problema de Programación Lineal pertenece a la clase P. Tal merecimiento no se debe a la existencia del Algoritmo Simplex, pues éste es exponencial, si no que a la publicación del Algoritmo de los Elipsoides de L. G. Khachian, en el año 1978, el cual sí es de tiempo polinomial.

Después de 1978 fueron publicados varios algoritmos de tiempo polinomial para programación lineal, el más eficiente en la actualidad es el Algoritmo de Puntos Interiores Primal-Dual. Todos ellos corroboran que el problema de programación lineal es un problema de la clase P.

Ejemplo 2.5

Un árbol es un grafo $G = \langle N, A \rangle$ no dirigido, conexo, y acíclico. Equivalentemente, un árbol puede ser definido como un grafo no dirigido en el cual debe existir exactamente un camino entre cualquier par de nodos (vértices).

El problema de encontrar el árbol generador mínimo es definido como sigue:

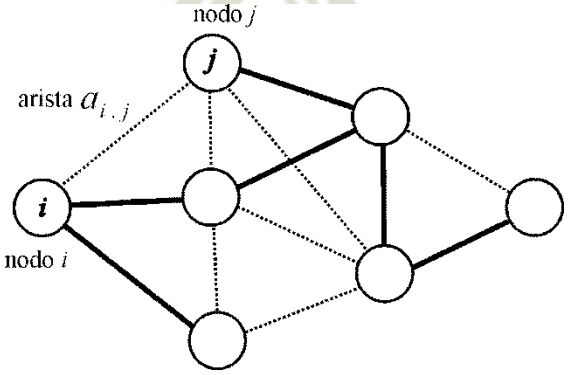


Figura 2.3. Árbol generador (líneas en negrito)

Sea $G = (N, A)$ un grafo conexo no dirigido donde cada arista tiene asignado un valor no negativo. El problema es hallar un subconjunto E de las aristas de G tal que todos los nodos de G resulten conectados, cuando sólo las aristas en E son usadas, además la suma de los valores de las aristas en E sea la menor posible.

Un algoritmo de tipo *guloso* que resuelve este problema es el algoritmo de Kruskal. Este algoritmo toma un tiempo de ejecución en $O(n^2 \log n)$ para resolver cualquier ejemplar de este problema. Lo que permite establecer este problema esté en la clase P.

2.4.2 Clase NP

Con respecto a un problema de reconocimiento, para que este problema esté en la clase *NP*, no necesitamos que todo ejemplar sea resuelto en tiempo polinomial por medio de un algoritmo, como sucedía en la clase P. Un problema de reconocimiento X está en NP, si dado un ejemplar ξ de X donde exista una solución factible, entonces deben existir un algoritmo Λ de tiempo polinomial y un *certificado conciso* c_ξ tal que:

i. $|c_\xi| \leq p(|\xi|)$, donde p es un polinomio.

ii. c_ξ debe ser verificado por Λ que realmente es una solución factible (obviamente esto es hecho en tiempo polinomial)

Donde $|\xi|$ representa el tamaño del ejemplar ξ. El certificado conciso c_ξ es una solución factible y no necesariamente debe ser conocida explícitamente, apenas se requiere garantizar su existencia.

La intención de exigir que el certificado conciso c_ξ cumpla (i), se debe al hecho que no estamos interesados en resultados óptimos los cuales no pueden ser escritos en tiempos razonables. La exigencia que cumpla (ii), se debe a que tampoco estamos interesados en obtener soluciones que no puedan ser verificadas en tiempo razonable.

Podemos decir que la clase NP está conformada por la mayoría de problemas de reconocimiento (decisión), para los cuales no se han encontrado algoritmos determinísticos que los resuelve en tiempo polinomial.

En otras palabras, son problemas en los cuales existe un algoritmo no determinístico que lo resuelve en tiempo polinomial.

Además, la clase *NP* se define sobre una Maquina de Turing no-determinística, se caracteriza por la ejecución de muchas instrucciones simples en forma simultánea. Supongamos una máquina que tiene la propiedad de dividirse en varias copies idénticas de sí misma cada vez que se ve enfrentada a una alternativa, y que las copias continúan trabajando en paralelo. Representando este proceso como un árbol de búsqueda. Si una de las copias responde afirmativamente, está resuelto el problema de decisión. Si el tiempo máximo que se requiere para recorrer una rama está acotado polinomialmente, el problema está en *NP* (López, 2008).

Los problemas *NP* son más difíciles que los de la clase *P.* Por lo tanto, la clase *NP* contiene los problemas de la clase P $\left(P \subseteq NP\right)$.

Entre los problemas *NP* existe una subclase formada por los problemas difíciles aún no resueltos eficientemente, los problemas *NP*-Completos que lo veremos posteriormente. (Papadimitriou y Kenneth, 1982)

Ejemplo 2.6

El problema de PLE está en NP, consideremos este problema en versión reconocimiento. Es posible transformar $c^{\mathrm{T}} x \leq L$ en una ecuación con la ayuda de variables de holgura para luego ser adicionadas en el sistema $Ax = b$, de este modo, la versión reconocimiento para este problema toma la siguiente forma: Dada la matriz A y el vector b, existe un vector entero x tal que $Ax = b$ y $x \geq 0$

Para verificar que este problema pertenece a la clase NP, consideremos un ejemplar que posea una solución factible para el problema de PLE, ahora, si este tiene solución factible, entonces todos sus componentes están limitados por lo siguiente

$$M = n(ma_1)^{2m+3}(1+a_2),$$

donde $a_1 = \max\{|a_{ij}|\}$ y $a_2 = \max\{|b_i|\}$, luego podemos tomar justamente esa solución factible como un certificado conciso para este ejemplar. Los detalles se pueden encontrar en la página 320 de Papadimitriou y Kenneth (1982).

Con respecto a relación entre la clase P y la clase NP, se puede decir que P es un subconjunto de NP, lo que no se sabe aún es si es un subconjunto propio o no. Lo que sí se sabe, es que por lo general, si un problema pertenece a NP, entonces debe existir un algoritmo determinístico que lo resuelve en un tiempo de ejecución exponencial. Aquí es donde reside la dificultad de los problemas que pertenecen a NP: de resolverse con algoritmos determinísticos sólo se puede garantizar un tiempo de ejecución exponencial, lo que significa que para instancias suficientemente grandes no hay esperanza de resolverlos rápidamente.

2.4.3 Reducciones para Tiempo Polinomial

Definición 2.4. Sean P_1 y P_2 problemas de reconocimiento, P_1 se reduce en tiempo polinomial a P_2, si y solamente si, existe un algoritmo de tiempo polinomial Λ_1 para P_1, el cual usa en varias llamadas (como una subrutina el costo unitario) un algoritmo Λ_2 para P_2. El algoritmo Λ_1 es denominado reducción de tiempo polinomial de P_1 a P_2.

Llamamos "costo unitario" al hecho que el algoritmo Λ_2 es considerado como una simple instrucción, tan igual como una operación elemental. No obstante Λ_2 puede ser ineficiente y el problema se complicaría. El hecho interesante sucede cuando el algoritmo Λ_2 es de tiempo polinomial, como indica el siguiente teorema.

Teorema 2.1. Si P_1 se reduce polinomialmente a P_2, y existe un algoritmo de tiempo polinomial para P_2, entonces debe existir un algoritmo polinomial para P_1.

Definición 2.5. Decimos que P_1 se transforma polinomialmente para P_2, si existe una reducción de tiempo polinomial de P_1 a P_2, con sólo una llamada del algoritmo Λ_2 para

P_2 exactamente al final del algoritmo Λ_1 para P_1; el resto del algoritmo Λ_1 es dedicado a construir el ingreso para Λ_2.

Ahora usaremos esta última definición para definir un problema NP-Completo.

2.4.4 Problema NP-Completo

Definición 2.6. Un problema de reconocimiento $R \in NP$ es llamado NP-Completo, si cualquier otro problema en NP se transforma polinomialmente a R.

Los problemas NP-Completos son problemas difíciles de resolverse dentro de la clase NP. La palabra completitud significa que la solución de un problema de decisión NP contiene, de alguna forma, la solución a todos los problemas de decisión de la clase NP.

Ahora, si existiese un algoritmo eficiente para un problema R el cual es NP-Completo, entonces por el teorema 2.1 deben existir algoritmos eficientes para todos los problemas en NP. En particular, si existiese un algoritmo polinomial para un problema NP-Completo, entonces sucederá que $P = NP$. Esto en la actualidad es un tema abierto para los investigadores dentro del área de la complejidad computacional.

Teorema 2.2 (Transitividad)**.** Sean los problemas R_1, R_2 y R_3. Si el problema R_1 se transforma polinomialmente en R_2, y R_2 se transforma polinomialmente en R_3, entonces R_1 se transforma polinomialmente en R_3.

Para probar que un determinado problema X es NP-Completo, nosotros debemos mostrar dos cosas:

1. El problema X está en NP, y

2. Que cualquier otro problema en NP se transforma polinomialmente en X.

La técnica para verificar que un problema X cumple (2) consiste en los siguiente: elegimos un problema conocido Y el cual es NP-Completo, de modo que éste se transforme

polinomialmente a X, por el teorema 2.2, cualquier otro problema se transforma polinomialmente en Y, y por consecuencia en X.

El objetivo de este capítulo es determinar que el problema *PLE* es NP-completo, el siguiente teorema garantiza lo mencionado.

Teorema 2.3. El problema de Programación Lineal Entera (PLE) es un Problema NP-Completo.

Demostración. Papadimitriou y Kenneth, (1982, p.358), demuestran el teorema 2.3.

Ejemplo 2.7: Seguidamente daremos a conocer problemas que pertenecer a los grupos de problemas *P* y *NP* – Completo. (Papadimitriou y Kenneth, 1982)

Problemas *P*: Ordenar un arreglo, Ruta más corta entre dos puntos, Calcular el determinante de una matriz, Programación Lineal.

Problemas NP – Completo: Problema del vendedor viajero, Programación Lineal Entera, El problema de Circuito Hamiltoniano, El problema Clan, El Problema Subgráfica Isomorfa, El problema 3-Colorabilidad

Ahora, mostraremos en la figura 2.4, las clases de complejidad computacional, la ubicación de las clases P, NP y NP-Completo.

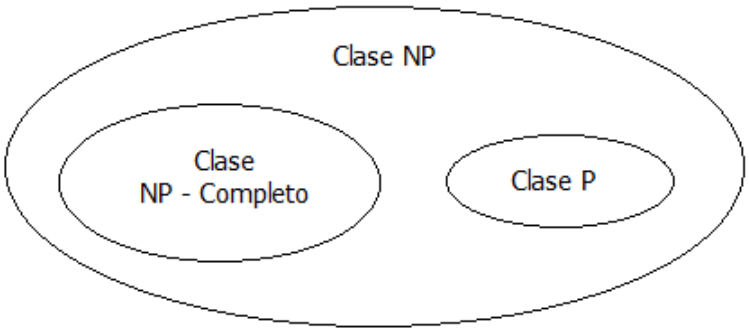


Figura 2.4. Clases de complejidad computacional

Capítulo 3

Implementación Computacional de Métodos para la Resolución de Problemas de Programación Lineal Entera.

3.1 Introducción

En este capítulo nos centraremos en los métodos llamados: "*Programación lineal entera vía el método de puntos interiores primal-dual* (*PLEPIPD*)" y "*Programación lineal entera vía el método de puntos interiores predictor-corrector* (*PLEPIPC*)*"*, el primer método resulta ser la combinación del método de Puntos Interiores Primal-Dual (para programación lineal) con el método de Ramificación y Acotamiento (R&A), y el segundo método es la fusión del método de Puntos Interiores Predictor-Corrector (para programación lineal) con el método de Ramificación y Acotamiento. Posteriormente, mostraremos el comportamiento de la fusión que tienen estos métodos, además la solución de problemas de programación lineal entera (*PLE*), ya que estos problemas pertenecen a la clase *NP*-Completos vistos en el capítulo 2.

Debemos tener en cuenta que desde el punto de vista matemático, el Problema de Programación Lineal Entera (PLE) es un caso particular del Problema de Programación Lineal (PL), la diferencia radica en el aspecto computacional, es decir, el modo de resolver estos problemas.

Por otro lado, el Problema de Programación Lineal (*PL*) está en la clase P, lo que significa que es un problema fácil de resolver, debido a la existencia de algoritmos polinomiales para su resolución. Aquí también mostraremos otra técnica de resolver problemas de programación lineal entera, utilizando algunos resultados de la teoría matemática de Programación Dinámica (*PD*).

Los problemas propios de la *PD* son aquellos que pueden ser divididos en subproblemas, los cuales, a su vez, tienen una estructura igual al problema original (en este sentido podría decirse que tiene una estructura "fractal", el cual veremos posteriormente una de ellas, la Alfombra de Sierpinski). Para esta determinación, el método consiste en dividir el problema en etapas, resolver la primera de estas, utilizar esta solución para resolver la etapa siguiente y continuar así sucesivamente hasta encontrar la solución del problema en su totalidad.

Este capítulo será dividido en dos partes. En la primera veremos un procedimiento para resolver el Problema de Programación Lineal con Variable Entera basado en la estrategia de Ramificación y Acotamiento, la cual utiliza a su vez, como una subrutina, el Algoritmo de Puntos Interiores Primal Dual para Programación Lineal (con variable continua), luego, para el mismo problema, también utilizaremos técnicas de Ramificación y Acotamiento, el cual utiliza como subrutina el Algoritmo de Puntos Interiores Predictor-Corrector para Programación Lineal (con variable continua). Además, mostraremos otra técnica de Programación Dinámica que resuelve este tipo de problema de programación lineal con variable entera. En la segunda parte presentaremos las implementaciones de los algoritmos y registraremos los resultados en los experimentos computacionales.

Por conveniencia, al Problema de Programación Lineal clásico lo denominaremos en este trabajo el Problema de Programación Lineal con variable continua.

La estrategia para resolver el problema de programación lineal entera, por lo general consiste en resolver este problema como si fuera un problema de programación lineal en variable continua, sin considerar la condición que x sea entera, este procedimiento recibe el nombre de Relajación del Problema de Programación Lineal Entera.

La técnica de Ramificación y Acotamiento aplicada a la Programación Lineal Entera mediante la utilización del bien conocido Algoritmo Simplex no es novedosa, de hecho constituye un modo clásico de abordar este problema. Lo cual en este capítulo mostraremos la modificación de esta técnica, la combinación de Ramificación y Acotamiento con el Algoritmo de Puntos Interiores Primal-Dual, también mostraremos la fusión del Algoritmo de Puntos Interiores Predictor-Corrector con la técnica de Ramificación y Acotamiento. El Problema de Programación Lineal Entera es NP-Completo, por lo tanto el algoritmo no será polinomial.

Ahora, consideremos el Problema de Programación Lineal Entera

$$\begin{aligned} &\text{Minimizar} \quad c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &\qquad Ax = b \\ &\qquad x \geq 0\ ,\ x \text{ entero} \end{aligned} \tag{3.1}$$

donde $A \in \Re^{m\times n}$, $b \in \Re^{m}$, $c \in \Re^{n}$ y $\text{rango}\,(A) = m$.

Los métodos de Puntos Interiores Primal-Dual (*PIPD*) y el método de puntos interiores Predictor-Corrector (*PIPC*), resuelven problemas de programación lineal, aquí veremos la combinación de cada uno de ellos al ser fusionados con el método de Ramificación y Acotamiento. El procedimiento en la resolución del problema de programación lineal entera utilizando cada uno de los métodos *PLEPIPD* y *PLEPIPC* son similares, presentan la misma estructura, por tal motivo a continuación veremos el procedimiento de dichos métodos.

En resumen, estos métodos que usan Ramificación y Acotamiento para resolver Programación Lineal Entera consiste en lo siguiente: Se resuelve Programación Lineal Entera como si fuese un problema de Programación Lineal con variable continua (PL) usando el algoritmo de Puntos Interiores Primal Dual (o el método de Puntos Interiores Predictor-Corrector), si el problema es infactible o tiene solución óptima ilimitada, entonces el proceso termina detectando infactibilidad o solución ilimitada, respectivamente. Si $x*$ es la solución óptima con todas sus componentes enteras, entonces el problema está resuelto. Pero si alguna componente de la solución óptima $x*$ no es entera, entonces escogemos cualquiera de las componentes no entera, digamos que sea la k-componente x_k^*, claramente se verifica $\lfloor x_k^* \rfloor < \lfloor x_k^* \rfloor + 1$. Seguidamente generamos dos subproblemas de PL con variable continua, a los cuales denominaremos $PL1$ y $PL2$, estos subproblemas se diferencian del problema original debido a que son añadidas las restricciones adicionales

$$x_k \leq \lfloor x_k^* \rfloor \quad , \quad \text{al problema } PL1$$

$$x_k \geq \lfloor x_k^* \rfloor + 1, \quad \text{al problema } PL2$$

Es decir, el problema $PL1$ corresponde a la siguiente formulación

$$\begin{aligned} &\text{Minimizar} \quad c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &\quad Ax = b \\ &\quad x_k \leq \lfloor x_k^* \rfloor \\ &\quad x \geq 0\ ,\ x \text{ entero} \end{aligned} \tag{3.2}$$

Note que el nuevo problema ahora consta de $m+1$ restricciones. Por otro lado, el problema $PL2$ corresponde a

$$\begin{aligned} &\text{Minimizar} \quad c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &\quad Ax = b \\ &\quad x_k \geq \lfloor x_k^* \rfloor + 1 \\ &\quad x \geq 0,\ x \text{ entero} \end{aligned} \tag{3.3}$$

Observe que $PL2$ tiene ahora $m+1$ restricciones. El proceso se repite tratando $PL1$ y $PL2$ tan igual como se trató PL al inicio. Si algún subproblema, digamos $PL1'$, tiene una solución entera con valor objetivo v', ella no puede ser aún considerada como la solución al problema original (3.1), sino que tenemos que comparar con las posibles soluciones enteras de todos los otros subproblemas y escoger la mejor.

Superficialmente esto último es un procedimiento muy extenso, afortunadamente podemos aplicar la técnica de acotamiento, la cual consiste en lo siguiente: si un problema, digamos $PL2''$, tiene valor objetivo v'' tal que $v'' > v'$, entonces no tiene sentido continuar la ramificación en búsqueda de mejores soluciones, esto nos permite disminuir la búsqueda de un modo inteligente evitando la enumeración completa de las posibilidades.

