UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

Departamento de Matemática

Tesis de Licenciatura

ALGORITMOS PROYECTIVOS DE SEPARACIÓN PARA PROBLEMAS DE PROGRAMACIÓN LINEAL ENTERA

Federico Rodes

Directora: Paula Zabala
Co-directora: Isabel Méndez-Díaz

Octubre de 2011

Agradecimientos

A mis directoras, Isabel y Paula, por la paciencia infinita durante estos dos años de trabajo.

A los jurados, Javier Echeverry y Guillermo Durán.

A Gabriel Acosta, Javier Marenco e Ignacio Ojea, por tomarse el tiempo de leer los primeros bocetos de la tesis.

A toda mi familia (incluyendo a los adoptivos), sin la cual esto no hubiera sido posible.

A la familia Laborde por abrirme las puertas, y en especial a la señorita Cecilia por el aguante (mutuo) incondicional.

A Bruno, Caro, Ceci Picchio, Cele, Croxy, Dardo, Dario, Delpe, Diego Castro, Droopy, El colo, El Garza, El Pocho, Etchevarne, Eze Cura, Ciccioli, Florcita, Igna, Inés, Irina, JP, Juancito, Juan Domingo, Lucas, Lucio, Luigi, Magui, Mariana, Maurette, Mauro, Nico Michan, Nico Sirolli, Pedersen, René, Sasa, Tatiana, Topa, Valdettaro, Vicky, Zeque, más todos los que me olvido, por hacer que el día a día en Ciudad fuera algo especial.

A Eduardo Palma, por las tardes y tardes ayudándome a corregir la tesis!

A todos los docentes y no docentes del departamento de matematica.

Resumen

En esta tesis nos concentraremos sobre *Problemas de Programación Lineal Entera* (PPLE) y, en especial, sobre los métodos *exactos* utilizados para su resolución.

Los PPLE son *problemas de optimización* con las siguientes características distintivas: la función que se pretende maximizar (o minimizar) es una función lineal; el dominio sobre el que se debe trabajar queda determinado por la intersección de un conjunto de desigualdades lineales; y, por último, al menos una de las variables en juego debe estar obligada a adquirir valores enteros. El interés por estudiar estas estructuras surge como consecuencia del gran número de situaciones reales que permiten representar. Típicamente, los modelos de programación lineal entera son utilizados para describir procesos industriales y actividades del sector *servicios* con el fin de minimizar los costos de producción y logística.

En cuanto a los métodos de resolución, hasta el momento no se conoce ningún algoritmo eficiente –de tiempo polinomial– que permita hallar la solución óptima para cualquier instancia. Es decir, los PPLE pertenecen a la clase *NP-Hard* [12]. Por dicho motivo, desde el surgimiento de esta rama de la matemática en la década del 50', se han ensayado distintas estrategias para abordar este tipo de problemas. Podemos agruparlas en tres categorias: *métodos exactos*, *métodos aproximados* y *métodos heurísticos*.

- **Métodos exactos**: son procedimientos que garantizan la obtención del óptimo. Si bien la pertenencia a la clase *NP-Hard* indica que el tiempo requerido para encontrar la solución óptima puede resultar prohibitivo, los algoritmos exactos no son dejados de lado. La posibilidad de resolver en forma exacta instancias cada vez más grandes aumenta debido al desarrollo de mejores algoritmos y a la aparición de nuevas tecnologías.

- **Métodos aproximados**: esta clase de procedimientos permite encontrar una *solución factible* del problema (es decir, una solución que satisfaga el conjunto de restricciones lineales, aunque no sea la mejor) y, al mismo tiempo, estimar la brecha entre esa solución y la solución óptima (certificado de optimalidad). Consumen menos recursos que los algoritmos exactos.

- **Métodos heurísticos**: son procedimientos que permiten hallar una solución factible del problema, pero son incapaces de determinar cuán cerca está esa solución de la solución óptima. Utilizan la menor cantidad de recursos y suelen ser una muy buena alternativa para instancias donde los algoritmos exactos no son adecuados.

Dentro de la primera familia de métodos, las técnicas *Branch-and-Bound* y *Branch-and-Cut* –basadas en la teoría poliedral– han demostrado ser una de las mejores herramientas para tratar este tipo de problemas. Una de las características más destacables de estos algoritmos es su *flexibilidad*, la cual les permite adaptarse a las distintas formulaciones y, de esa manera, explotar características propias de cada problema (ver por ejemplo: *Problema del Viajante* [1], *Ruteo de Vehículos* [14] y *Orden Lineal* [17]). En un contexto más general, podemos destacar a los *solvers*

CPLEX [20], XPRESS-MP [9] y GuroBi [16] como algunos de los paquetes académico-comerciales que ofrecen las mejores implementaciones de las técnicas anteriormente citadas.