Vale recalcar que la estrategia que está usando es el método de Ramificación y Acotamiento, la cual usa como una subrutina el Algoritmo de Puntos Interiores Primal Dual para Programación Lineal con variable continua ($PIPD$), o también puede usar como una subrutina al Algoritmo de Puntos Interiores Predictor-Corrector ($PIPC$). Los Algoritmos $PIPD$ y $PIPC$ son los más eficiente en la actualidad, debido a que son de orden polinomial. Si en lugar de ellos se usara el Algoritmo Simplex, como se solía hacer clásicamen-

te, el procedimiento se complicaría debido a que el Simplex es un algoritmo de orden exponencial y es ineficiente para problemas especiales o de gran tamaño.

Otro de los métodos para resolver Programación Lineal Entera es el Método del Plano Cortante de Gomory, este método trabaja con soluciones básicas para construcción de los cortes, como se vio en el primer capítulo, y puede ser considerado como un derivado del algoritmo Simplex, en realidad derivado del algoritmo Dual-Simplex. Computacionalmente, se ha visto que el método de Ramificación y Acotamiento resulta ser más eficiente que el método del Plano Cortante (Taha H., 2012), por este motivo es que en este trabajo no mostraremos un método que combine el Plano Cortante con el Algoritmo de Puntos Interiores Primal-Dual.

La Programación Dinámica es otro método que resuelve Programación Lineal Entera, es recursivo, para lograr la comprensión de la técnica de la *PD*, se ha estructurado sobre ejemplos que esclarecen la versatilidad de la *PD* que lo mostraremos posteriormente.

3.2 Implementación Computacional de los Métodos *PLEPIPD* y *PLEPIPC*

En esta sección mostraremos la combinación del Método de Ramificación y Acotamiento con los métodos de Puntos Interiores Primal-Dual ($PIPD$) y la combinación del método de Puntos Interiores Predictor-Corrector (*PIPC*) con el Método de Ramificación y Acotamiento como se mencionó anteriormente, de estas fusiones tendremos los métodos para resolver problemas de Programación Lineal Entera.

Ahora, consideremos el Problema de Programación Lineal Entera (PLE)

$$\begin{aligned} &\text{Minimizar} \quad c^{\mathrm{T}}x \\ &\text{Sujeto a:} \quad Ax = b \\ &\qquad x \geq 0,\ x \text{ entero} \end{aligned} \tag{3.4}$$

donde $A \in \Re^{m\times n}$, $b \in \Re^{m}$, $c \in \Re^{n}$ y $\text{rango}(A) = m$.

Paralelamente a este problema, consideremos el Problema de Programación Lineal con Variable Continua (PL):

$$\begin{aligned} &\text{Minimizar} \quad c^{\mathrm{T}} x \\ &\text{Sujeto a:} \\ &\qquad Ax = b \\ &\qquad x \geq 0 \end{aligned} \tag{3.5}$$

Si asumimos inicialmente que existe una solución óptima $x*$ para (PL), un resultado básico de programación lineal nos dice que tal punto puede ser un extremo, pero se debe aclarar que no estamos interesados necesariamente en soluciones óptimas que sean extremos.

La región de factibilidad asociada al problema (PL) así como el punto $x*$ son mostrados en la figura 3.1, donde x_k^* es la k-componente del vector $x*$.

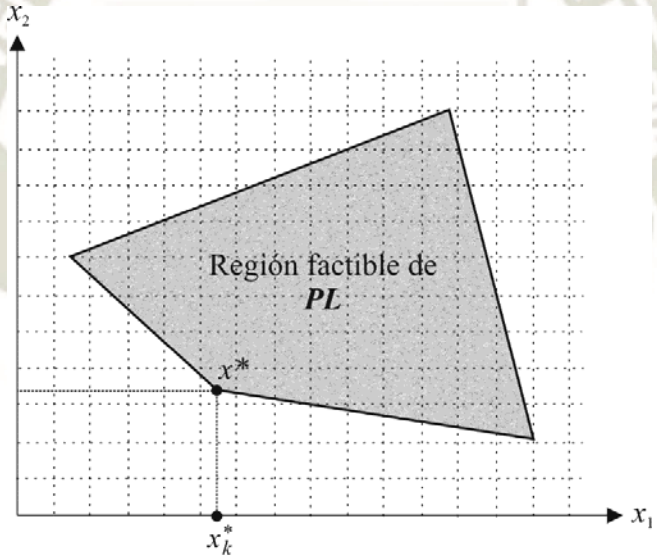


Figura 3.1. Región factible del problema de relajación de PLE

Volviendo al problema (PLE), la región de factibilidad para este problema no es un poliedro, como lo era en programación lineal, si no son los vectores, cuyas componentes son enteras, los cuales pertenecen al poliedro de factibilidad definido por (PL), a esos vectores simplemente los denominaremos soluciones factibles enteras. Los puntos blancos mostrados en la figura 3.2 representan la región de factibilidad para PLE.

El método consiste en lo siguiente: Resolver PL utilizando el algoritmo $PIPD$ (o el algoritmo $PIPC$) y obtener una solución óptima $x*$ (figura 3.1). Si $x*$ es además una

solución factible entera, entonces el problema está resuelto, y $x*$ constituye una solución óptima para el problema PLE. Por otro lado, si se detecta que el problema PL es ilimitado o infactible, entonces el problema PLE también lo será, y esa es la respuesta.

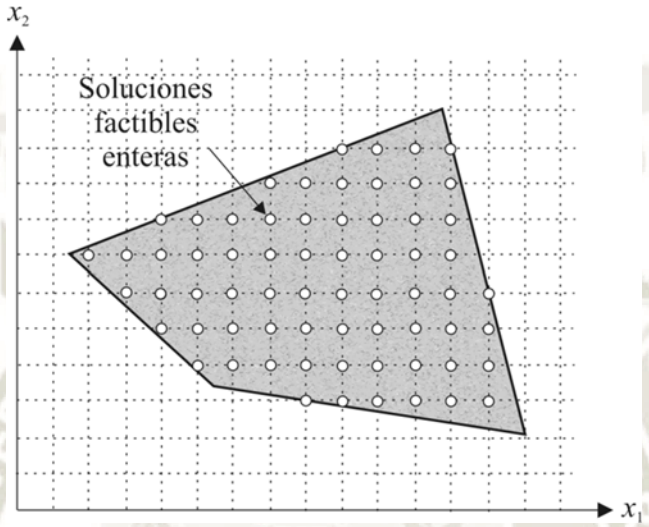


Figura 3.2. Soluciones factibles enteras limitadas por el poliedro de programación lineal

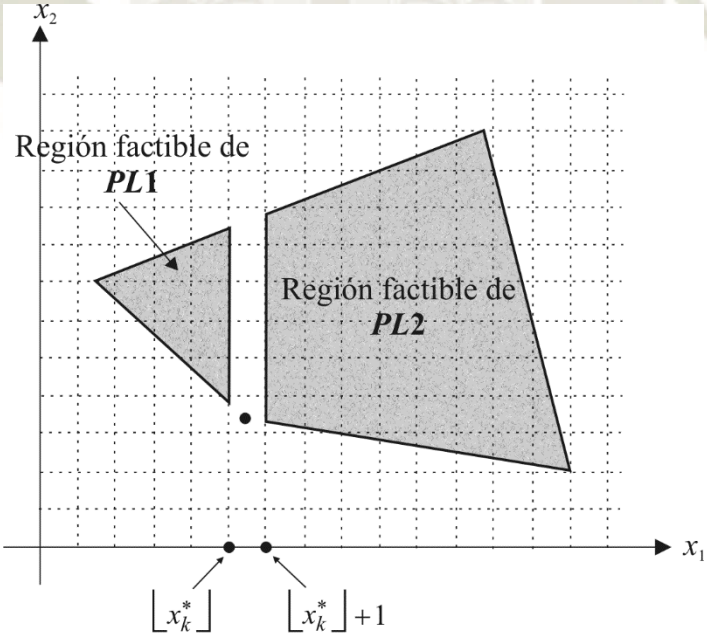


Figura 3.3. Modificación de la región original en dos nuevas regiones

Pero si existe al menos una componente no entera, digamos x_k^*, entonces ningún y, tal que $\lfloor x_k^* \rfloor < y < \lfloor x_k^* \rfloor + 1$, hará parte de una solución entera, pudiendo deshacernos de la región definida por puntos cuya *k*-componente se encuentre entre $\lfloor x_k^* \rfloor$ y $\lfloor x_k^* \rfloor + 1$, esto se muestra en la figura 3.3. El modo de deshacerse de tal región es mediante la introducción de nuevas restricciones en el problema original PLE, estas son justamente

$$x_k \leq \lfloor x_k^* \rfloor \quad \text{y} \quad x_k \geq \lfloor x_k^* \rfloor + 1$$

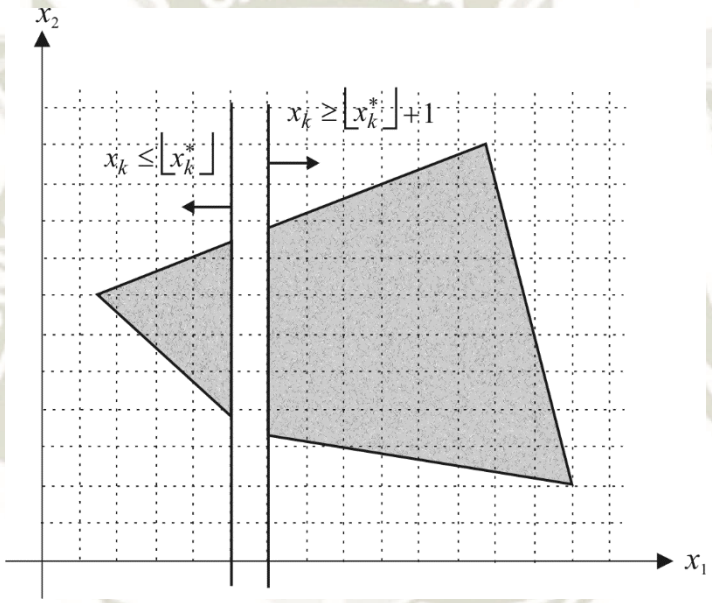


Figura 3.4

La introducción de esas nuevas restricciones determina ahora dos regiones factibles disjuntas, como ilustra la figura 3.4.

La solución entera óptima debemos buscarla ahora sobre estas dos nuevas regiones, eso es equivalente a resolver dos Problemas de Programación Lineal Entera. El primer problema lo definimos por $PL1$, y consiste en

$$\begin{aligned} &\text{Minimizar} \quad c^{\mathrm{T}} x \\ &\text{Sujeto a:} \end{aligned} \tag{3.6}$$

$$
\begin{aligned}
&Ax = b \\
&x_k \leq \lfloor x_k^* \rfloor \\
&x \geq 0 \ , \ x \ \text{entero}
\end{aligned}
$$

Mientras que el segundo lo definimos por $PL2$, y consiste en

$$
\begin{aligned}
&\text{Minimizar} \quad c^{\mathrm{T}} x \\
&\text{Sujeto a:} \\
&\qquad Ax = b \\
&\qquad x_k \geq \lfloor x_k^* \rfloor + 1 \\
&\qquad x \geq 0 \ , \ x \ \text{entero}
\end{aligned}
\tag{3.7}
$$

Ahora repetimos el procedimiento aplicado a los nuevos Problemas de Programación Lineal entera (3.6) y (3.7). Al finalizar el análisis de todas las alternativas, resta apenas escoger la mejor solución entera si existiera, ella constituirá la solución del problema original PLE, sino existe tal solución, significa que el problema PLE es infactible. Todo eso constituye la *fase de ramificación*, aparentemente se va a requerir resolver problemas del tipo PL un gran número de veces. Afortunadamente, por lo general la *fase de acotación* permite en ocasiones evitar la enumeración total de las alternativas.

Fase de Acotación: Durante la fase de ramificación, supongamos que al resolver algún problema**,** digamos $PL1'$, obtuvimos una solución entera con valor objetivo v', y si al resolver otro problema, digamos $PL2''$, obtuvimos una solución óptima, no necesariamente entera, con valor objetivo v'', y si además $v' \leq v''$. Entonces no es necesario seguir ramificando el problema $PL2''$, debido a que no existe mejor solución óptima entera en $PL2''$, o en cualquier subproblema de este, que la encontrada en $PL1'$. En estas circunstancias decimos que el subproblema $PL1$ está acotado. También consideraremos como acotado un subproblema infactible o con solución óptima ilimitada.

Un esquema del método, usando un árbol, puede ser visto en la figura 3.5

Note que en el nivel 0 se requiere aplicar una sola vez el algoritmo $PIPD$ (o aplicar una sola vez el algoritmo *PIPC*), en el nivel 1 se requiere aplicar $PIPD$ dos veces (o aplicar dos veces *PIPC*), en el nivel 2 cuatro veces, en el nivel 3 a lo mucho ochos veces, y así

sucesivamente, naturalmente en el nivel H se requiere aplicar el algoritmo $PIPD$ (o aplicar el algoritmo *PIPC*) a lo máximo 2^H veces.

Asumiendo que existe una solución factible entera para el problema PLE, entonces el proceso es finito. Luego es posible establecer una cota superior para el número total de veces que se requiere resolver subproblemas de programación lineal, está límite superior corresponde a

$$\sum_{i=0}^{H} 2^i$$

donde H es la profundidad (nivel máximo) del árbol de ramificación. Generalmente $H >> n$, lo que implica que nuestro algoritmo requiere un número exponencial de iteraciones para resolver PLE. En realidad se esperaba este resultado puesto que PLE es un problema *NP*-Completo. Esto nos permite establecer la siguiente proposición.

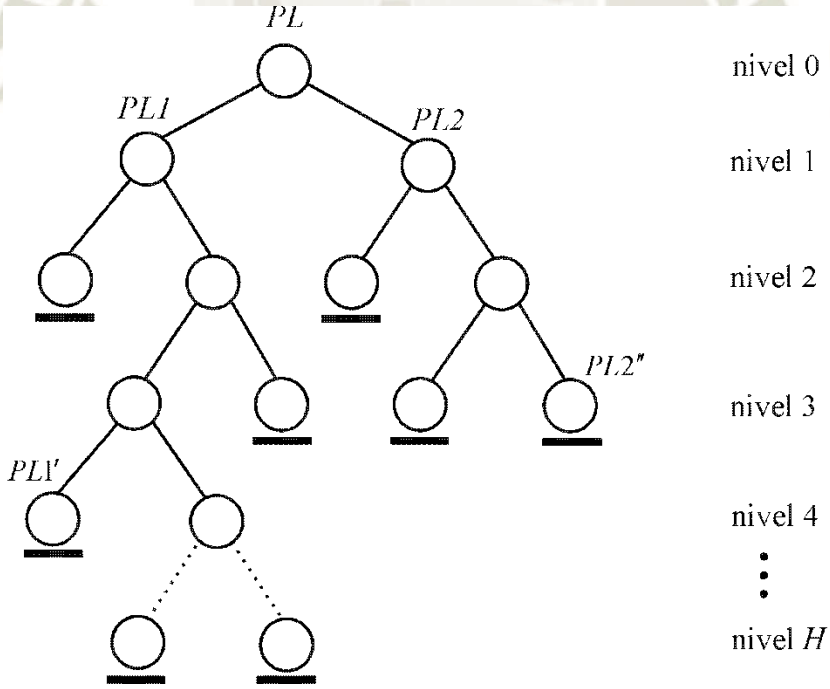


Figura 3.5. Árbol de ramificación donde interviene las tres fases, relajación, ramificación y acotación

Proposición 3.1 Si existe una solución óptima para el problema de programación lineal entera, entonces el método para resolver el problema de programación lineal entera vía el

algoritmo de puntos interiores primal-dual (o el método para resolver el problema de programación lineal entera vía el algoritmo de puntos interiores predictor-corrector) termina después de un número finito de iteraciones, en el peor de los casos este número está acotado superiormente por una cantidad exponencial en función del tamaño del problema.

Ejemplo 3.1

Considere el siguiente problema

$$\text{Minimice} \quad z = -x_1 - x_2$$

Sujeto a:

$$-4x_1 + 15x_2 \leq 30$$
$$30x_1 - 7x_2 \leq 60$$
$$x_1, x_2 \geq 0 \text{ enteros}$$

La figura 3.6 muestra la región de factibilidad (parte sombreada) de la relajación de *PLE*
.

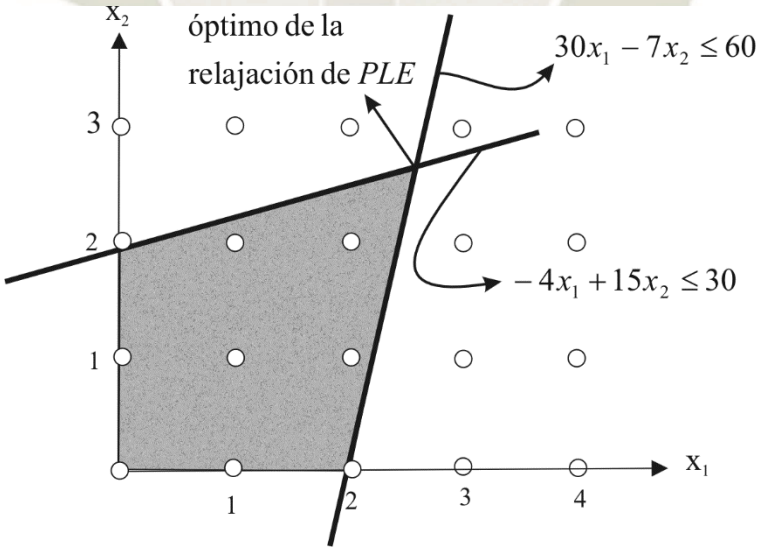


Figura 3.6.

Para colocar el problema en la forma estándar ingresamos las variables de holgura x_3, x_4, y obtenemos

$$\text{Minimice} \quad z = -x_1 - x_2 + 0x_3 + 0x_4$$

Sujeto a:

$$\begin{aligned} -4x_1 + 15x_2 + x_3 \quad &= 30 \\ 30x_1 - 7x_2 \quad + x_4 &= 60 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

$$x_1, x_2 \quad \text{enteros}$$

Ahora resolvemos la relajación de PLE usando el algoritmo $PIPD$ (o el algoritmo $PIPC$), es decir, lo resolvemos sin considerar que las variables x_1, x_2 sean enteras, de donde obtenemos la solución óptima x^*:

$$x^* = \begin{bmatrix} 2.6303 \\ 2.7014 \\ 0.0008 \\ 0.0015 \end{bmatrix}$$

y su valor objetivo óptimo está dado por $z = c^T x = -5.3316$.