En esta tesis propondremos un nuevo método exacto que, utilizando *proyecciones* para determinar la solución óptima, pueda ser aplicado sobre cualquier PPLE. Si bien la actual implementación del método puede considerarse un prototipo sobre el cual hay espacio para introducir muchas mejoras, los resultados computacionales, comparados con aquellos obtenidos con paquetes académico-comerciales, son altamente satisfactorios. La experiencia computacional nos demuestra que la propuesta es válida y que aporta una nueva visión dentro de la clase de métodos exactos.

El trabajo está organizado de la siguiente manera: en el capítulo 1, introducimos los conceptos básicos de la programación lineal entera y describimos los principales algoritmos usados para su resolución. En el capítulo 2, presentamos dos casos particulares de problemas de programación lineal entera: el *Problema de la Mochila No Acotado* (UKP) y el *Problema de la Mochila Multidimensional* (MKP). En el capítulo 3, motivamos el uso de proyecciones para determinar soluciones enteras y proponemos nuestro algoritmo. El comportamiento del algoritmo es analizado para los problemas UKP y MKP en los capítulos 4 y 5 respectivamente. Finalmente, en el capítulo 6, formulamos nuestras conclusiones y futuras líneas de trabajo.

Índice general

Capítulo 1

Conceptos Básicos

La finalidad de este capítulo es hacer una repaso –sin entrar en detalles– de los principales conceptos y herramientas asociados a programación lineal entera. El lector interesado puede consultar [37] y [29].

Comenzaremos el capítulo describiendo en qué consiste un problema de programación lineal y cómo podemos hallar su solución a través del método *Simplex*. A continuación, explicaremos qué es un problema de programación lineal entera y cuál es la estrategia de los algoritmos de *Planos de Corte*, *Branch-and-Bound* y *Branch-and-Cut* utilizados para su resolución.

1.1. Programación Lineal y el Método *Simplex*

Un *Problema de Programación Lineal* (PPL) es un problema de optimización donde la función que se pretende maximizar (o minimizar) es una función lineal y el dominio sobre el que se debe trabajar queda determinado por la intersección de un conjunto de desigualdades e igualdades lineales. Luego, cualquier PPL siempre puede ser reformulado de la siguiente manera (*forma Standard*):

$$\begin{aligned} \text{maximizar} \quad & f = c_1x_1 + \cdots + c_nx_n \quad \longleftarrow \text{ función objetivo} \\ \text{sujeta a} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ & \qquad\qquad\qquad\qquad \vdots \qquad \} \text{ conjunto de restricciones lineales} \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\ & x_i \geq 0,\ x_i \in \mathbb{R}\ \forall i = 1, \ldots, n \end{aligned}$$

George B. Dantzig propuso el modelo de programación lineal en junio de 1947. Su intención era representar, de una manera formal y unificada, un conjunto de problemas de planificación surgidos a partir de actividades militares. Por aquella época, Dantzig formaba parte de un proyecto de investigación destinado a mecanizar los procesos de planificación de la Fuerza Aérea estadounidense, el proyecto SCOOP (Scientific Computation of Optimum Programs). De esta manera, se dio inicio a una de las ramas de la matemática con mayor número de aplicaciones prácticas.

Pocos meses después de formular el modelo, Dantzig publicó un procedimiento para resolver problemas de programación lineal: *el método Simplex*. La idea que hay detrás del *Simplex* es muy sencilla, se basa en la siguiente propiedad: *en todos los PPL –cualquiera sea el número de variables que estos contengan–, si existe el óptimo, siempre se ubica en alguno de los vértices de la región determinada por el conjunto de desigualdades lineales*. Esta propiedad tiene dos consecuencias: por un lado, reduce a un número finito la cantidad de puntos a analizar (a partir de ahora sólo tenemos que comparar el valor de la función en los vértices de la región) y, por otro lado, nos indica dónde debemos buscar la solución óptima.