Como la solución óptima del problema de relajación de PLE no es entera (sólo consideramos las dos primeras variables a ser enteras), entonces modificamos la región factible seleccionando una de las variables originales cuyo valor no sea entero, digamos $x_1 = 2.6303$, de tal modo que la franja definida por $\lfloor x_1^* \rfloor < x_1 < \lfloor x_1^* \rfloor + 1$, es decir $2 < x_1 < 3$, sea eliminada.

Luego, la relajación de PLE se subdivide en dos subproblemas $PL1$ y $PL2$ mediante la adición de nuevas restricciones: $x_1 \leq 2$ a $PL1$ y $x_1 \geq 3$ a $PL2$. De este modo la región es subdividida en dos subconjuntos S_1 y S_2 (disjuntos) como se muestra a continuación

$$\text{Subconjunto } S_1 = \text{región factible de } PL \cap \left\{ x \in \Re^2 : \ x_1 \leq 2 \right\}$$
$$\text{Subconjunto } S_2 = \text{región factible de } PL \cap \left\{ x \in \Re^2 : \ x_1 \geq 3 \right\}$$

En la figura 3.7 se aprecia la modificación del conjunto de solución de PL con sus correspondientes conjuntos S_1 y S_2.

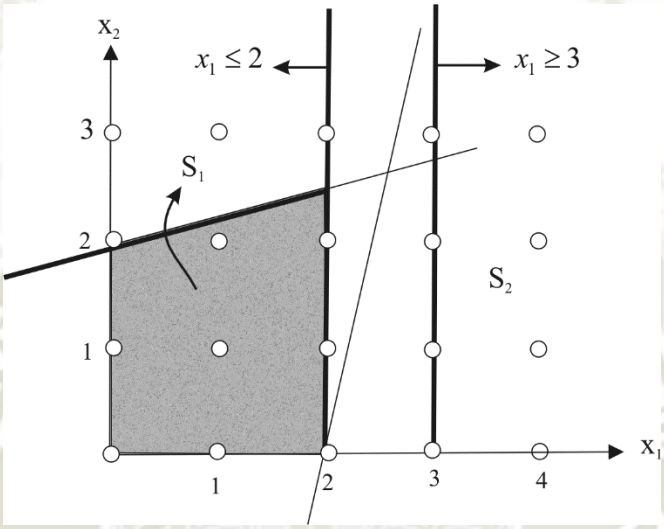


Figura 3.7.

Subproblema $PL1$

$$\begin{aligned} &\text{Minimice } z = c^{\mathrm{T}} x \\ &\text{Sujeto a:} \\ &\qquad Ax \leq b \\ &\qquad x_1 \leq 2 \\ &\qquad x_1, x_2 \geq 0 \\ &\qquad x_1, x_2, \text{ enteros} \end{aligned} \qquad (PL1)$$

Subproblema $PL2$

$$\begin{aligned} &\text{Minimice } z = c^{\mathrm{T}} x \\ &\text{Sujeto a:} \end{aligned} \qquad (PL2)$$

$$Ax \le b$$
$$x_1 \ge 3$$
$$x_1, x_2 \ge 0$$
$$x_1, x_2, \text{ enteros}$$

En ambos subproblemas permanece $A = \begin{bmatrix} -4 & 15 \\ 30 & -7 \end{bmatrix}$, $b = [30 \quad 60]^T$ y $c = [-1 \quad -1]^T$.

Ahora examinamos el subproblema $PL1$, resolviéndolo como si fuese un problema de programación lineal usando el algoritmo *PIPD* (o el algoritmo *PIPC*), las variables originales correspondientes a la solución óptima de $PL1$ son: $x_1 = 2.0010$, $x_2 = 2.5335$; y el valor óptimo es $z = -4.5345$. La figura 3.8 muestra esta situación con respecto al espacio asociado a las dos primeras variables.

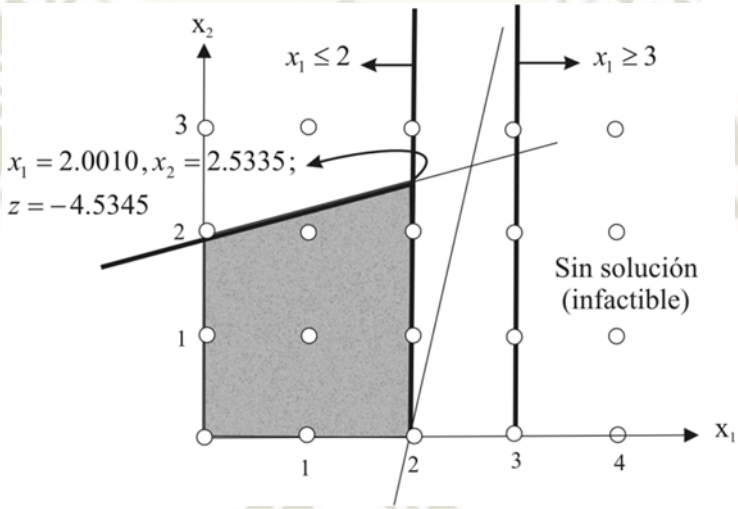


Figura 3.8.

Ahora describiremos por medio de un esquema al problema PL y a los subproblemas $PL1$ y $PL2$, que está constituida por una red con estructura de árbol, donde a cada nodo se le asocia lo siguiente: el número (de la estructura lineal correspondiente) de cada subproblema, el valor de las variables y su correspondiente valor óptimo para esa estructura.

Como la solución del subproblema $PL1$ no es entera, entonces escogemos a $x_2 = 2.5335$ y subdividiendo al subproblema $PL1$ en dos nuevos subproblemas $PL3$ y $PL4$. Esto se

consigue adicionando las nuevas restricciones al subproblema $PL1$, de tal modo que $x_2 \leq 2$ corresponde a $PL3$ y $x_2 \geq 3$ corresponde a $PL4$.

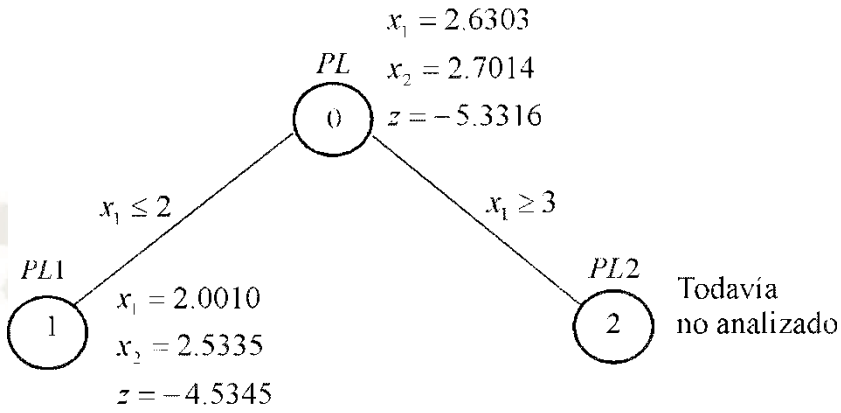


Figura 3.9.

Los correspondientes subproblemas son los siguientes

Subproblema $PL3$

$$\begin{aligned} &\text{Minimice } z = c^{\mathrm{T}}x \\ &\text{Sujeto a:} \\ &Ax \leq b \\ &x_1 \leq 2 \\ &x_2 \leq 2 \\ &x_1, x_2 \geq 0 \\ &x_1, x_2, \text{ enteros} \end{aligned} \qquad (PL3)$$

Subproblema $PL4$

$$\begin{aligned} &\text{Minimice } z = c^{\mathrm{T}}x \\ &\text{Sujeto a:} \end{aligned} \qquad (PL4)$$

$$Ax \le b$$
$$x_1 \le 2$$
$$x_2 \ge 3$$
$$x_1, x_2 \ge 0$$
$$x_1, x_2, \text{ enteros}$$

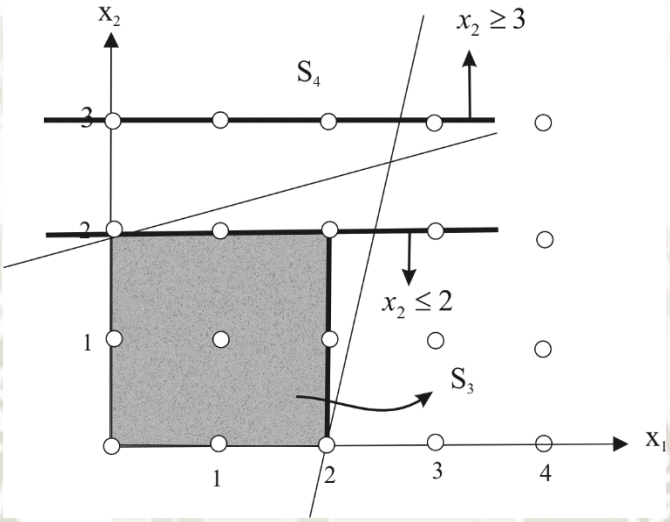


Figura 3.10.

Como anteriormente escogimos la variable x_2 de la solución del subproblema $PL1$, entonces la franja definida por $2 < x_2 < 3$ será eliminada, porque no contiene valores enteros de x_2, originando dos nuevos subconjuntos S_3 y S_4. En la figura 3.10 son mostrados los subconjuntos S_3 y S_4, los cuales están definidos por.

$$\text{Subconjunto } S_3 = s\text{subconjunto} S_1 \cap \left\{ x \in \Re^2 : \ x_2 \le 2 \right\}$$
$$= \text{región factible} PL \cap \left\{ x \in \Re^2 : x_1 \le 2 \right\} \cap \left\{ x \in \Re^2 : x_2 \le 2 \right\}$$

$$\text{Subconjunto } S_4 = s\text{subconjunto} S_1 \cap \left\{ x \in \Re^2 : x_2 \ge 3 \right\}$$
$$= \text{región factible} PL \cap \left\{ x \in \Re^2 : x_1 \le 2 \right\} \cap \left\{ x \in \Re^2 : x_2 \ge 3 \right\}$$

Ahora, pasamos a analizar el subproblema $PL3$ usando el Algoritmo *PIPD* (o el algoritmo *PIPC*) , él nos otorga la solución: $x_1 = 2, x_2 = 2; z = -4$, como esta solución es entera entonces el subproblema $PL3$ queda acotado, esta solución es por ahora una candidata potencial al problema PLE original.

Ahora falta observar los subproblemas que no fueron analizados, note que la solución del subproblema $PL3$ todavía no es la óptima, porque los subproblemas $PL4$ y $PL2$ pueden producir una solución entera mejor. Lo único que podemos decir es que $z = -4.0020$ es el límite superior para los posibles valores óptimos de PLE original.

Entonces, si existiese una solución entera óptima mejor que la del subproblema $PL3$, este límite superior es actualizado. Debemos tener presente que cualquier subproblema no examinado, que no pueda producir un valor objetivo mejor, se debe descartar pues es considerado como no promisorio.

Cuando examinamos el subproblema $PL4$ mediante el algoritmo $PIPD$ (o el algoritmo $PIPC$), vemos que la solución es infactible y el subproblema $PL4$ es acotado. Por último examinamos el subproblema $PL2$, su solución también es infactible de tal manera que el problema es acotado.

Por lo tanto, la solución óptima entera del problema PLE fue obtenida resolviendo el subproblema $PL3$, donde $x_1 = 2, x_2 = 2; z = -4$.

En la siguiente figura 3.11 mostramos la solución entera del problema PLE

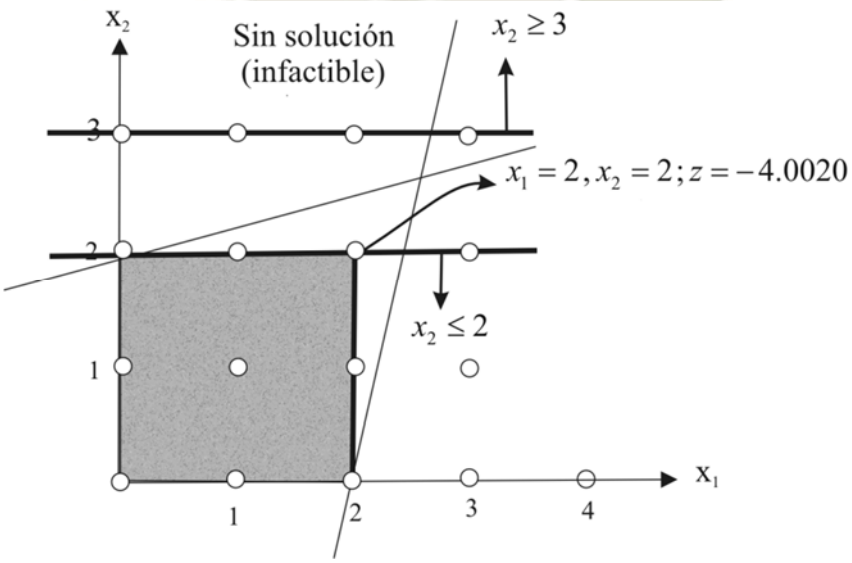


Figura 3.11.

Su correspondiente esquema es el siguiente

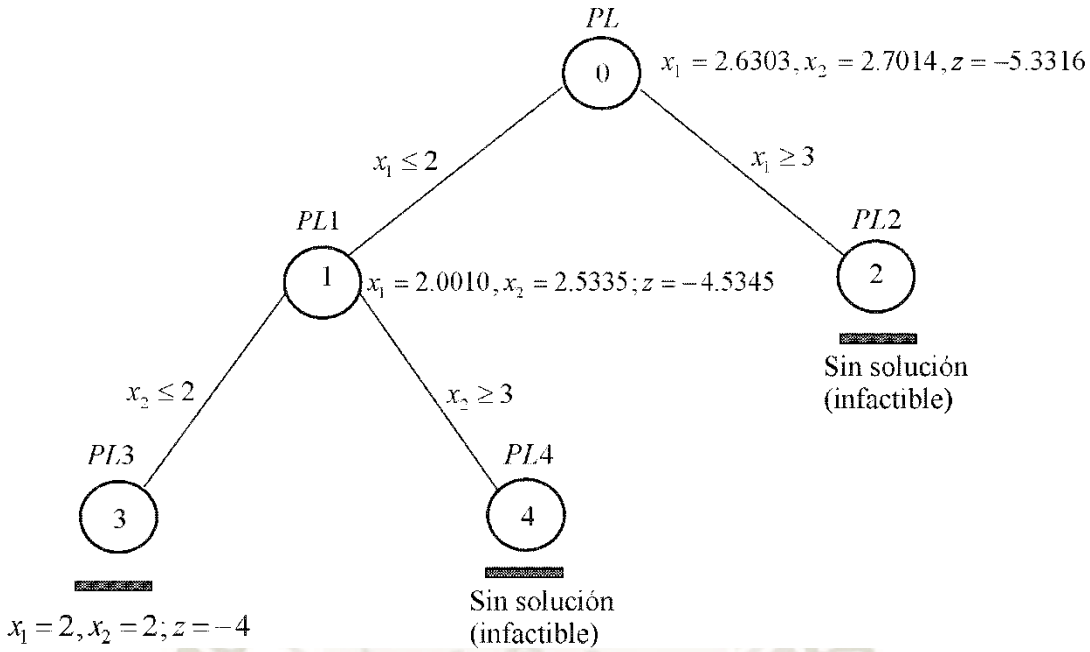


Figura 3.12. Árbol de ramificación del ejemplo 3.1

3.3 El Algoritmo

El método propuesto en este trabajo, el cual resuelve el Problema de Programación Lineal Entera usando la técnica de Ramificación y Acotamiento combinada con el Método de Puntos Interiores Primal-Dual, en el algoritmo principal es llamado *PLEPIPD*.

El otro método que también propusimos en esta tesis y resuelve el problema de Programación Lineal Entera, usa el método de Puntos Interiores Predictor-Corrector fusionado con el método de Ramificación y Acotamiento, en el algoritmo principal lo llamamos *PLEPIPC.*

El proceso a seguir en la resolución de problemas de Programación Lineal Entera mediante el método de ramificación y acotamiento fusionado con el método de *PIPD* (o el método *PIPC*) se resume en el siguiente esquema algorítmico:

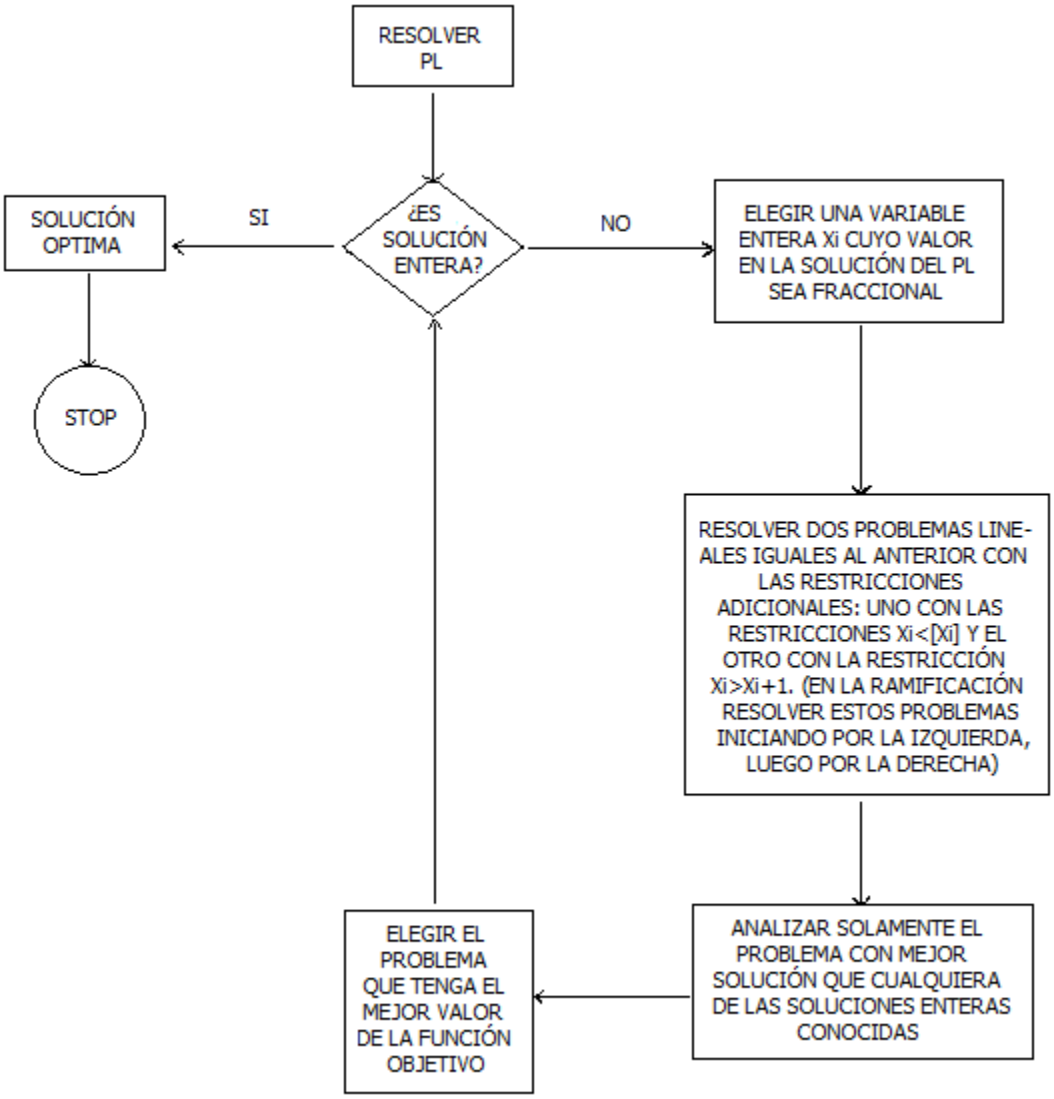


Figura 3.13: Esquema del Algoritmo PLEPIPD_(PLEPIPC)

Seguidamente mostramos los algoritmos respectivos en pseudo-código.