El *Simplex* es un algoritmo que consta de dos etapas. En la primera, busca una solución factible coincidente con alguno de los vértices de la región (si esto no fuese posible, el problema en cuestión es infactible). En la etapa siguiente, y a partir de la solución hallada en la fase anterior, el algoritmo pasa, en cada iteración, de un vértice del poliedro a otro vértice adyacente aumentando el valor de la función. Una vez arribado al óptimo, como no puede alcanzar una ulterior mejora, se detiene.

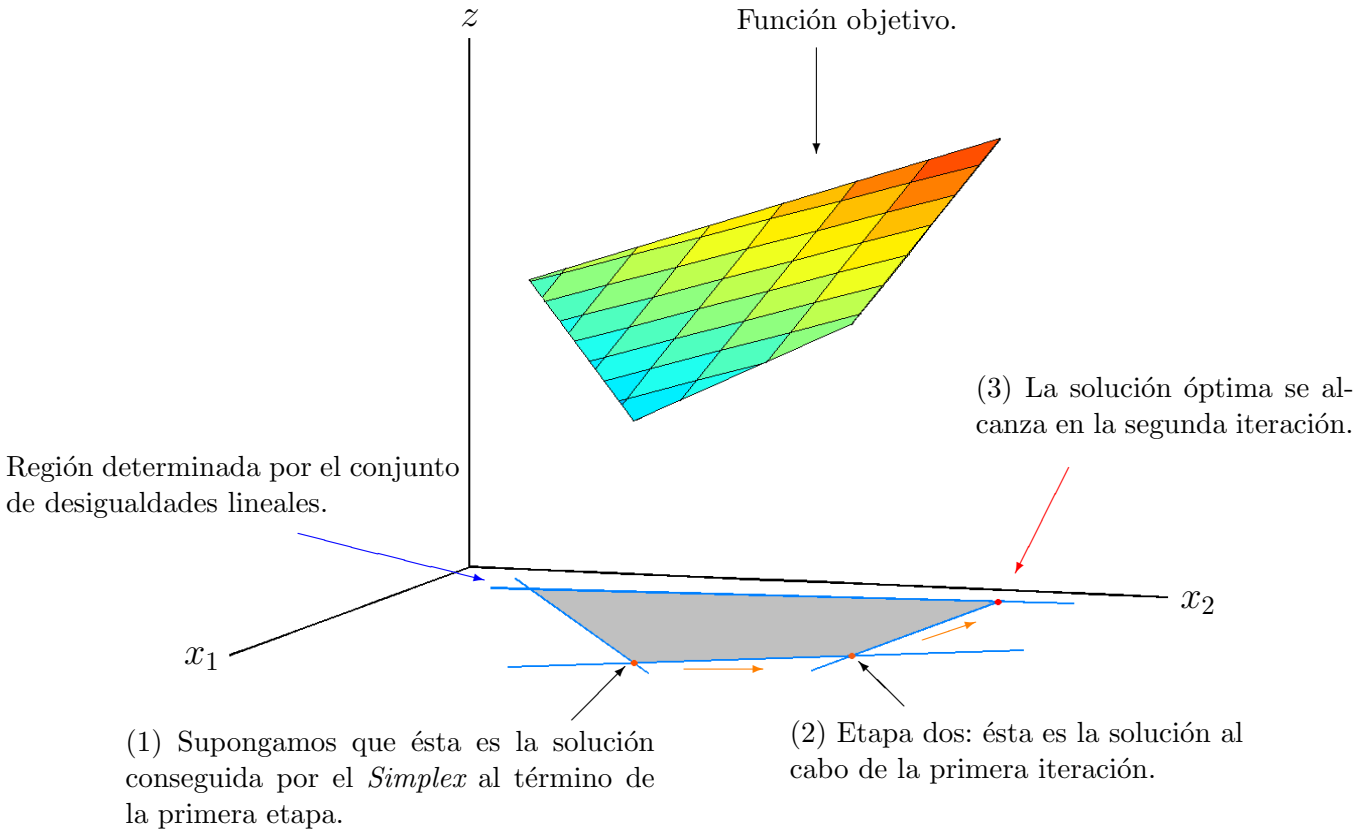

Figura 1.1: el método *Simplex*.

Si bien la cantidad de vértices de un poliedro es finita y, por lo tanto, el algoritmo de Dantzig siempre termina después de un número finito de pasos, la cantidad de iteraciones puede resultar *muy grande*. En 1972, Klee y Minty [25] encontraron ejemplos de n variables y $2n$ restricciones para los cuales el *Simplex* necesita 2^n iteraciones para hallar la solución. Demostraron, de esta manera, que no se trata de un algoritmo polinomial (si bien en la práctica su performance es muy buena). Posteriormente, en 1979, Khachiyan [23] diseñó un algoritmo que resuelve todos

los PPL en tiempo polinomial; aunque en la práctica resulta ser más lento que el *Simplex*. Karmarkar [22], en 1985, combinó las características de ambos métodos para crear un algoritmo eficiente y de tiempo polinomial, dando origen a la familia de algoritmos de "*Punto Interior*". En la actualidad, todos los paquetes académico-comerciales cuentan con implementaciones muy eficientes tanto de *Simplex* como de alguna variante de *Punto Interior*.

1.2. Programación Lineal Entera

Después del trabajo inicial de Dantzig, y a medida que empezaron a surgir cada vez más aplicaciones (sobre todo fuera del ámbito militar), también comenzaron a aparecer extensiones de la programación lineal. Los *Problemas de Programación Lineal Entera* (PPLE) son problemas de programación lineal en los cuales, al menos una de las variables incluidas en el modelo, debe adquirir valores enteros. Mediante esta nueva estructura se pueden representar muchas situaciones reales que hasta ese momento no podían ser abordadas por modelos de programación lineal. Por ejemplo, situaciones donde se involucran entidades indivisibles –como aviones, barcos o personas– en las que no tendría sentido dar como respuesta una solución con valores fraccionarios.

Dentro de la programación lineal entera se distinguen tres categorías de problemas; se agrupan de acuerdo a la cantidad de variables enteras que contengan y a la libertad que se les otorga a esas variables.

1. **Problemas enteros puros**: se trata de PPL donde <u>todas</u> las variables deben asumir valores enteros. Su formulación es la siguiente:

$$\begin{aligned} \text{maximizar} \quad & f = c_1x_1 + \cdots + c_nx_n \\ \text{sujeta a} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\ & x_i \geq 0,\ \underline{x_i \in \mathbb{Z}},\ \forall i = 1, \ldots, n \end{aligned}$$

2. **Problemas enteros binarios**: son PPL en los que todas las variables deben ser enteras y, además, sólo pueden valer 0 o 1. Su formulación es la siguiente:

$$\begin{aligned} \text{maximizar} \quad & f = c_1x_1 + \cdots + c_nx_n \\ \text{sujeta a} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\ & x_i \geq 0,\ \underline{x_i \in \{0,1\}},\ \forall i = 1, \ldots, n \end{aligned}$$

3. **Problemas enteros mixtos**: se trata de PPL en los cuales un grupo de variables –no todas– deben asumir valores enteros. Su formulación es la siguiente:

$$\begin{aligned} \text{maximizar} \quad & f = c_1x_1 + \cdots + c_nx_n \\ \text{sujeta a} \quad & a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\ & x_i \geq 0,\ x_i \in \mathbb{Z} \text{ si } i \in \text{I y } x_i \in \mathbb{R} \text{ si } i \in \text{R, I}\cup\text{R} = \{1, \ldots, n\} \end{aligned}$$

donde I representa el conjunto de índices de las variables enteras y R el conjunto de índices de las variables reales.

La diferencia sustancial entre un problema de programación lineal y uno de programación lineal entera es que, salvo casos particulares, hasta el momento no se conoce ningún algoritmo capaz de resolver la segunda clase de problemas en tiempo polinomial. Más específicamente, los PPLE son de tipo *NP-Hard* [12].

El primer intento razonable por atacar un PPLE consiste en resolver el PPL asociado (que se obtiene de reemplazar la condición de integridad "$x_i \in \mathbb{Z}$" por la condición *relajada* "$x_i \in \mathbb{R}$"), y, a continuación, redondear la solución obtenida a una solución entera. Este procedimiento motiva la siguiente definición.

Definición 1.2.1 *Dado un PPLE, llamaremos* relajación lineal *al PPL que resulta de eliminar la condición de integridad de todas aquellas variables que deban asumir valores enteros.*