3.3.1 Algoritmo Principal *PLEPIPD*

El siguiente algoritmo denominado *PLEPIPD*, es de naturaleza recursiva y sintetiza el método.

Procedimiento $[fx, valor, iter, \text{cota}] \leftarrow \text{PLEPIPD}(A, b, c, nv, iter, \text{cota})$
- $[\text{x}, \text{v}] \leftarrow \text{PRIMALDUAL}(A, b, c)$
- $[x, ques, k, xi, xd] \leftarrow \text{TESTINT}(x, nv)$
- if $v = \infty$ or $v > \text{cota}$ then
  - $valor \leftarrow \infty$
  - $fx \leftarrow x$
- elseif $ques = 'si'$ and $v < \text{cota}$ then
  - $valor \leftarrow v$
  - $fx \leftarrow x$
  - $\text{cota} \leftarrow v$
- else
  - $[Ai, bi, ci] \leftarrow \text{CONSTRUYEI}(xi, k, A, b, c)$
  - $[Ad, bd, cd] \leftarrow \text{CONSTRUYED}(xd, k, A, b, c)$
  - $[xi, valori, iter, \text{cota}] \leftarrow \text{PLEPIPD}(Ai, bi, ci, nv, iter + 1, \text{cota})$
  - $[xd, valord, iter, \text{cota}] \leftarrow \text{PLEPIPD}(Ad, bd, cd, nv, iter + 1, \text{cota})$
  - if $valori < valord$ then
    - $valor \leftarrow valori$
    - $fx \leftarrow xi$
  - else
    - $valor \leftarrow valord$
    - $fx \leftarrow xd$
  - end
- end

3.3.2 Algoritmo Principal *PLEPIPC*

El siguiente algoritmo denominado *PLEPIPC*, es también de naturaleza recursiva.

Procedimiento $[fx, valor, iter, \text{cota}] \leftarrow \text{PLEPIPC}(A, b, c, nv, iter, \text{cota})$
    $[\text{x}, \text{v}] \leftarrow \text{PIPC}(A, b, c)$
    $[x, ques, k, xi, xd] \leftarrow \text{TESTINT}(x, nv)$
    if $v = \infty$ or $v > \text{cota}$ then
        $valor \leftarrow \infty$
        $fx \leftarrow x$
    elseif $ques = 'si'$ and $v < \text{cota}$ then
        $valor \leftarrow v$
        $fx \leftarrow x$
        $\text{cota} \leftarrow v$
    else
        $[Ai, bi, ci] \leftarrow \text{CONSTRUYEI}(xi, k, A, b, c)$
        $[Ad, bd, cd] \leftarrow \text{CONSTRUYED}(xd, k, A, b, c)$
        $[xi, valori, iter, \text{cota}] \leftarrow \text{PLEPIPC}(Ai, bi, ci, nv, iter + 1, \text{cota})$
        $[xd, valord, iter, \text{cota}] \leftarrow \text{PLEPIPC}(Ad, bd, cd, nv, iter + 1, \text{cota})$
        if $valori < valord$ then
            $valor \leftarrow valori$
            $fx \leftarrow xi$
        else
            $valor \leftarrow valord$
            $fx \leftarrow xd$
        end
    end

3.3.3 El Algoritmo de Puntos Interiores Primal-Dual

El siguiente algoritmo resume el método de puntos interiores primal-dual para programación lineal continua. Es decir, resuelve, en nuestro caso, el problema de relajación del problema PLE .

Procedimiento $[x, v] \leftarrow \text{PRIMALDUAL}(A, b, c)$

$x \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; $z \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; $e \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; (n - dimencionales)

$y \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; (m - dimencional)

$gap \leftarrow 10^{14}$;

while $\left| \dfrac{gap}{1 + |b^{\mathrm{T}} y|} \right| > 0.0001$ then

$X \leftarrow \text{diag}(x)$; $Z \leftarrow \text{diag}(z)$; $u \leftarrow |gap| / n^2$;

$dP \leftarrow b - Ax$;

$dD \leftarrow A^{\mathrm{T}} y + z - c$;

$matriz \leftarrow (AZ^{-1} XA^{\mathrm{T}})$;

if $rcond(matriz) < 10^{-14}$ then

$v \leftarrow \infty$; $break$;

else

$dy \leftarrow (b - uAZ^{-1} e - AZ^{-1} XdD)$;

end

$dz \leftarrow -dD - A^{\mathrm{T}} dy$;

$Zdx \leftarrow ue - XZe - Xdz$;

$\alpha_{\mathrm{P}} \leftarrow 0.98 \min_j \left\{ \dfrac{-x_j}{dx_j} \mid dx_j < 0 \right\}$;

$\alpha_D \leftarrow 0.98 \min \left\{ \dfrac{-z_j}{dz_j} \mid dz_j < 0 \right\}$;

$x \leftarrow x + \alpha_{\mathrm{P}} dx$; $y \leftarrow y + \alpha_D dy$; $z \leftarrow z + \alpha_D dz$;

$v \leftarrow c^{\mathrm{T}} x$; $gap \leftarrow v - y^{\mathrm{T}} b$;

end

3.3.4 El Algoritmo de Puntos Interiores Predictor-Corrector

El siguiente algoritmo resume al método de puntos interiores predictor-corrector para programación lineal continua, que también se aplicara en el problema de relajación de *PLE*.

Procedimiento $[x, v] \leftarrow \text{PIPC}(A, b, c)$

$x \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; $z \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; $e \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; (n - dimencionales)

$y \leftarrow [1 \quad \dots \quad 1]^{\mathrm{T}}$; (m - dimencional)

$eps \leftarrow 10^{7}$;

while $(x^{T} z) > eps$ then

$X \leftarrow \text{diag}(x)$; $Z \leftarrow \text{diag}(z)$; $dP \leftarrow b - Ax$; $dD \leftarrow c - A^{T} y - z$; $matriz \leftarrow AZ^{-1} XA^{T}$;

if $rcond(matriz) < 10^{-14}$ *then*

$v = \text{inf}$; *break*

else

$(matriz)dya \leftarrow \left(b + AZ^{-1} X(dD)\right)$;

end

$dza \leftarrow (dD - A^{T} dya)$;

if $rcond(Z) < 10^{-14}$ *then*

break;

else

$Z\, dxa \leftarrow (-XZe - Xdza)$;

end

$$\alpha_{\mathrm{P}} \leftarrow 0.98 \min_{j} \left\{ \frac{-x_j}{dxa_j} \mid dxa_j < 0 \right\}; \qquad \alpha_D \leftarrow 0.98 \min \left\{ \frac{-z_j}{dza_j} \mid dza_j < 0 \right\};$$

$x \leftarrow x + \alpha_{\mathrm{P}} dxa; \quad y \leftarrow y + \alpha_D dya; \quad z \leftarrow z + \alpha_D dza;$

$$u \leftarrow \left(\frac{(x + \alpha_p dxa)^T (z + \alpha_D dza)}{x^T z} \right)^2 \left(\frac{(x + \alpha_p dxa)^T (z + \alpha_D dza)}{n} \right)$$

$DX \leftarrow diag(dxa)$; $DZ \leftarrow diag(dza)$; $dP \leftarrow b - Ax$; $dD \leftarrow c - A^{T} y - z$;

$\left(AZ^{-1} XA^{T}\right) dyc \leftarrow \left(b - uAZ^{-1} e + AZ^{-1} (DX)(DZ)e + AZ^{-1} XdD\right)$;

$dzc \leftarrow \left(dD - A^{T} dyc\right)$; $Z\, dxc \leftarrow \left(ue - XZe - (DX)(DZ)e - Xdzc\right)$;

$$\alpha_{\mathrm{P}} \leftarrow 0.98 \min_{j} \left\{ \frac{-x_j}{dxc_j} \mid dxc_j < 0 \right\}; \qquad \alpha_D \leftarrow 0.98 \min \left\{ \frac{-z_j}{dzc_j} \mid dzc_j < 0 \right\};$$

$x \leftarrow x + \alpha_{\mathrm{P}} dxc; \quad y \leftarrow y + \alpha_D dyc; \quad z \leftarrow z + \alpha_D dzc; \quad v \leftarrow c^{T} x;$

end

3.3.5 El Algoritmo de Redondeo TESTINT

El siguiente algoritmo resume la técnica de redondeo expuesta en la construcción del método de *PLEPIPD* y *PLEPIPC*.

Procedimiento $[x, ques, k, xi, xd] \leftarrow \text{TESTINT}(x, nv)$

$aux \leftarrow x - round(x);$

if $\max\{|aux(i)|, 1 \le i \le nv\} < 0.001$ then

$x(i) \leftarrow round(x(i)), \quad 1 \le i \le nv;$

$ques \leftarrow si;\ k \leftarrow 0;\ xi \leftarrow 0;\ xd \leftarrow 0;$

else

$ques \leftarrow no;$

$[m, k] \leftarrow \max\{|aux(i)|, 1 \le i \le \text{nv}\};$

$xi \leftarrow \lfloor x_k \rfloor;$

$xd \leftarrow \lfloor x_k \rfloor + 1;$

end

3.3.6 El Algoritmo CONSTRUYEI

Este algoritmo construye el subproblema correspondiente a la ramificación izquierda de los algoritmos *PLEPIPD* y *PLEPIPC*.

Procedimiento $[Ai, bi, ci] \leftarrow \text{CONSTRUYEI}(xi, k, A, b, c)$

$$Ai \leftarrow \begin{bmatrix} & A & & \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \\ \underbrace{[0 \cdots 1 \cdots 0]}_{k-\text{posición}} & & 1 \end{bmatrix}$$

$$bi \leftarrow \begin{bmatrix} b \\ \vdots \\ xi + 0.001 \end{bmatrix}$$

$$ci \leftarrow \begin{bmatrix} c \\ \vdots \\ 0 \end{bmatrix}$$

3.3.7 El Algoritmo CONSTRUYED

Este algoritmo construye el subproblema correspondiente a la ramificación derecha de los algoritmos *PLEPIPD* y *PLEPIPC*.

Procedimiento $[Ad, bd, cd] \leftarrow \text{CONSTRUYED}(xd, k, A, b, c)$

$$Ad \leftarrow \begin{bmatrix} A & \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \\ \underbrace{[0 \cdots 1 \cdots 0]}_{k-\text{posición}} & -1 \end{bmatrix}$$

.

$$bd \leftarrow \begin{bmatrix} b \\ \vdots \\ xd - 0.001 \end{bmatrix}$$

$$cd \leftarrow \begin{bmatrix} c \\ 0 \end{bmatrix}$$

3.4 Implementación en MatLab

Aquí mostraremos las implementaciones computacionales de los algoritmos de la sección 3.3.

3.4.1 El Programa Principal *PLEPIPD*

```
function [fx,valor,iter,cota]=PLEPIPD(A,b,c,nv,iter,cota)
[x,v]=PRIMALDUAL(A,b,c);
[x,ques,k,xi,xd]=TESTINT(x,nv);
if v==inf | cota<=v
    valor=inf; fx=x;
elseif ques=='si' & v<cota
    valor=v; fx=x; cota=v;
else
    [Ai,bi,ci]=CONSTRUYEI(xi,k,A,b,c);
    [Ad,bd,cd]=CONSTRUYED(xd,k,A,b,c);
    [xi,valori,iter,cota]=PLEPIPD(Ai,bi,ci,nv,iter+1,cota);
    [xd,valord,iter,cota]=PLEPIPD(Ad,bd,cd,nv,iter+1,cota);
    if valori<valord
        valor=valori; fx=xi;
    else
        valor=valord; fx=xd;
    end
end
```

3.4.2 El Programa Principal *PLEPIPC*

```
function [fx,valor,iter,cota]=PLEPIPC(A,b,c,nv,iter,cota)
[x,v,iter]=PIPC(A,b,c);
[x,ques,k,xi,xd]=TESTINT(x,nv);
if v==inf | cota<=v
    valor=inf; fx=x;
elseif ques=='si' & v<cota
    valor=v; fx=x; cota=v;
else
    [Ai,bi,ci]=CONSTRUYEI(xi,k,A,b,c);
    [Ad,bd,cd]=CONSTRUYED(xd,k,A,b,c);
    [xi,valori,iter,cota]=PLEPIPC(Ai,bi,ci,nv,iter+1,cota);
    [xd,valord,iter,cota]=PLEPIPC(Ad,bd,cd,nv,iter+1,cota);
    if valori<valord
        valor=valori; fx=xi;
    else
        valor=valord; fx=xd;
    end
end
```

3.4.3 El Programa PRIMALDUAL

```
function [x,v]=PRIMALDUAL(A,b,c)
[m,n]=size(A);e=ones(n,1);gap=1.e14;
x=ones(n,1);y=ones(m,1);z=ones(n,1);
while abs(gap/(1+b'*y))>0.0001
    X=diag(x);Z=diag(z);u=abs(gap)/(n^2);
    dP=b-A*x;
    dD=A'*y+z-c;
    matriz=(A*inv(Z)*X*A');
    if rcond(matriz)<1.e-14
        v=inf;break;
    else
        dy=(matriz)\(b-u*A*inv(Z)*e-A*inv(Z)*X*dD);
    end
    dz=-dD-A'*dy;
    dx=inv(Z)*(u*e-X*Z*e-X*dz);
    aP=1;aD=1;
    for i=1:n
        if dx(i)<0
            minimo=-x(i)/dx(i);
            if minimo<aP
                aP=minimo;
            end
        end
        if dz(i)<0
            minimo=-z(i)/dz(i);
                if minimo<aD
                    aD=minimo;
                end
        end
    end
    aP=0.98*aP;aD=0.98*aD;
    x=x+aP*dx; y=y+aD*dy;z=z+aD*dz;
    v=c'*x;gap=v-y'*b;
end
```

3.4.4 El Programa PREDICTORCORRECTOR

```
function [x,v,iter]=PIPC(A,b,c)
[m,n]=size(A); e=ones(n,1); eps=0.0000001;
x=ones(n,1);y=ones(m,1); z=ones(n,1); iter=0;
while (x'*z)>eps
    X=diag(x); Z=diag(z); dP=b-A*x;dD=c-A'*y-z;
    matriz=A*inv(Z)*X*A';
    if rcond(matriz)<1.e-14
        v=inf;break;
    else
    dya=matriz\(b+A*inv(Z)*X*dD);
    end
    dza=dD-A'*dya;
    if rcond(Z)<1.e-14
        break;
    else
    dxa=Z\(-X*Z*e-X*dza);
    end
    aP=1;aD=1;
    for i=1:n
        if dxa(i)<0 & -x(i)/dxa(i)<aP
            aP=-x(i)/dxa(i);
        end
        if dza(i)<0 & -z(i)/dza(i)<aD
            aD=-z(i)/dza(i);
        end
    end
    x=x+0.98*aP*dxa;y=y+0.98*aD*dya;z=z+0.98*aD*dza;
u=(((x+0.98*aP*dxa)'*(z+0.98*aD*dza))/(x'*z))^2*(((x+0.98*aP*dxa)'
*(z+0.98*aD*dza))/n);
    DX=diag(dxa);DZ=diag(dza);dP=b-A*x;dD=c-A'*y-z;
 dyc=(A*inv(Z)*X*A')\(b-u*A*inv(Z)*e+A*inv(Z)*DX*DZ*e+A*inv(Z)*X*dD);
    dzc=dD-A'*dyc;
    dxc=Z\(u*e-X*Z*e-DX*DZ*e-X*dzc);
    aP=1;aD=1;
    for i=1:n
       if dxc(i)<0 & -x(i)/dxc(i)<aP
          aP=-x(i)/dxc(i);
        end
        if dzc(i)<0 & -z(i)/dzc(i)<aD
           aD=-z(i)/dzc(i);
        end
    end
    x=x+0.98*aP*dxc; y=y+0.98*aD*dyc;  z=z+0.98*aD*dzc;
    iter=iter+1;
end;end
v=c'*x;
format short;
```

3.4.5 El Programa TESTINT

```
function [x,ques,k,xi,xd]=TESTINT(x,nv)
aux=x-round(x);
if norm(aux(1:nv),inf)<0.001
    x(1:nv)=round(x(1:nv)); ques='si'; k=0; xi=0;xd=0;
else
    ques='no';
    [m,k]=max(abs(aux(1:nv)));
    xi=floor(x(k));
    xd=floor(x(k))+1;
end
```

3.4.6 El Programa CONSTRUYEI

```
function [Ai,bi,ci]=CONSTRUYEI(xi,k,A,b,c)
[m,n]=size(A);
cank=zeros(1,n+1);
cank(k)=1;
cank(n+1)=1;
Ai=[A zeros(m,1);cank];
bi=[b;xi+0.001];
ci=[c;0];
```

3.4.7 El Programa CONSTRUYED

```
function [Ad,bd,cd]=CONSTRUYED(xd,k,A,b,c)
[m,n]=size(A);
cank=zeros(1,n+1);
cank(k)=1;
cank(n+1)=-1;
Ad=[A zeros(m,1);cank];
bd=[b;xd-0.001];
cd=[c;0];
```

3.5 Programación Dinámica

La programación dinámica se encuentra incluida dentro de un conjunto de técnicas matemáticas, que a su vez forman parte de un área más amplia conocida como Investigación de Operaciones. Esta última puede definirse como una ciencia interdisciplinaria que tiene por objeto la búsqueda de estrategias que permitan obtener resultados óptimos en el desarrollo de actividades por parte de sistemas hombre-máquina (es decir, formado exclusivamente por hombres, por máquinas o por la combinación de los dos).