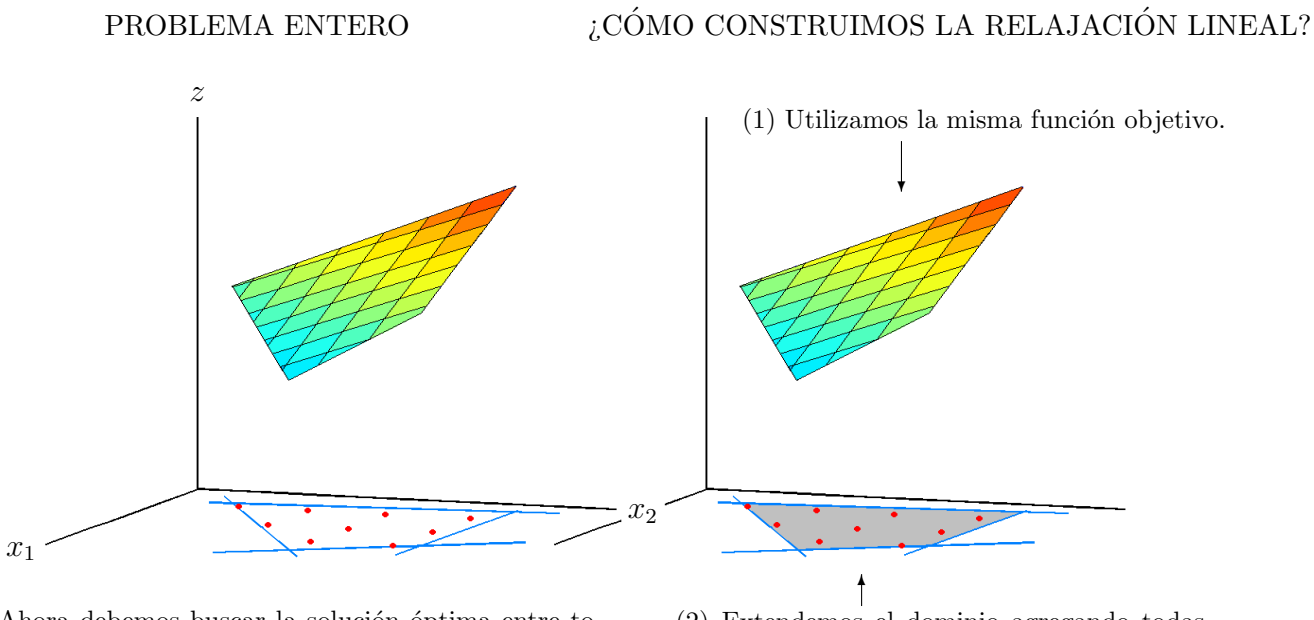

Figura 1.2: problema entero y su relajación lineal.

Si bien en algunos casos es posible obtener la solución óptima del problema entero redondeando la solución de la relajación lineal, en general esta propiedad no es válida. Habitualmente este procedimiento arroja soluciones sub-óptimas o infactibles. Sin embargo, dada la estrecha relación que existe entre ambos problemas, siempre podremos extraer la siguiente información al resolver la relajación lineal del PPLE:

1. *Si la relajación lineal es infactible*, como el dominio del problema entero está incluido en el de la relajación, esto implica que el problema entero también es infactible.
2. *Si el óptimo de la relajación lineal se alcanza en un punto de coordenadas enteras*, como el dominio del problema entero está incluido en el de la relajación lineal y la función es la misma en ambos casos, entonces, la solución hallada resulta óptima para el problema entero.
3. *Si el óptimo de la relajación lineal tiene al menos una coordenada fraccionaria que debería ser entera*. En este caso, como el valor óptimo de la función objetivo del problema entero siempre es menor o igual que el valor óptimo de la función objetivo de la relajación lineal –utilizando la justificación de item anterior–, la relajación lineal nos provee una cota superior para el valor óptimo de la función objetivo del problema entero. Esta cota es de mucha utilidad. Si tuviéramos una solución factible del problema entero cuyo valor objetivo fuera el valor de la cota superior, entonces, podríamos concluir que se trata de una solución óptima. De lo contrario, al menos nos permitiría estimar cuán lejos está nuestra solución de la solución óptima.

1.2.1. Métodos de Resolución

La mayoría de los métodos de resolución exacta de un modelo de programación lineal entera se encuadran en alguno de los siguientes esquemas:

Método de *Planos de Corte*

La estrategia de un algoritmo de *Planos de Corte* consiste en modificar el dominio de la relajación lineal asociada al problema entero hasta obtener una solución óptima que satisfaga las condiciones de integridad requeridas. Para llevar a cabo esta operación, se utilizan una serie de desigualdades lineales, llamadas "*planos de corte*", que, al ser agregadas a la formulación del problema, permiten eliminar soluciones fraccionarias de la relajación y conservar el conjunto de soluciones enteras del problema original.