Los problemas de la programación dinámica, son aquellos que pueden ser divididos en subproblemas, los cuales, a su vez, tienen una estructura igual al problema original (podría decirse que tiene una estructura "fractal", posteriormente mostraremos una de ellas "La alfombra de Sierpinski"). El método consiste en dividir el problema en etapas, resolver la primera de estas, utilizar esta solución para resolver la etapa siguiente y continuar así sucesivamente hasta encontrar la solución del problema en su totalidad, la naturaleza del método es recursivo (Rodríguez, 2005). La recursividad se mostró en el capítulo 2.

Veremos ahora como solucionar un problema de programación lineal

Consideremos el siguiente problema de programación lineal:

$$\text{Maximizar } z = c_1x_1 + c_2x_2 + \ldots c_px_p$$

Sujeto a:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1p}x_p &\le b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2p}x_p &\le b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mp}x_p &\le b_m \\ x_1, x_2, \ldots, x_p &\ge 0 \end{aligned}$$

Se puede formular como un problema de programación dinámica. El problema de programación lineal es de variable continua. El nivel de actividad x_j (≥ 0) representa la actividad en la etapa *j,*(es decir, cada actividad *j* (*j=1,2,...,p*) se puede considerar como una etapa). Cada etapa posee un número infinito de alternativas dentro del espacio factible. Los estados pueden definirse como las cantidades de recursos que se asignan a la etapa

actual y a las anteriores, además los estados pueden representarse con un vector de *m* dimensiones.

Sean $(v_{1j}, v_{2j}, ..., v_{mj})$ los estados del sistema en la etapa *j*, o sea, las cantidades de los recursos 1, 2,…, *j*. Sea $f_j(v_{1j}, v_{2j}, ..., v_{mj})$ el valor óptimo de la función objetivo para las etapas 1,2,…, *j* dados los estados $v_{1j}, v_{2j}, ..., v_{mj}$. Por lo tanto

$$f_1(v_{11}, v_{21}, ..., v_{m1}) = \underset{\substack{0 \le a_{i1} \le v_{i1} \\ i=1,2,...,m}}{máx} \{c_1 x_1\} \tag{3.8}$$

$$f_j(v_{1j}, v_{2j}, ..., v_{mj}) = \underset{\substack{0 \le a_{ij} \le v_{ij} \\ i=1,2,...,m}}{máx} \{c_j x_j + f_{j-1}(v_{1j} - a_{1j}x_{1j}, \ldots, v_{mj} - a_{mj}x_{mj})\} \tag{3.9}$$
$$j = 2, 3, \ldots, p$$

Donde $0 \le v_{ij} \le b_i$ para todas *i* y *j*. (Rodríguez, 2005)

Ejemplo: (Problema de Programación Lineal)

Considere el siguiente problema de programación lineal de variable continua:

$$\text{Maximizar } 20x_1 + 16x_2$$
$$\text{Sujeto a:}$$
$$5x_1 + 3x_2 \le 105$$
$$2x_1 + 4x_2 \le 70$$
$$x_1, x_2 \ge 0$$

Al resolver este problema aplicamos (3.8)

$$f_1(v_{11}, v_{21}) = \underset{\substack{0 \le 5x_1 \le v_{11} \\ 0 \le 2x_1 \le v_{21}}}{máx} \{20x_1\}$$

Como $5x_1 \le v_{11}$ y $2x_1 \le v_{21}$ se concluye que $x_1 \le mín\left\{\frac{v_{11}}{5}, \frac{v_{21}}{2}\right\}$ como se trata de maximizar se tiene que $x_1 = mín\left\{\frac{v_{11}}{5}, \frac{v_{21}}{2}\right\}$ y por lo tanto

$$f_1(v_{11}, v_{21}) = 20\, mín\left\{\frac{v_{11}}{5}, \frac{v_{21}}{2}\right\}$$

Ahora pasamos hacer los cálculos de la segunda etapa:

$$f_2(v_{12}, v_{22}) = \underset{\substack{0\le 3x_2\le v_{12}\\ 0\le 4x_2\le v_{22}}}{máx}\left\{16x_2 + 20\, mín\left\{\frac{v_{12}-3x_2}{5}, \frac{v_{22}-4x_2}{2}\right\}\right\}$$

Pero se tiene $v_{12} = 105$ y $v_{22} = 70$, entonces, $3x_2 \le 105$ y $4x_2 \le 70$ lo cual equivale a $x_2 \le 35/2$. Se tiene

$$f_2(v_{12}, v_{22}) = \underset{0\le x_2\le 35/2}{máx}\left\{16x_2 + 20\, mín\left\{\frac{105-3x_2}{5}, \frac{70-4x_2}{2}\right\}\right\}$$

Para $0 \le x_2 \le 35/2$ es necesario resolver $\frac{105-3x_2}{5} \le \frac{70-4x_2}{2}$, tiene como solución $x_2 \le 10$. Con lo cual se tiene

$$f_2(v_{12}, v_{22}) = máx\left\{16x_2 + 20\left\{\begin{array}{l}\frac{105-3x_2}{5},\ 0 \le x_2 \le 10\\ \frac{70-4x_2}{2},\ 10 \le x_2 \le 35/2\end{array}\right.\right\}$$

o de forma equivalente

$$f_2(v_{12}, v_{22}) = máx\left\{\begin{array}{l}420 + 4x_2,\ 0 \le x_2 \le 10\\ 700 - 24x_2,\ 10 \le x_2 \le 35/2\end{array}\right\}$$

En el intervalo [0, 10] la función es creciente y por lo tanto tiene su máximo en $x_2 = 10$ con un valor de 460; en [10, 35/2] la función es decreciente donde el máximo está también en $x_2 = 10$ y tiene el mismo valor, en donde se concluye que en [0, 35/2] el máximo se localiza en $x_2 = 10$ y tiene un valor de 460.

Para obtener x_1 se tiene en cuenta que:

$$v_{11} = v_{12} - 3x_2 = 105 - 30 = 75$$
$$v_{21} = v_{22} - 4x_2 = 70 - 40 = 30$$

pero como $x_1 = \min\left\{\frac{v_{11}}{5}, \frac{v_{21}}{2}\right\}$ entonces $x_1 = 15$.

En el siguiente ejemplo se resolverá un problema de programación lineal entera con una leve variación del procedimiento del ejemplo anterior.

Ejemplo: Problema de Programación Lineal Entera

Considere el siguiente problema

$$\begin{aligned} &\text{Maximizar } 8x_1 + 7x_2 \\ &\text{Sujeto a:} \\ &\qquad 2x_1 + x_2 \le 8 \\ &\qquad 5x_1 + 2x_2 \le 15 \\ &\qquad x_1, x_2 \text{ enterosno negativos} \end{aligned}$$

Al resolver el problema tenemos que

$$f_1(v_{11}, v_{21}) = \max_{\substack{0 \le 2x_1 \le v_{11} \\ 0 \le 5x_1 \le v_{21}}} \{8x_1\}, \; x_1 \text{ entero}$$

Como $2x_1 \le v_{11}$ y $5x_1 \le v_{21}$ se concluye que $x_1 \le mín\left\{\frac{v_{11}}{2}, \frac{v_{21}}{5}\right\}$ además tratándose de un problema de maximización con valores enteros se tiene que $x_1 = \left\lfloor mín\left\{\frac{v_{11}}{2}, \frac{v_{21}}{5}\right\}\right\rfloor$ (donde $\lfloor x \rfloor$ representa la parte entera de x), por lo tanto

$$f_1(v_{11}, v_{21}) = 8\left\lfloor mín\left\{\frac{v_{11}}{2}, \frac{v_{21}}{5}\right\}\right\rfloor$$

Dado que $v_{12}=8$ y $v_{22}=15$ para la segunda etapa se tiene:

$$f_2(v_{12},v_{22})=\underset{\substack{0\le x_2\le 8\\ 0\le 2x_2\le 15}}{máx}\left\{7x_2+8\left\lfloor mín\left\{\frac{8-x_2}{2},\frac{15-2x_2}{5}\right\}\right\rfloor\right\}$$

las desigualdades $x_2\le 8$ y $2x_2\le 15$ equivalen a $x_2\le 7$ (x_2 entero), entonces para este rango resolvemos las desigualdades $\frac{8-x_2}{2}\le\frac{15-2x_2}{5}$, tiene como solución $x_2\ge 10$, por lo tanto

$$f_2(v_{12},v_{22})=\underset{0\le x_2\le 7}{máx}\left\{7x_2+8\left\lfloor\frac{15-2x_2}{5}\right\rfloor\right\}$$

En la siguiente tabla mostraremos la solución $f_2(v_{12},v_{22})$

x_2	$7x_2+8\left\lfloor\frac{15-2x_2}{5}\right\rfloor$
0	24
1	23
2	30
3	29
4	36
5	43
6	42
7	49

Vemos que el máximo se consigue con $x_2=7$ y que tiene un valor de 49.

Como

$$x_1=\left\lfloor mín\left\{\frac{8-x_2}{2},\frac{15-2x_2}{5}\right\}\right\rfloor \text{ entonces } x_1=\left\lfloor mín\left\{\frac{1}{2},\frac{1}{5}\right\}\right\rfloor=0.$$

Este método que resuelve problemas de programación dinámica, sólo trabaja para cierta cantidad de variables, es decir para problemas de grandes dimensiones no encuentra solución. Ahora veamos la Alfombra de Sierpinski

Alfombra de Sierpinski

Anteriormente mencionamos que los problemas de programación dinámica pueden ser divididos en subproblemas, los cuales, a su vez, tienen una estructura igual al problema original, podría decirse que tiene una estructura "fractal", aquí mostraremos uno de ellos la alfombra de Sierpinski".

La alfombra de Sierpinski fue publicado en 1916 por el matemático de origen polaco Waclaw Sierpinski. (Rodriguez, Crespo y Jiménez, 2015)

La construcción de este fractal se define de forma recursiva, siguiendo los siguientes pasos: Primero, se tiene un cuadrado y se divide en nueves cuadrados iguales de lado 1/3, eliminando el de la mitad. Se vuelve aplicar el paso anterior a los ocho cuadrados restantes.

En la siguiente figura se observa las primeras cinco iteraciones.

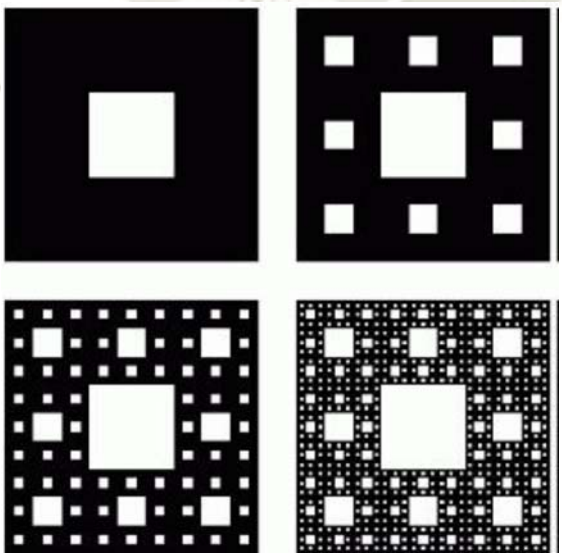


Figura 3.14. [Imagen de Rodriguez et al]. (2015). Proyecto Alfombra de Sierpinski, (p.3). Universidad de Almeira, España.

Para conocer la dimensión de un fractal, B. Mandelbrot propuso un método, es el siguiente:

Si una figura de dimensión D se puede componer a partir de n copias de escala $(1/S)$ se tiene que $S^D = n$, entonces, en la Alfombra de Sierpinski de lado 1 se puede componer a partir de 9 copias de lado 1/3, eliminando el interior del cuadrado que ocupa la posición central, por lo tanto tenemos $\left(3^D = 8\right) \rightarrow D = \frac{\log 8}{\log 3} \approx 1.893$.

Finalmente podemos ver que los problemas de programación dinámica y la geometría fractal de la Alfombra de Sierpinski, ambos se pueden subdividir, el cual en PD tiene una estructura igual al problema original, en cambio en la Alfombra de Sierpinski tiene una estructura igual a la figura original, ambos aplican técnicas de recursividad.

En la siguiente sección presentaremos diferentes problemas donde se aplicará los métodos de *PLEPIPD* y *PLEPIPC* para sus soluciones.

3.6 Experimentos Computacionales

Mostraremos a continuación los experimentos realizados a los diversos problemas de programación lineal entera, el cual se aplicara los métodos *PLEPIPD* y *PLEPIPC*.

Problema 3.1: Considere el siguiente problema de programación lineal entera con 2 variables originales y 2 restricciones.

$$\begin{aligned} &\text{Minimizar } -x_1 - 5x_2 \\ &\text{Sujeto a:} \\ &\qquad 6x_1 + 7x_2 \le 44 \\ &\qquad 9x_1 \;\; + x_2 \le 36 \\ &\qquad\quad x_1, x_2 \ge 0 \quad \text{enteros} \end{aligned} \tag{3.10}$$

La región factible del problema original es la siguiente

El algoritmo no se aplica directamente sobre este problema, necesitamos estandarizarlo. Para eso introducimos las variables de holgura x_3, x_4, de donde obtenemos lo siguiente.

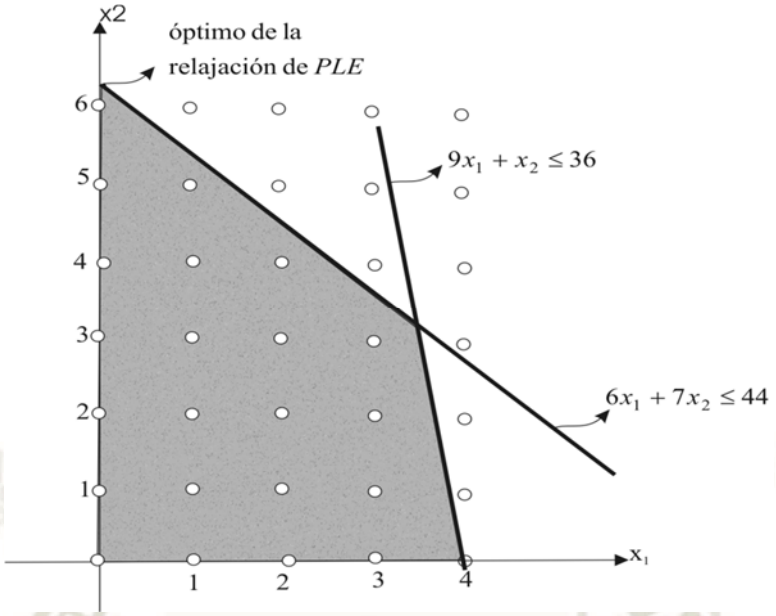


Figura 3.15.

Minimizar $-x_1 - 5x_2 + 0x_3 + 0x_4$

Sujeto a:

$$\begin{aligned} 6x_1 + 7x_2 + x_3 &= 44 \\ 9x_1 + x_2 + x_4 &= 36 \\ x_1, x_2, x_3, x_4 &\geq 0 \\ x_1, x_2 &\text{ enteras} \end{aligned}$$

El ejemplar correspondiente a este problema (datos numéricos ingresados) es:

```
A = [6     7     1     0
     9     1     0     1]
b = [44   36]'
c = [-1  -5  0  0]'
```

Llamamos al algoritmo *PLEPIPD*, indicando sólo las dos primeras variables (originales) sean enteras, éste otorgó la siguiente respuesta:

```
fx =
         0
    6.0000
    1.9886
   29.9920
```

```
    0.0000
    0.0002

valor =
  -30.0056

iter =
     5

cota =
  -30.0056
```

Los dos primeros valores de fx son justamente los valores óptimos enteros de las variables originales del problema (3.1), el resto son valores de las variables auxiliares (holgura y exceso) que surgieron en el momento de las ramificaciones y las cuales no requieren ser enteras. El *valor* de -30.0056 es una aproximación al valor objetivo óptimo asociado al problema (3.1). El valor de *iter* nos indica la cantidad de subproblemas (relajaciones) resueltos por el algoritmo *PIPD*. El valor de "cota" representa el mejor valor objetivo obtenido correspondiente a una solución entera.

Por lo tanto, la solución entera óptima del problema original (3.1) es $x^* = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$, y el valor objetivo óptimo es $c^T x^* = -30$.

Ahora, con respecto al algoritmo *PLEPIPC* aplicado al mismo problema, también indicando sólo las dos primeras variables (originales) sean enteras, nos otorga la siguiente respuesta:

```
fx =
         0
    6.0000
    1.9917
   29.9959
    0.0001
    0.0007
valor =
  -30.0046

iter =
     6

cota =
  -30.0046
```

En conclusión, aplicando los algoritmos *PLEPIPD* y *PLEPIPC* al problema original (3.1), la solución entera óptima es $x^* = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$, y el valor objetivo óptimo es $c^T x^* = -30$.

Problema 3.2: Considere el problema

$$\begin{aligned} &\text{Minimizar } -8x_1 - 5x_2 \\ &\text{Sujeto a:} \\ &x_1 + x_2 \le 6 \\ &9x_1 + 5x_2 \le 45 \\ &x_1, x_2 \ge 0 \\ &x_1, x_2, \text{enteros} \end{aligned}$$

La región factible del problema se muestra en la siguiente figura 3.16.

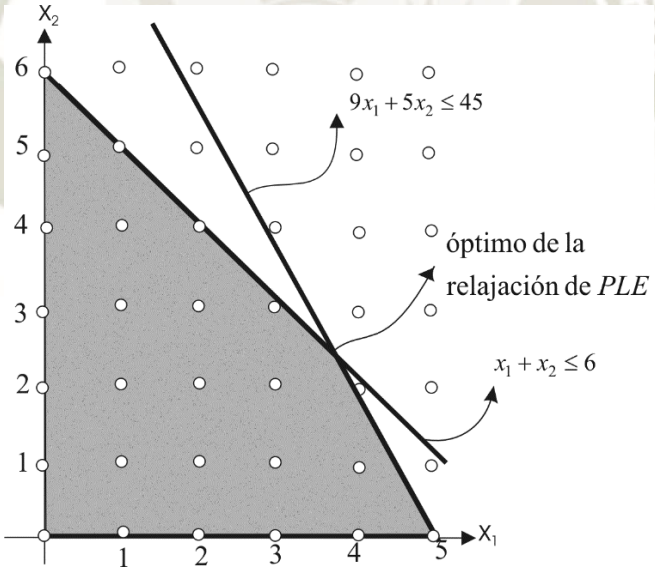


Figura 3.16.