El esquema general del algoritmo comienza calculando la solución óptima de la relajación lineal asociada al problema entero. Si la solución óptima resulta entera, el algoritmo se detiene. En caso contrario, si al menos una de las variables que debía ser entera resultó fraccionaria, se busca identificar una desigualdad lineal que separe la actual solución del conjunto de soluciones factibles enteras. Al agregar esta desigualdad a la formulación, se obtiene una nueva relajación del problema –más ajustada– sobre la cual puede repetirse el procedimiento. El éxito de la metodología depende, en gran medida, de la posibilidad y la eficiencia de encontrar *planos de corte* que puedan ser agregados a la formulación para *separar* soluciones fraccionarias. Es decir, de disponer de un buen *algoritmo de separación*.

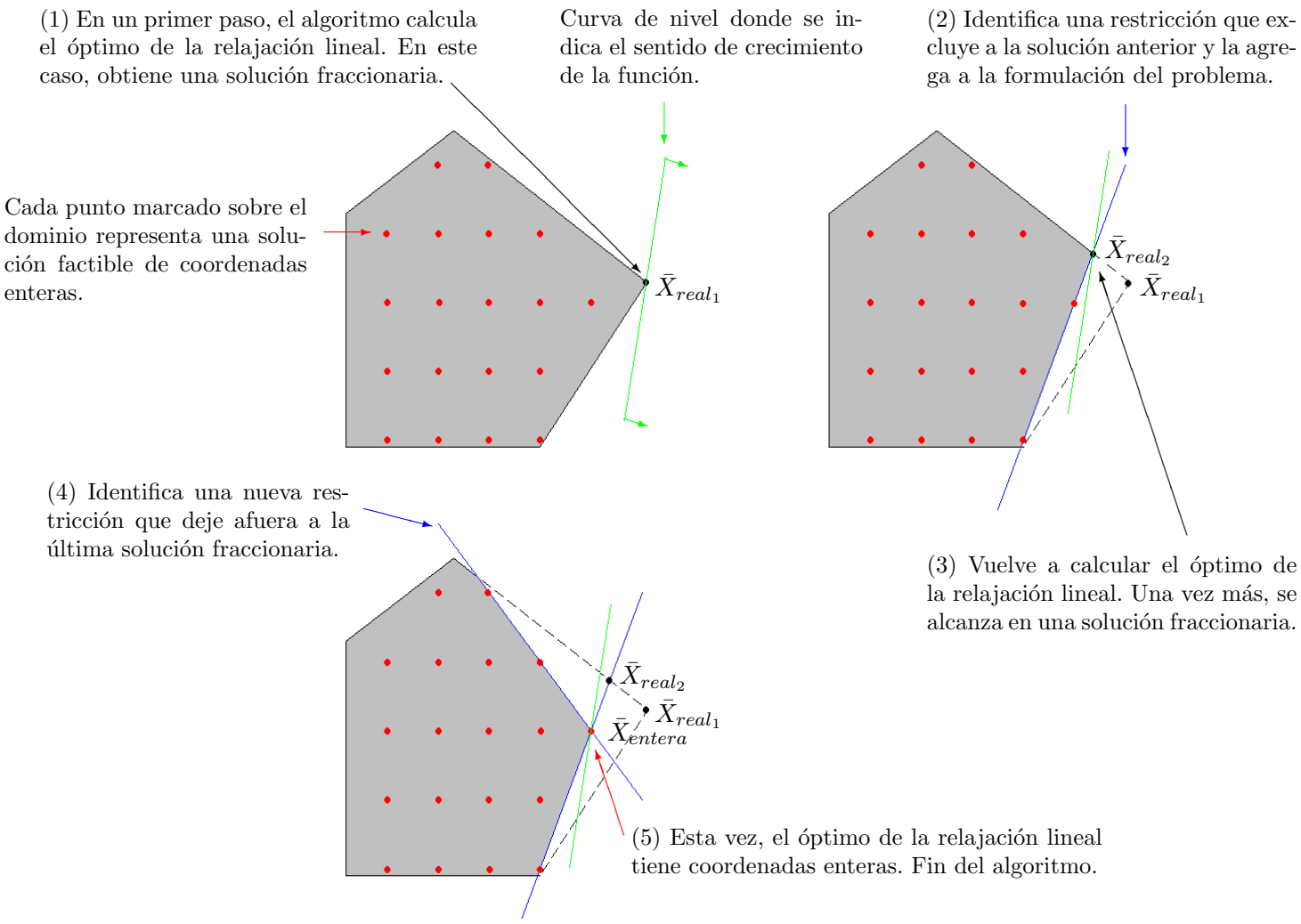

Figura 1.3: algoritmo de *Planos de Corte*.