Estandarizamos el problema introduciendo las variables de holgura x_3 y x_4, luego

Minimizar $-8x_1 - 5x_2 + 0x_3 + 0x_4$

Sujeto a:

$$x_1 + x_2 + x_3 = 6$$
$$9x_1 + 5x_2 + x_4 = 45$$
$$x_1, x_2, x_3, x_4 \geq 0$$
x_1, x_2 enteras

El ejemplar correspondiente al problema es

```
A = [1     1     1     0
     9     5     0     1]
b = [6    45]'
c = [-8   -5     0     0]'
```

La solución otorgada por el algoritmo *PLEPIPD* es la siguiente

```
fx =
    5.0000
         0
    0.9998
    0.0009
    1.0006
    1.0004
    0.0006
    0.0004

valor =
  -39.9995

iter =
     9

cota =
  -39.9995
```

Por lo tanto, la solución entera óptima del problema original es $x^* = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, y el valor objetivo óptimo es $c^{\mathrm{T}} x^* = -39.9995$.

Note que el valor objetivo óptimo de este problema es en realidad -40, pero el algoritmo *PLEPIPD* nos retorna -39.9995, esta diferencia se debe a que el algoritmo *PIPD* re-

suelve cada subproblema del tipo PL de forma aproximada. Esta es una característica propia de los métodos de puntos interiores para programación lineal. Debemos tener presente esta observación para todos los problemas que se presenten.

Ahora, al aplicar el algoritmo *PLEPIPC* nos otorga la siguiente solución:

```
fx =
    5.0000
         0
    0.9996
    0.0000
    2.0000
    1.0004
    1.0000
    0.0004
    0.0000
    0.0000
    0.0004
    0.0004
    0.0004


valor =
  -40.0006

iter =
     3

cota =
  -40.0006
```

Por lo tanto, la solución entera óptima del problema original es $x^* = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, y el valor objetivo óptimo es $c^{\mathrm{T}} x^* = -40.0006$.

Problema 3.3: Considere el problema

$$\begin{aligned} &\text{Minimizar } 2x_1 + x_2 \\ &\text{Sujeto a:} \\ &\qquad x_1 + x_2 \le 5 \\ &\qquad -x_1 + x_2 \le 0 \\ &\qquad 6x_1 + 2x_2 \le 21 \\ &\qquad x_1, x_2 \ge 0 \quad x_1, x_2, \text{enteros} \end{aligned}$$

La región de factibilidad del problema entero es el siguiente

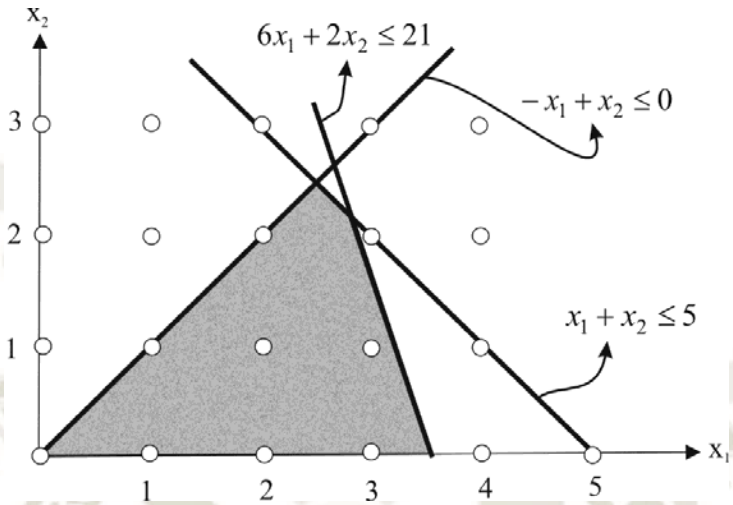


Figura 3.17.

Estandarizamos el problema introduciendo las variables de holgura x_3, x_4, x_5.

$$\text{Minimizar } 2x_1 + x_2 + 0x_3 + 0x_4 + 0x_5$$

Sujeto a:

$$\begin{aligned} x_1 + x_2 + x_3 &= 5 \\ -x_1 + x_2 + x_4 &= 0 \\ 6x_1 + 2x_2 + x_5 &= 21 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

$$x_1, x_2, \text{ enteros}$$

Ahora mostraremos el ejemplar del problema *PLE*

```
A = [ 1     1     1     0     0
     -1     1     0     1     0
      6     2     0     0     1]
b = [ 5  0  21]'
c = [ 2  1  0  0  0]'
```

La solución de *PLE* , otorgada por el algoritmo *PLEPIPD*, es la siguiente

```
fx =
         0
         0
    5.0000
    0.0000
   21.0000

valor =
  1.5302e-005

iter =
     1

cota =

  1.5302e-005
```

La solución entera óptima del problema original es $x^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, y el valor objetivo óptimo es $c^{\mathrm{T}} x^* = 1.53 \times 10^{-5}$.

Ahora, la solución de PLE , otorgada por el algoritmo *PLEPIPC*, es la siguiente

```
fx =
         0
         0
    5.0000
    0.0000
   21.0000
    0.0010

valor =
  1.3511e-006

iter =
     9

cota =
  1.3511e-006
```

Entonces, la solución entera óptima del problema original es $x^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, y el valor objetivo óptimo es $c^{\mathrm{T}} x^* = 1.35 \times 10^{-6}$.

Problema 3.4: Considere el problema

$$\text{Minimizar} \quad -3x_1 - 4x_2 - x_3$$

Sujeto a:

$$\begin{aligned} 2x_1 + 4x_2 + 3x_3 &\leq 15 \\ x_1 + 3x_2 + 5x_3 &\leq 30 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

$$x_1, x_2, x_3, \text{ enteros}$$

Ahora mostraremos el ejemplar del problema de *PLE*

```
A = [2     4     3     1     0
     1     3     5     0     1]
b = [15   30]'
c = [-3 -4 -1 0  0]'
```

Ahora la solución del problema *PLE* , otorgada por el algoritmo *PLEPIPD*, es la siguiente:

```
fx =
    7.0000
         0
         0
    0.9917
   22.9919
    0.0000
    0.0000
    0.0002

valor =
  -21.0076

iter =
     7

cota =
  -21.0076
```

La solución entera óptima del problema original es $x^* = [7 \quad 0 \quad 0]^T$, y el valor objetivo óptimo es $c^T x^* = -21.0076$.

Ahora, la solución de PLE , otorgada por el algoritmo *PLEPIPC*, es la siguiente

```
fx =
    7.0000
         0
         0
    0.9962
   22.9933
    0.0000
    0.0007
    0.0015
    0.0000
    0.0007
    0.0007
    0.0024
    0.0024
    0.0007
    0.0007

valor =
  -21.0025

iter =
     4

cota =
  -21.0025
```

Entonces, la solución entera óptima del problema original es $x^* = \begin{bmatrix} 7 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$, y el valor objetivo óptimo es $c^{\mathrm{T}} x^* = -21.0025$.

Problema 3.5: Considere el siguiente problema, el cual está constituido por diez variables y cinco restricciones.

$$\text{Minimizar} \quad 3x_1 + 5x_2 + x_3 + 2x_4 + x_5 + 4x_6 + 3x_7 + x_8 + 2x_9 + 2x_{10}$$

Sujeto a:

$$\begin{aligned} x_1 + x_2 + x_4 + x_5 + x_8 + x_9 &\geq 1 \\ x_1 + x_3 &\geq 1 \\ x_2 + x_5 + x_7 + x_{10} &\geq 1 \\ x_3 + x_6 + x_8 + x_9 &\geq 1 \\ x_1 + x_2 + x_4 + x_5 + x_7 + x_9 + x_{10} &\geq 1 \\ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10} &\geq 0 \end{aligned}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, \text{ enteros}$$

Estandarizamos el problema introduciendo las variables de exceso $x_{11}, x_{12}, x_{13}, x_{14}, x_{15}$, de donde obtenemos el problema

$$\text{Minimizar } 3x_1 + 5x_2 + x_3 + 2x_4 + x_5 + 4x_6 + 3x_7 + x_8 + 2x_9 + 2x_{10} + 0x_{11} + 0x_{12} + 0x_{13} + 0x_{14} + 0x_{15}$$

Sujeto a:

$$\begin{aligned}
x_1 + x_2 + 0x_4 + x_5 + x_8 + x_9 + x_{11} &= 1 \\
x_1 + x_3 + x_{12} &= 1 \\
x_2 + x_5 + x_7 + x_{10} + x_{13} &= 1 \\
x_3 + x_6 + x_8 + x_9 + x_{14} &= 1 \\
x_1 + x_2 + x_4 + x_5 + x_7 + x_9 + x_{10} + x_{15} &= 1 \\
x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15} &\geq 0 \\
x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, &\text{ enteras}
\end{aligned}$$

El ejemplar del problema es el siguiente

```
A =
[1   1   0   1   1   0   0   1   1   0  -1   0   0   0   0
1   0   1   0   0   0   0   0   0   0   0  -1   0   0   0
0   1   0   0   1   0   1   0   0   1   0   0  -1   0   0
0   0   1   0   0   1   0   1   1   0   0   0   0  -1   0
1   1   0   1   1   0   1   0   1   1   0   0   0   0  -1]

b = [1  1  1  1  1]'


c = [3  5  1  2  1  4  3  1  2  2  0  0  0  0  0]'
```

La solución otorgada por el algoritmo *PLEPIPD* es:

```
fx =
         0
         0
    1.0000
         0
    1.0000
         0
         0
         0
         0
```

```
          0
    0.0001
    0.0000
    0.0000
    0.0001
    0.0001

valor =
    2.0002

iter =
     1

cota =
    2.0002
```

La solución entera óptima del problema original es $x^* = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$, y el valor objetivo óptimo es $c^T x^* = 2$.

Ahora, la solución de PLE, otorgada por el algoritmo *PLEPIPC*, es la siguiente

```
fx =
          0
          0
    1.0000
          0
    1.0000
          0
          0
          0
          0
          0
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000

valor =
    2.0000

iter =
     6

cota =
    2.0000
```

Entonces, la solución entera óptima del problema original es:

$x^* = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$, y el valor objetivo óptimo es $c^T x^* = 2$.

Problema 3.6: Considere el siguiente problema

Minimizar $2x_1 - x_2 + x_3 - x_4$

Sujeto a:

$$1.5x_1 + 1.7x_2 + x_3 + 5.2x_4 \leq 100.520$$
$$-1.12x_1 + 1.89x_2 - 1.9x_3 + 6.33x_4 \leq 200.7$$
$$2.45x_1 + 5.5x_2 + 10x_3 + 8.4x_4 \leq 500.89$$
$$x_1, x_2, x_3, x_4 \geq 0 \quad x_1, x_2, x_3, x_4, \text{enteros}$$

Estandarizando el problema adicionamos las variables de holgura x_5, x_6, x_7.

Minimizar $2x_1 - x_2 + x_3 - x_4 + 0x_5 + 0x_6 + 0x_7$

Sujeto a:

$$1.5x_1 + 1.7x_2 + x_3 + 5.2x_4 + x_5 = 100.520$$
$$-1.12x_1 + 1.89x_2 - 1.9x_3 + 6.33x_4 + x_6 = 200.7$$
$$2.45x_1 + 5.5x_2 + 10x_3 + 8.4x_4 + x_7 = 500.89$$
$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0$$
$$x_1, x_2, x_3, x_4, \text{enteros}$$

Ahora mostraremos el resultado del algoritmo *PLEPIPD*

```
fx =
         0
   59.0000
         0
         0
    0.2151
   89.1858
  176.3771
    0.0003
    0.0004

valor =
  -59.0006

iter =
     5

cota =
```

```
   -59.0006
```

Por lo tanto la solución entera óptima del problema original es $x^* = [0 \quad 59 \quad 0 \quad 0]^T$, y el valor objetivo óptimo es $c^T x^* = -59.0006$.

Ahora, la solución de PLE, otorgada por el algoritmo *PLEPIPC*, es la siguiente

```
fx =
         0
   59.0000
         0
         0
    0.2131
   89.1818
  176.3761
    0.0000
    0.0020
    0.0000

valor =
  -59.0020

iter =
     8

cota =
  -59.0020
```

Entonces, la solución entera óptima del problema original es $x^* = [0 \quad 59 \quad 0 \quad 0]^T$, y el valor objetivo óptimo es $c^T x^* = -59.0006$.

Problema 3.7: Considere el siguiente problema (Hipercubo)

$$\begin{aligned} &\text{Minimizar} \quad -x_1 - x_2 - x_3 \\ &\text{Sujeto a:} \\ &\qquad x_1 \le 124.5 \\ &\qquad x_2 \le 124.5 \\ &\qquad x_3 \le 124.5 \\ &\qquad x_1, x_2, x_3 \ge 0 \quad \text{enteros} \end{aligned}$$

El correspondiente grafico nos muestra la región de factibilidad del problema entero.

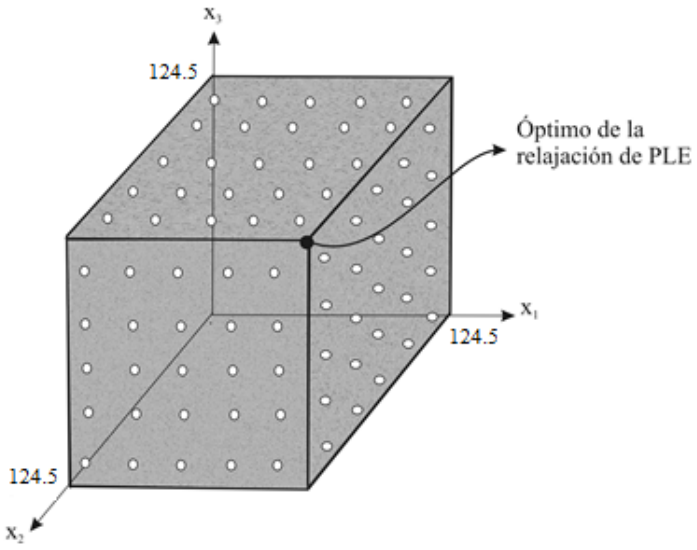


Figura 3.18. Región de factibilidad del problema entero (Hipercubo)

Estandarizamos el problema introduciendo las variables de holgura x_4, x_5, x_6:

Minimizar $-x_1 - x_2 - x_3 + x_4 + x_5 + x_6$

Sujeto a:

$$x_1 + x_4 = 124.5$$
$$x_2 + x_5 = 124.5$$
$$x_3 + x_6 = 124.5$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$
$$x_1, x_2, x_3, \text{enteros}$$

Ahora mostraremos el ejemplar del problema

```
A =
     1     0     0     1     0     0
     0     1     0     0     1     0
     0     0     1     0     0     1
b =
  124.50000000000000
  124.50000000000000
  124.50000000000000
c =
    -1
    -1
    -1
     0
     0
```

```
     0
```

En la primera iteración la solución óptima del problema de relajación es:

```
x =
  124.4985
  124.4985
  124.4985
    0.0015
    0.0015
    0.0015

v =
 -373.4955
```

Como no es una solución entera, entonces el problema original se subdivide en dos nuevos subproblemas dando origen a dos nuevos subconjuntos S1 y S2, donde S2 es vacío, ya que no contiene soluciones factibles. Nosotros pasamos a analizar el subconjunto S1 hasta encontrar la solución óptima entera.

En la siguiente figura 3.19 se aprecia la modificación de la región factible con sus respectivos subconjuntos S1 y S2.

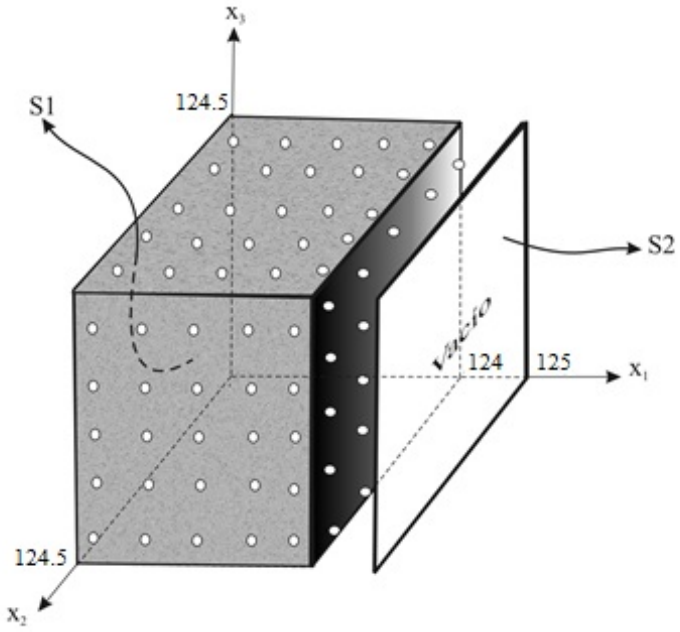


Figura 3.19. Modificación de la región factible (hipercubo) en dos nuevos subconjuntos S1 y S2

Por lo tanto la solución dada por el algoritmo *PLEPIPD* es la siguiente

```
fx =
  124.0000
  124.0000
  124.0000
    0.4999
    0.4999
    0.4999
    0.0009
    0.0009
    0.0009

valor =
 -372.0002

iter =
     7

cota =
 -372.0002
```

La solución entera óptima del problema original es $x^* = [124\ 124\ \ 124]^T$, y el valor objetivo óptimo es $c^T x^* = -372.0002$.

Ahora, la solución de *PLE* , otorgada por el algoritmo *PLEPIPC*, es la siguiente

```
fx =
  124.0000
  124.0000
  124.0000
    0.4990
    0.5007
    0.4990
    4.0000
    0.0017
    0.0000
    0.0000
    0.0017
    0.0020

valor =
 -372.0009

iter =
     5

cota =
 -372.0009
```

Entonces, la solución entera óptima del problema original es $x^* = [124\ 124\ \ 124]^T$, y el valor objetivo óptimo es $c^T x^* = -372.0009$.