Los planos de corte pueden ser generados bajo dos enfoques:

- *Con herramientas generales aplicables a cualquier PPLE.*

  A comienzos de los 60, Gomory [15] desarrolló un procedimiento de aplicación general para producir desigualdades válidas que cortan la solución óptima de la relajación lineal. En cada iteración, la desigualdad es obtenida a partir de la solución fraccionaria que provee la relajación lineal y utilizando exclusivamente argumentos de integrabilidad. Dicho algoritmo es convergente bajo ciertas condiciones. Si bien hasta no hace mucho tiempo no era un algoritmo usado en la práctica –por las dificultades numéricas que envuelve– hoy en día existen implementaciones muy eficientes en varios paquetes académico-comerciales. Dentro de este enfoque también están incluidos los cortes disyuntivos, cortes *cover* y cortes *flow cover*.

  Un estudio más específico del problema ayuda a obtener mejores procedimientos. Este es el sentido del próximo abordaje.

- *Explotando la estructura particular del problema.*

  Hay propiedades inherentes a cada problema que pueden ayudar a identificar mejores planos de corte. Un trabajo pionero en esta dirección fue el de Dantzig et al [8], en 1954, para el *Problema del Viajante de Comercio*. La técnica empleada permitió resolver una instancia de 49 ciudades (grande para la época) y sentó las bases de lo que hoy en día es una de las herramientas más efectivas: las técnicas poliedrales.

  Una propiedad deseada para un plano de corte es que elimine la mayor cantidad posible de soluciones no enteras del poliedro asociado a la relajación lineal. Un estudio poliedral del conjunto de soluciones factibles enteras permite disponer de *buenos* planos de corte. Los primeros estudios poliedrales fueron realizados a principios de la década de los 70 para el *Problema de Conjunto Independiente* [30] y el *Problema del Viajante* [19]. Estos trabajos significaron un importante progreso en la resolución de dichos problemas.

  Por supuesto, la aplicación de esta clase de cortes está limitada al problema particular. Sin embargo, algunas desigualdades que han sido obtenidas a partir de problemas específicos, posteriormente pudieron ser empleadas en situaciones más complejas. Desigualdades válidas para el *Problema de la Mochila* [29], y generalizaciones de las mismas, están incluídas como planos de corte en varios de los más importantes paquetes comerciales, como por ejemplo, CPLEX [20].

Método *Branch-and-Bound*

El método *Branch-and-Bound* es el resultado de una investigación financiada por *British Petroleum* para encontrar soluciones enteras a PPL. El método fue publicado por Ailsa Land y Alison Doig en 1960 [26, 21].

La estrategia de un algoritmo *Branch-and-Bound* consiste en particionar el espacio de soluciones con el objetivo de facilitar la búsqueda del óptimo. Al aplicar este concepto recursivamente, se genera un *árbol* cuya raíz corresponde al problema original y sus nodos tienen asociados subproblemas que resultan de la división en partes del espacio de búsqueda. Debido al tamaño que puede alcanzar el árbol, es esencial disponer de herramientas eficientes que permitan eliminar algunas de sus ramas. A cada nodo del árbol se le asocia la relajación lineal del problema en el subespacio de búsqueda correspondiente al nodo. En el caso que la solución del problema relajado satisfaga los requerimientos originales de integralidad, ésta es la solución buscada en esa región del espacio. Si el subproblema relajado no tiene solución, o si su valor óptimo es peor que la mejor solución entera conocida hasta el momento, no hay necesidad de seguir explorando el subconjunto de soluciones asociado a ese nodo. Por lo tanto, en cualquiera de los casos mencionados, la rama del árbol que se genera a partir del mismo puede ser *podada* (proceso de *bound*). Por el contrario, si al menos una variable de la solución óptima relajada que debe ser entera resultó fraccionaria, y si el valor óptimo de la relajación es mayor que la mejor solución entera que se dispone, no hay razón para detener la búsqueda en esa región del espacio. Para continuar con la misma, se deben generar nuevos nodos (proceso de *branching*).

El *Branch-and-Bound* es el algoritmo tradicional para resolver PPLE. La implementación más difundida, y que se utiliza en la mayoría de los paquetes comerciales, emplea la solución de la relajación lineal para el proceso de *branching* y fundamentalmente para el de *bound*.