Problema 3.8: Considere el problema (Hipercubo en dimensión n=50)

$$\begin{aligned}
&\text{Minimizar} \quad -x_1 - x_2 - \ldots - x_{50} \\
&\text{Sujeto a:} \\
&\qquad x_1 \le 22.5 \\
&\qquad x_2 \le 22.5 \\
&\qquad \vdots \\
&\qquad x_{50} \le 22.5 \\
&\qquad x_1,\ x_2\ , \ldots , x_{50} \ge 0 \quad \text{enteros}
\end{aligned}$$

Estandarizando el problema introducimos las variables de holgura $x_{51},\ x_{52},\ \ldots\ , x_{100}$.

$$\begin{aligned}
&\text{Minimizar} \quad -x_1 - x_2 - \ \ldots\ - x_{50} + 0x_{51} \ldots + 0x_{100} \\
&\text{Sujeto a:} \\
&\qquad x_1 + x_{51} \qquad\qquad\qquad = 22.5 \\
&\qquad x_2 \quad + x_{52} \qquad\qquad = 22.5 \\
&\qquad \vdots \\
&\qquad x_{50} \quad \ldots \quad + x_{100} = 22.5 \\
&\qquad x_1,\ x_2\ , \ldots , x_{50}, x_{51}, \ldots, x_{100} \ge 0 \\
&\qquad x_1,\ x_2\ ,\ \ldots , x_{50}, \text{enteras}
\end{aligned}$$

La solución entera óptima entera otorgada por el algoritmo *PLEPIPD* es el siguiente

```
fx =
     x_i

     x_j

     x_k

valor =
 -1.1000e+003
```

```
iter =
   101

cota =
 -1.1000e+003
```

Donde $x_i = 22.0000$, para $i = 1,...,50$, $x_j = 0.4990$, para $j = 1,...,50$ y $x_k = 0.0000$, para $k = 1,...,50$. El computador muestra la solución entera en un tiempo de 39.235 seg.

Entonces la solución óptima del problema es $x^* = [x_1 \; \ldots \; x_{50}]^T$, y el valor objetivo óptimo es $c^T x^* = -1.10 * 10^3$.

Ahora, la solución de *PLE* , otorgada por el algoritmo *PLEPIPC*, es la siguiente

```
fx =
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
   22.0000
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.4996
    0.4990
    0.4990
    0.4990
    0.4990
    0.4990
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0006
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0006
    2.0020

valor =
 -1.1000e+003

iter =
     6

cota =
 -1.1000e+003
```

Por lo tanto, la solución entera óptima del problema es $x^* = [x_1 \ \ldots x_{50}]^T$, y el valor objetivo óptimo es $c^T x^* = -1.10 * 10^3$.

En los siguientes problemas de Aplicación, sólo aplicaremos el algoritmo *PLEPIPD,* ya que nos muestran los resultados en menor tiempo que el algoritmo de *PLEPIPC*, esto porque la recursividad del algoritmo *PLEPIPC* es mayor con respecto al algoritmo *PLEPIPD*.

Problema 3.9:

(Problema de Aplicación)

Consideremos el siguiente problema de aplicación:

Supongamos que tenemos que escribir un trabajo en un tiempo máximo de 40 horas, el trabajo consta de 200 páginas de texto, 62 tablas y 38 figuras. Para realizar este trabajo fueron consultados 4 trabajadores diferentes (tipeadores), los detalles de la rapidez, tarifa y disponibilidad de cada trabajador son dadas en la siguiente tabla

	Trabajador 1		Trabajador 2		Trabajador 3		Trabajador 4	
	Rapidez	Costo en soles por unidad	Rapidez	Costo en soles por unidad	Rapidez	Costo en soles por unidad	Rapidez	Costo en soles por unidad
Texto	8 páginas por hora	0.5 soles por página	10 páginas por hora	0.55 soles por página	9 páginas por hora	0.6 soles por página	14 páginas por hora	0.65 soles por página
Tabla	12 tablas por hora	0.6 soles por tabla	11 tablas por hora	0.6 soles por tabla	10 tablas por hora	0.5 soles por tabla	11 tablas por hora	0.55 soles por tabla
Figura	5 figuras por hora	1.2 soles por figura	2 figuras por hora	0.9 soles por figura	3 figuras por hora	1 sol por figura	2 figuras por hora	0.8 soles por figura
Disponibilidad	14 horas		16 horas		12 horas		10 horas	

Se pide determinar la política óptima para distribuir el trabajo entre los tipeadores, de modo que demanden el menor costo posible (Chirinos y Ticona, 2003).

El problema obedece a la siguiente formulación matemática, la cual es un Problema de Programación Lineal Entera con 12 variables de tipo entero:

Minimizar $0.5x_{1,1} + 0.6x_{2,1} + 1.2x_{3,1} + 0.55x_{1,2} + 0.6x_{2,2} + 0.9x_{3,2} +$

$$+0.6x_{1,3} + 0.5x_{2,3} + x_{3,3} + 0.65x_{1,4} + 0.55x_{2,4} + 0.8x_{3,4}$$

Sujeto a:

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 200$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 62$$

$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 38$$

$$\tfrac{1}{8}x_{1,1} + \tfrac{1}{12}x_{2,1} + \tfrac{1}{5}x_{3,1} \le 14$$

$$\tfrac{1}{10}x_{1,2} + \tfrac{1}{11}x_{2,2} + \tfrac{1}{2}x_{3,2} \le 16$$

$$\tfrac{1}{9}x_{1,3} + \tfrac{1}{10}x_{2,3} + \tfrac{1}{3}x_{3,3} \le 12$$

$$\tfrac{1}{14}x_{1,4} + \tfrac{1}{11}x_{2,4} + \tfrac{1}{2}x_{3,4} \le 10$$

$$\tfrac{1}{8}x_{1,1} + \tfrac{1}{12}x_{2,1} + \tfrac{1}{5}x_{3,1} + \tfrac{1}{10}x_{1,2} + \tfrac{1}{11}x_{2,2} + \tfrac{1}{2}x_{3,2} + \tfrac{1}{9}x_{1,3} + \tfrac{1}{10}x_{2,3} + \tfrac{1}{3}x_{3,3} + \tfrac{1}{14}x_{1,4} + \tfrac{1}{11}x_{2,4} + \tfrac{1}{2}x_{3,4} \le 40$$

$$x_{1,1}, x_{2,1}, x_{3,1}, x_{1,2}, x_{2,2}, x_{3,2}, x_{1,3}, x_{2,3}, x_{3,3}, x_{1,4}, x_{2,4}, x_{3,4} \ge 0$$

Figura 3.20. Modelo Matemático del problema de aplicación 3.9

Para resolver este problema, primero estandarizamos e ingresamos la matriz A y los vectores b y c:

```
c=[0.5  0.6  1.2  0.55  0.6  0.9  0.6  0.5  1.0  0.65  0.55  0.8  0  0  0  0  0]'

A=[1     0     0    1    0     0    1    0     0   1    0    0   0   0   0   0   0
   0     1     0    0    1     0    0    1     0   0    1    0   0   0   0   0   0
   0     0     1    0    0     1    0    0     1   0    0    1   0   0   0   0   0
   1/8  1/12  1/5   0    0     0    0    0     0   0    0    0   1   0   0   0   0
   0     0     0  1/10  1/11  1/2   0    0     0   0    0    0   0   1   0   0   0
   0     0     0    0    0     0   1/9  1/10  1/3  0    0    0   0   0   1   0   0
   0     0     0    0    0     0    0    0     0   1/14 1/11 1/2 0   0   0   1   0
   1/8 1/12  1/5  1/10  1/11  1/2  1/9  1/10  1/3  1/14  1/11 1/2  0  0   0   0  1]

b=[200 62 38 14 16 12 10 40]'
```

Seguidamente hacemos una llamada al algoritmo *PLEPIPD.* La solución devuelta es la siguiente:

```
fx =

   85.0000
         0
   15.0000
  115.0000
         0
         0
         0
```

```
   62.0000
   17.0000
         0
         0
    6.0000
    0.3751
    4.4997
    0.1333
    7.0002
    0.0083
    0.0005
    0.0010
    0.0004
    0.0007
    0.0003
    0.0006

valor =

  176.5501

iter =

   147

cota =

  176.5501
```

El computador encuentra la solución óptima entera en un tiempo de 1 min.

La respuesta la interpretamos mediante la siguiente política óptima:

La distribución del trabajo entre los tipeadores es el siguiente: El primer trabajador debe hacer 85 páginas de texto, 0 tablas y 15 figuras. El segundo trabajador hará 115 páginas de texto, 0 tablas y 0 figuras. El tercer trabajador sólo hará 0 páginas de texto, 62 tablas y 17 figuras. Finalmente, el cuarto trabajador realizará 0 paginas, 0 tablas y tan sólo 6 figuras.

El costo que demanda esta política es de 176.55 soles. El cual, es el costo óptimo.

Problema 3.10: (Problema de Aplicación)

El siguiente problema (Producción de Aspersores), está formulado en (Greene y Knuth, 1982), aplicaremos el algoritmo *PLEPIPD* y mostraremos la solución entera óptima del problema que consiste en lo siguiente.

Un taller de montaje de aspersores consta de tres secciones *A*, *B* y *C*. En cada sección se pueden ensamblar tres tipos diferentes de aspersores *A-10*, *A-12* y *A-15*. La capacidad de producción de cada sección varía dependiendo del tipo de aspersores a ensamblar como indica la tabla 1. Acorde con una orientación, se necesita que por cada aspersor *A-10*, se fabrique un aspersor *A-12* y dos *A-15*. Determinar y resolver el modelo matemático que maximice el número de aspersores a fabricar.

Tabla 1. Sección de taller con sus capacidades de producción diaria de aspersores

SECCIÓN	CAPACIDAD DE PRODUCCIÓN DIARIA		
	A-10	A-12	A-15
A	85	85	70
B	75	65	55
C	75	70	80

Formulación estándar del modelo

Variables de decisión:

XA_{A-10} = Cantidad de aspersores *A*-10 a producir por día en la sección *A*.

XA_{A-12} = Cantidad de aspersores *A*-12 a producir por día en la sección *A*.

XA_{A-15} = Cantidad de aspersores *A*-15 a producir por día en la sección *A*.

XB_{A-10} = Cantidad de aspersores *A*-10 a producir por día en la sección *B*.

XB_{A-12} = Cantidad de aspersores *A*-12 a producir por día en la sección *B*.

XB_{A-15} = Cantidad de aspersores *A*-15 a producir por día en la sección *B*.

XC_{A-10} = Cantidad de aspersores *A*-10 a producir por día en la sección *C*.

XC_{A-12} = Cantidad de aspersores *A*-12 a producir por día en la sección *C*.

XC_{A-15} = Cantidad de aspersores *A*-15 a producir por día en la sección *C*.

Función objetivo:

Max Z= $XA_{A-10}+XA_{A-12}+XA_{A-15}+XB_{A-10}+XB_{A-12}+XB_{A-15}+XC_{A-10}+XC_{A-12}+XC_{A-15}$

Max$Z=$ –Mín $(-Z)$

Max(-Z)= $-[-XA_{A-10}-XA_{A-12}-XA_{A-15}-XB_{A-10}-XB_{A-12}-XB_{A-15}-XC_{A-10}-XC_{A-12}-XC_{A-15}]$

Restricciones de capacidad de producción

$$\frac{XA_{A-10}}{85}+\frac{XA_{A-12}}{85}+\frac{XA_{A-15}}{70}\leq 1$$

$$\frac{XB_{A-10}}{75}+\frac{XB_{A-12}}{65}+\frac{XB_{A-15}}{55}\leq 1$$

$$\frac{XC_{A-10}}{75}+\frac{XC_{A-12}}{70}+\frac{XC_{A-15}}{80}\leq 1$$

Otras restricciones adicionales:

$$XA_{A-10}+XB_{A-10}+XC_{A-10}-XA_{A-12}-XB_{A-12}-XC_{A-12}=0$$

$$XA_{A-10}+XB_{A-10}+XC_{A-10}-0.5XA_{A-15}-0.5XB_{A-15}-0.5XC_{A-15}=0$$

Restricciones de no negatividad:

$$XA_{A-10}, XA_{A-12}, XA_{A-15},\ XB_{A-10}, XB_{A-12}, XB_{A-15}, XC_{A-10}, XC_{A-12}, XA_{A-15}\geq 0$$

El problema sigue la siguiente formulación matemática en la figura 3.21

Max Z= $XA_{A-10}+XA_{A-12}+XA_{A-15}+XB_{A-10}+XB_{A-12}+XB_{A-15}+XC_{A-10}+XC_{A-12}+XC_{A-15}$

Sujeto a:

$$\frac{XA_{A-10}}{85}+\frac{XA_{A-12}}{85}+\frac{XA_{A-15}}{70}\leq 1$$

$$\frac{XB_{A-10}}{75}+\frac{XB_{A-12}}{65}+\frac{XB_{A-15}}{55}\leq 1$$

$$\frac{XC_{A-10}}{75}+\frac{XC_{A-12}}{70}+\frac{XC_{A-15}}{80}\leq 1$$

$$XA_{A-10}+XB_{A-10}+XC_{A-10}-XA_{A-12}-XB_{A-12}-XC_{A-12}=0$$

$$XA_{A-10}+XB_{A-10}+XC_{A-10}-0.5XA_{A-15}-0.5XB_{A-15}-0.5XC_{A-15}=0$$

$$XA_{A-10}, XA_{A-12}, XA_{A-15},\ XB_{A-10}, XB_{A-12}, XB_{A-15}, XC_{A-10}, XC_{A-12}, XA_{A-15}\geq 0$$

Figura 3.21. Modelo Matemático del problema de aplicación 3.10 (Producción de Aspersores)

Ahora, estandarizamos e ingresamos la matriz A y los vectores b y c:

```
c=[-1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0]'

A=[1/85 1/85 1/70  0     0     0   0       0     0    1    0  0
    0     0    0  1/75 1/65 1/55   0       0     0    0    1  0
    0     0    0   0     0     0   1/75  1/70  1/80   0    0  1
    1     -1   0   1     -1    0   1       -1     0    0    0  0
    1     0  -1/2  1     0   -1/2  1       0    -1/2    0    0  0]

b=[1 1 1 0 0]'
```

Seguidamente hacemos una llamada al algoritmo *PLEPIPD* y nos muestra la siguiente solución

```
fx =

    5.0000
   54.0000
   21.0000
   52.0000
    1.0000
   16.0000
         0
    2.0000
   77.0000
    0.0059
    0.0004
    0.0089
    3.0002
    5.0007
    1.0000
    0.0008
    2.0002
    0.0012
    1.0019
    1.0002
    1.0020
    0.0000
    0.0014
    1.0000
    0.0002
    0.0006
    0.0000
    0.0002
    0.0018
    0.0000


valor =

 -228.0056


iter =

       13343


cota =

 -228.0056
```

El computador encuentra la solución óptima entera en un tiempo de 2 min.

Por lo tanto el número de aspersores a fabricar es:

$$XA_{A-10} = 5 \qquad XB_{A-10} = 52 \qquad XC_{A-10} = 0$$
$$XA_{A-12} = 54 \qquad XB_{A-12} = 1 \qquad XC_{A-12} = 2$$
$$XA_{A-15} = 21 \qquad XB_{A-15} = 16 \qquad XC_{A-15} = 77$$

Valor de la función objetivo es: $\text{Max } Z = -\text{Mín } (-Z) = 228$.

Problema 3.11: (Problema de Aplicación)

(Problema de Producción de Helados)

El siguiente problema esta formulado como un problema de programación lineal con variable continua (Calvo, 2013), allí se resolvió y se obtuvo una solución óptima que incluía componentes no enteras, con valor objetivo óptimo 876,3750 u.m.

Gama Gourmet

Ahora nosotros vamos a aplicar *PLEPIPD* para encontrar la solución óptima con todas las variables enteras.

El problema consiste sobre la producción de una gama de helados que podemos considerar "Gourmet", está conformada por cinco tipos de polos artesanos: Polos de menta, Polos de

chocolate, Polos de yogurt y melocotón, Polos de almendras y Polos “Fiordilatte”, el cual la empresa ha solicitado que se determine el plan semanal de producción de los diferentes tipos de polos que conforman la “Gamma Gourmet”, con el objetivo de maximizar beneficios.

El problema propuesto por la empresa se corresponde con un problema de programación lineal. El cuál, se definen las distintas actividades a realizar (variables de decisión), las restricciones del problema original por las limitaciones de los recursos (materias primas) y las demandas a satisfacer, así como la función objetivo (beneficios) que se tiene que maximizar.

Variables de decisión:

x_1 = n° kilos a fabricar semanalmente de polos de menta

x_2 = n° kilos a fabricar semanalmente de polos de chocolate

x_3 = n° kilos a fabricar semanalmente de polos de yogurt y melocotón

x_4 = n° kilos a fabricar semanalmente de polos de almendras

x_5 = n° kilos a fabricar semanalmente de polos fiordilatte

Restricciones con respecto a la disponibilidad de materias primas:

- Limitación de Jarabe Base: 20,5 kilos

$$550x_1 + 500x_2 + 430x_3 + 400x_4 + 510x_5 \leq 20500$$

- Limitación de Leche Fresca Entera: 13 kilos

$$550x_1 + 500x_2 + 430x_3 + 400x_4 + 510x_5 \leq 20500$$

- Limitación de Yogurt desnatado: 5 kilos

$$300x_3 \leq 5000$$

- Limitaciones de Nata: 8,5 kilos

$$500x_3 + 550x_5 \leq 8500$$

- Limitaciones de azúcar invertido: 1,3 kilos

$$25x_2 + 20x_3 + 25x_4 + 200x_5 \leq 1300$$

- Limitaciones de Chocolate: 27 kilos

$$800x_1 + 800x_2 + 500x_3 + 800x_4 + 1300x_5 \leq 27000$$

- Limitaciones de Manteca de cacao: 5 kilos

$$200x_1 + 200x_2 + 200x_4 + 200x_5 \leq 5000$$

- Limitaciones de Esencia de menta: 150 gotas

$$10x_1 \leq 150$$

- Limitaciones de Cacao: 0,28 kg

$$35x_2 \leq 280$$

- Limitación de Melocotón Batido: 4 kilos

$$250x_3 \leq 4000$$

- Limitación de Pasta de Almendras: 0,8 kilo

$$80x_4 \leq 800$$

- Restricciones con respecto a la demanda:

$$x_1 + x_2 \geq 10$$
$$x_3 + x_4 \geq 10$$

- Función Objetivo: Beneficio

$$23x_1 + 22.5x_2 + 21x_3 + 20.5x_4 + 21x_5$$

Por lo tanto, la formulación del problema es la siguiente figura 3.22

$$Max \;\; Z = 23x_1 + 22.5x_2 + 21x_3 + 20.5x_4 + 21x_5$$

Sujeto a :

$$550x_1 + 500x_2 + 430x_3 + 400x_4 + 510x_5 \le 20500$$
$$450x_1 + 440x_2 + 495x_4 + 510x_5 \le 13000$$
$$300x_3 \le 5000$$
$$500x_3 + 550x_5 \le 8500$$
$$25x_2 + 20x_3 + 25x_4 + 200x_5 \le 1300$$
$$800x_1 + 800x_2 + 500x_3 + 800x_4 + 1300x_5 \le 27000$$
$$200x_1 + 200x_2 + 200x_4 + 200x_5 \le 5000$$
$$10x_1 \le 150$$
$$35x_2 \le 280$$
$$250x_3 \le 4000$$
$$80x_4 \le 800$$
$$x_1 + x_2 \ge 10$$
$$x_3 + x_4 \ge 10$$
$$x_1, x_2, x_3, x_4, x_5 \ge 10$$

Figura 3.22. Modelo matemático del problema de aplicación 3.11 (Producción de Helados)

Veamos que sucede al aplicar los algoritmos *PIPD* y *PIPC* para programación lineal, luego procedemos con la aplicación del algoritmo *PLEPIPD,* consideremos $\text{Max } Z = -\text{Mín}\,(-Z)$.

Una llamada al algoritmo *PIPD* tenemos:

```
x =
  1.0e+003 *
    0.0150
    0.0080
    0.0160
    0.0008
    0.0000
    1.0701
    2.3587
    0.2000
    0.5000
    0.7612
    0.0000
    0.2500
    0.0000
    0.0000
    0.0000
    0.7400
    0.0130
    0.0068
v =
 -876.3722
iter =
    12
```

Una llamada al algoritmo *PIPC* tenemos:

```
x =
  1.0e+003 *
    0.0150
    0.0080
    0.0160
    0.0008
    0.0000
    1.0700
    2.3587
    0.2000
    0.5001
    0.7613
    0.0000
    0.2500
    0.0000
    0.0000
    0.0000
    0.7400
    0.0130
    0.0067
v =
 -876.3728
iter =
    13
```

Entonces de ambos resultados, la empresa obtiene un beneficio máximo de 876.37 u.m., produciendo semanalmente 15 kilos de polos de menta, 8 kilos de polos de chocolate, 16 kilos de yogurt y melocotón y 0,8 kilos de polos de almendra. Aquí no se contempla la fabricación de polos de Fiordilatte.

Ahora al aplicar el algoritmo *PLEPIPD* nos proporciona la siguiente solución:

```
fx =
  1.0e+003 *
    0.0138
    0.0080
    0.0159
    0.0020
    0.0000
    1.2644
    2.2941
    0.2134
    0.5161
    0.7289
    0.0031
    0.2465
    0.0122
    0.0000
    0.0112
    0.6413
    0.0118
    0.0079
    0.0010
    0.0000
valor =
```

```
   Inf
iter =
   155
cota =
   Inf
```

Entonces, en la iteración 155, el algoritmo detectó que el problema *PLE* es infactible. Esto quiere decir el algoritmo no realiza una sola iteración para detectar infactibilidad, se debe a que la relajación del problema, en cada iteración, si posee una solución óptima no entera. El algoritmo *PLEPIPD* bifurca la región de factibilidad buscando una solución entera óptima, el cual es ilimitada tal solución y no es alcanzada.

Cuando pedimos al algoritmo *PLEPIPD* que nos muestre las cuatro primeras variables enteras tenemos:

```
fx =
  1.0e+003 *
    0.0150
    0.0070
    0.0160
    0.0010
    0.0005
    1.2345
    2.4396
    0.2000
    0.2464
    0.6877
    0.0000
    0.3077
    0.0000
    0.0350
    0.0000
    0.7200
    0.0120
    0.0070
    0.0000
    0.0000
    0.0000
valor =
 -868.6976
iter =
    17
cota =
 -868.6976
```

Aquí podríamos decir que se aproxima a la solución donde la producción semanal de las cuatro primeras variables sería: 15 kilos de polos de menta, 7 kilos de polos de chocolate, 16 kilos de yogurt y melocotón y 1 kilos de polos de almendra, pero en la función objetivo considera la quinta variable 0,5 kilos (de polos de Fiordilatte) que no es entera, el cual nos otorga un beneficio máximo de 868.6976 u.m., es decir calcula el problema de relajación,

Recordemos que la estrategia para resolver el problema de programación lineal entera, por lo general consiste en resolver este problema como si fuera un problema de programación lineal en variable continua, sin considerar la condición que x sea entera. Este procedimiento recibe el nombre de Relajación del problema de Programación Lineal Entera.

En este problema de producción de helados fue formulado como un problema de programación lineal donde sus variables de decisión son reales positivas. Ahora si en el modelo lineal, las variables de decisión deben tomar valores enteros, entonces puede utilizarse la Programación Lineal Entera. (Calvo, 2013)

Conclusiones

En el presente trabajo “Implementación Computacional de Métodos para la Resolución de Problemas de Programación Lineal Entera”, hemos tratado de dos métodos *PLEPIPD* y *PLEPIPC* para resolver el Problema de Programación Lineal Entera, el cual se fundamenta en el primer método, de las técnica de Ramificación y Acotamiento (Branch and Bound) y el Método de Puntos Interiores Primal-Dual, y en el segundo método, también de las técnicas de Ramificación y Acotamiento y el método de Puntos Interiores Predictor-Corrector.

Los métodos *PLEPIPD* y *PLEPIPC* son similares al adoptado por los métodos clásicos, la diferencia radica en que los métodos anteriores usaron como una subrutina el Algoritmo Simplex, mientras que nosotros adoptamos los Algoritmos de Puntos Interiores Primal-Dual (*PIPD*) y Puntos Interiores Predictor-Corrector (*PIPC*). La razón de la elección de estos algoritmos *PIPD* y *PIPC* es que han mostrado ser los algoritmos más eficientes para resolver el problema de programación lineal (*PL*) hoy en día.

Además, en este trabajo nosotros hemos incluido en pseudo-código, el algoritmo principal respectivo con todas las subrutinas necesarias. Más aún, incluimos los códigos necesarios de modo que puedan ser implementados fácilmente en cualquier versión moderna de MatLab.

Debemos observar que el algoritmo principal responsable de la ramificación es sumamente complicado, ya que está basado en una técnica *recursiva*.

Varios ejemplares de *PLE* fueron incluidos en los experimentos computacionales, intentando mostrar los diferentes inconvenientes con los que podría encontrarse un método de esta naturaleza. Los resultados fueron registrados e ilustrados.

Internamente los métodos propuestos por nosotros, a medida que los problemas comienzan a ramificarse, obtenemos nuevos subproblemas, y la matriz A crece debido a la adición de nuevas restricciones, provocando que aparezcan nuevas variables de holgura y/o exceso. Todo esto constituye un inconveniente crítico en todos los métodos de programación lineal entera, ocasionando que el tamaño del ejemplar aumente considerablemente, a medida que las iteraciones aumentan. Pero los problemas de *PLE* resultan ser complicados por lo mismo que son NP-Completos, y son los problemas más difíciles de la Clase *NP*. Es por eso

que no se conocen algoritmos “eficientes” determinísticos que resuelvan el problema PLE en tiempo polinomial.

Podemos concluir lo siguiente:

Hemos tratado de dos métodos para resolver el Problema de Programación Lineal Entera basado en el Método de Puntos Interiores Primal-Dual y el Método de Puntos Interiores Predictor-Corrector, estos métodos son teóricamente superiores a cualquier método clásico encontrado en la literatura.

El Problema de Programación Lineal Entera es por naturaleza difícil de resolver, debido a que este problema pertenece a la clase NP-Completo.

El Problema de Programación Lineal (PL) y el Problema de Programación Lineal Entera (PLE) son similares apenas en sus estructuras, pero totalmente diferentes en el aspecto computacional, debido a que PL está en la clase P.

Los tiempos de ejecución de los algoritmos *PLEPIPD* y *PLEPIPC* están en $O(2^H)$ y no depende del computador en que trabajamos ni del lenguaje de programación que utilicemos, la única diferenciará con respecto a otro computador u otra implementación en algún lenguaje será apenas por una constante multiplicativa.

Claramente, si usáramos un lenguaje de programación, tal como Delphi o el C++, el tiempo de ejecución será menor a los que obtuvimos usando MatLab, ya que este último es un interpretador. Pero, en virtud del *principio de invariabilidad*, la única diferencia entre un tiempo de ejecución y otro radicará apenas en una constante multiplicativa.

Por último, notamos que los algoritmos PLEPIPD y PLEPIPC trabajan bajo la hipótesis que las primeras *nv* variables del vector *x* sean enteras, estos fueron mostrados en los experimentos computacionales.

Bibliografía

Adler I., Karmarkar N., Resende M.G.C. and Veiga G. (1989). An implementation of Karmarkar´s algorithm for linear programming. *Mathematical Programming*, vol 44, pp. 279-335

Avila J.F. (1995). Método de Karmarkar, Revista de Matemática: Teoría y Aplicaciones 2(1): 45-55.

Armitano O., Edelman J. y Palomares G. (1985). *Programación no lineal*. Editorial Limusa, S. A. de C.V. México.

Arbel A. (1993). Exploring Interior-Point Linear Programming. The MIT Press.

Allgower E. & Georg K.(1980). *Simplicial and Continuation Methods for Approximating Fixed Points and Solutions to Systems of Equations.* Siam Review 22. pp. 28-85.

Anstreicher, K.M. (1990). *Progress in interior point algorithms since 1984,* SIAM News 3.

Bazaraa M. S. and Shetty C.M. (1970). *Nonlinear Programming: Theory and algorithms*, Toronto: Wiley.

Bazaraa M. S., Jarvis J. J., & Sherali H. D. (1990) , *Linear Programming and Network Flows,* Second Edition. John Wiley & Sons.

Bazaraa M., Sherali H. and Shetty C.(1993). *Nonlinear Programming*, John Wiley&Sons.

Brassard, Gilles & Bratley P. (1988), *Algorithms: Theory and Practice*, Pretice-Hall.

Barnhart Cyntia, Johnson Ellis L., Nemhauser George L., Savelsbergh Martin W.P., Vance Pamela H. (1996). *Branch-and-Price: Column Generation for Solving Huge Integer Programs.* School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta.

Castro J. (1998). Implementación de un Algoritmo Primal-Dual de orden superior mediante el uso de un Método Predictor-Corrector para Programación Lineal, *Universitat Rovira i Virgili*, QÜESTIIÓ, vol. 22, pp. 103-116.

Chirinos Y.S. & Ticona P. (2003). *Programación Lineal Entera Vía El Método de Puntos Interiores Primal-Dual.* (Tesis de posgrado). Universidad Nacional de San Agustín, Arequipa, Perú.

Chen, D. S., Batson R., y Dang Y. (2010). Applied Integer Programming: Modeling and Solutions. *Wiley*, Nueva York.

Calvo de León, E. C. (2013). Programación Lineal: Aplicación a la Producción de Helados, Sevilla, España.

Dantzing G. (1963). Linear Programming and extensions. *Princeton, NJ: Priceton Unive Press.*

Derpich, I., Vera, A.(2016). *Improving the efficiency of the Branch-and-Bound algorithm for integer programming based on "flatness" informations,* European Journal of Operational Reseach, 174(I1): 92-101.

David G., Espitia G. y Monroy H. (2016*).* Ambiente visual para el aprendizaje de los conceptos básicos asociados a la recurrencia. Escuela Colombiana de Ingeniería Julio Garavito. Bogotá D.C – Colombia.

Greene, D. H., Knuth, D. E. (1982). *Mathematics for the Analysis of Algorithms,* Boston; Basel; Stuttgart: Birkhäuser.

Soria M. D., Redchuck A. (2000). Un Método Predictor-Corrector para Programación Cuadrática, II RIMO – Reunión de Investigación Militar Operativa, Madrid España.

Edgar M., Carreño F., Eliana M., Toro O. & Antonio Escobar Z. (2004). *Optimización de Sistemas Lineales usando Métodos de Punto Interior,* Universidad Tecnológica de Pereira.

Fiacco A.V. & Mcormick G. P. (1968). Nonlinear programming: sequential unconstrained minimization techniques. *Jhon Wiley & Sons*.

Gomory, R. (1963). *An algorithm for integer solutions to linear programs. In Recent Advances in Mathematical Programming.* NY, McGraw-Hill, pp. 269-302.

Graves, R., Schrage L., y Sankaran J. (1993). *An Auction Method for Course Registration. Interfaces,* vol. 23, núm. 5, 1993, pp. 81-97.

Guéret, C., Prins, C. y Sevaux M. (2002). Applications of Optimizations with Xpress-MP, Dash Optimization, Londres.

Granville S. (1994). *Optimal reactive dispatch through interior point methods*. IEEE Transactions on Power System, vol 9, pp. 136-146.

Gill P., Murray W. , Saunders M., Tomlin J. & Wright M. (1985). *On Projected Newton Barrier Method for Linear Programming and an Equivalence to Karmarkar´s Projective Method,* Systems Optimization Laboratory, Stanford University, Technical Report SOL 85-101.

Hiller, F., and Lieberman, G.(2001), *Introduction to Operations Research,* 7th Edition, New York, NY, McGraw-Hill. pp. 978

Jarvis, J., Rardin R., Unger V., Moore R. y C. Schimpler C. (1978). *Optimal desing of Regional Wastewater Systems: A Fixed Charge Network Flow Model.* Operations Research. vol. 26, núm. 4, pp. 538-550.

Karmarkar, N. (1984). *A new polinomial-time algorithm for linear programming, Combinatorica,* vol 4, núm.4, pp. 373-395.

Luenberger D. G. (1984). *Linear and Nonlinear Programming,* Addisson-Wesley, Reading, Massachusetts, second ed.

Lee, J., *A First Course in Combinatorial Optimization,* Cambridge University Press, 2004. Liberatore, M., and T. Miller, "A Hierarchical Production Planning System" *Interfaces,* vol. 15, núm. 4, 1985, pp. 1-11.

Lecca E.R., Lizama E. R. (2007). La geometría del método Simplex y sus aplicaciones utilizando Matlab, Departamento Académico de Sistemas e Informática, UNMSM.

López L. H. (2008), *Complejidad de Algoritmos y Problemas,* Ingeniería Industrial, Universidad de Chile.

Mehrotra, S. (1992), *On the Implementation of a Primal-Dual Interior Point Method,* SIAM Journal on Optimization, 2(4), 575-601

Molina P. D. y Cabrera E.E. (2014), *Programación entera para modelos lineales.* Ingeniería Hidráulica y Ambiental, Vol. XXXV. Nro. 1, pp. 62-76.

M.C. Ferris M.C., Mangasarian O.L., Wright S.J. (2007). *Linear Programming with MATLAB.* MPS-SIAM Series on Optimization.

Megiddo, N. (1986). *Pathways to the Optimal Set in Linear Programming. in Proc. of the Mathematical Programming Symposium of Japan.* Nagoya, Japan. pp. 1-36. (See also *Progress in Mathematical Programming-Interior-Point and Related Methods.* Nimrod Megiddo (Ed.). Springer-Verlag. pp. 131-158. 1989.)

Nemhauser, G y Wolsey L. (1988) Integer and Combinatorial Optimization.. *Wiley.* Nueva York.

Papadimitriou, Ch. H. y Kenneth S. (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall.

Papadimitriou CH., *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA. 1994.

Przemyslaw B.(2015). *Computational Complexity of Solving Equation Systems.*Springer Cham Heidelberg New York Dordrecht London.

Padberg, M. and Rinaldi, G. (1991), *A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems,* SIAM Rev., 33(1): 60-100.

Ryan D.M. and Foster B.A. (1981). *An integer programming approach to scheduling,* A. Wren (ed.) Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, North-Holland, Amsterdam, 269-280.

Rodríguez Rojas W. J. (2005). *Conceptos y ejemplos básicos de Programación Dinámica.* Fundación Universidad Konrad Lorenz, Facultad de Matemáticas, Bogotá D.C.

Rodriguez, Crespo y Jiménez. (2015). *Proyecto Alfombra de Sierpinski,* Universidad de Almería, España. Recuperado de: http://17jaem.semrm.com/aportaciones/n122.pdf

Salahi M., Peng J. and Terlaky T. (2005). On Mehrotra-Type Predictor-Corrector Algorithms.

Schrijver, A. (1998). *Theory of Linear and Integer Programming.* Wiley. Nueva York.

Seebach Cl. (2006), Condiciones de Karush Kuhn Tucker*, Pontificia Universidad Católica, Escuela de Ingeniería. Departamento de Ingenniería de Sistemas*.

Taha H. (2012). *Investigación de Operaciones*, Prentice Hall, 9º Edición.

Taha, H. (1975). *Integer Programming: Theory, Applications, and Computations,* Academic Press, Orlando, FL.

Torres G.L., Quintana V.H. (1998). *An interior point method for nonlinear optimal power flow using voltage rectangular coordinates.* IEEE Transactions on Power Systems, vol 13, pp. 1211-1218.

Vanelli, A. (1993). *Teaching Large Scale Optimization by an Interior Point Approach. IEEE trans on Education*, vol 36, núm. 1, pp. 204-209.

Vanderbei R. J. *Linear Programming: foundations and extensions,* Second Editions, http://www.princeton.edu/~rvdb/LPbook/,Dec2000.

Valuenzuela V. (2003). *Manual Análisis de Algoritmos, Versión 1.0.* Ingenieria Informática INACAP, Copiapó, Chile.

Wright, S.J. (1996). Primal-Dual Interior-Point Methods. *SIAM Publications*. Philadelphia.

Wolsey, L.(1998). Integer Programing. *Wiley*, Nueva York,

Wu Y. Ch., Debs A. S., Marsten R.E. (Feb. 1994). *A direct predictor-corrector primal dual interior point algorithm for optimal power flows.* IEEE Transactions on Power system, vol 9, pp. 876-883.

YoonAnn, Eun-Mee . (2006), *Branch and Bound Algorithm for Binary Quadratic Programming with Application in Wireless Network Communications.* M.S. in Applied Mathematics. Ames, lowa State University, College of Liberal Arts and Sciences, 89p.

Ye, Y. (1990). *Recovering optimal basic variables in Karmarkar polynomial algorithm for linear programming.* Mathematics of Operation Research 3.

Ye, Y (1997). *Interior Point Algorithms Theory and Analysis.* Wiley-Interscience series in Discrete Mathematics and Optimization, New York.