El esquema básico del algoritmo es el siguiente:

- **Paso 1-***Inicialización:* Crear una lista L con el nodo raíz y la relajación lineal del problema. Sea $Z_{sup} = -\infty$.
- **Paso 2-***Elección del nodo:* Si L esta vacía, el algoritmo termina. Si no, elegir un nodo de L y eliminarlo de la lista.
- **Paso 3-***Bound:* Resolver la relajación lineal asociada al nodo. Si no es factible, volver al paso 2. Sea $\bar{X}$ la solución óptima y $\bar{Z}$ el valor de la función objetivo.
  - Si $\bar{X}$ es solución factible del problema, sea $Z_{sup} = max(Z_{sup}, \bar{Z})$. Volver al paso 2.
  - Si $\bar{Z} \leq Z_{sup}$, en esa rama no existe ninguna solución factible mejor que la actual. Volver al paso 2.
- **Paso 4-***Branching:* Generar subproblemas del nodo actual y agregarlos a la lista L. Volver al paso 2.

En este esquema básico no está especificada la regla a seguir para la elección de un nodo de la lista ni el proceso de generación de los subproblemas.

Con respecto a la elección del nodo, las opciones más usuales para los algoritmos enumerativos son las siguientes: *Búsqueda en Profundidad* (último nodo de la lista), *Búsqueda a lo Ancho* (primer nodo de la lista) o *Mejor Cota Primero* (el nodo con mejor valor óptimo de la relajación).

Para generar los subproblemas suele usarse la clásica dicotomía en una variable x_i. Del conjunto de variables fraccionarias de la solución óptima de la relajación, $\bar{X}$, que debían ser enteras, puede elegirse x_i como la variable tal que:

- $\bar{x}_i$ *tiene su parte fraccionaria más cercana a* $1/2$.
- $\bar{x}_i$ *tiene su parte fraccionaria más cercana a* 0.
- $\bar{x}_i$ *tiene su parte fraccionaria más cercana a* 1.
- *tiene el mayor coeficiente en la función objetivo.*
- *cumple alguna propiedad específica del problema.*

Los dos nuevos nodos que se generan, tienen asociada la región del espacio del nodo padre con el agregado de $x_i \leq \lfloor \bar{x}_i \rfloor$ o $x_i \geq \lfloor \bar{x}_i \rfloor + 1$ respectivamente. Cualquier combinación de estas reglas da origen a un árbol distinto. En [27] se realiza un análisis muy detallado sobre la performance de las distintas estrategias, pero no se llega a una respuesta concluyente sobre la supremacía de una sobre la otra aplicable a cualquier problema.

Método *Branch-and-Cut*

A comienzos de los 80, Crowder et al [4] tuvieron un gran éxito al aplicar una metodología mixta para resolver problemas binarios. Trabajaron con un algoritmo *Branch-and-Bound* pero, antes de comenzar la primera etapa de *branching*, aplicaron un algoritmo de *Planos de Corte* a la relajación lineal asociada a la raíz del árbol. De esta manera lograron mejorar la cota superior brindada por la relajación lineal y eso disminuyó el tamaño del árbol explorado. A mediados de la misma década aparecieron los primeros trabajos que extendieron la aplicación de planos de corte a otros nodos del árbol; Grötschel et al [18] presentan este enfoque en el *Problema de Ordenamiento Lineal* y Padberg et al [31] en el *Problema del Viajante de Comercio* (donde fue introducido el término "*Branch-and-Cut*").

El esquema básico del algoritmo es:

- **Paso 1-***Inicialización:* Crear una lista L con el nodo raíz y la relajación lineal del problema. Sea $Z_{sup} = -\infty$.
- **Paso 2-***Elección de nodo:* Si L esta vacía, el algoritmo termina. Si no, elegir un nodo de L y eliminarlo de la lista.
- **Paso 3-***Bound:* Resolver la relajación lineal asociada al nodo. Si no es factible, volver al paso 2. Sea $\bar{X}$ la solución óptima y $\bar{Z}$ el valor de la función objetivo.
  - Si $\bar{X}$ es solución factible del problema, sea $Z_{sup} = max(Z_{sup}, \bar{Z})$. Volver al paso 2.
  - Si $\bar{Z} \leq Z_{sup}$, no existe ninguna solución factible mejor que la actual. Volver al paso 2.
- **Paso 4-***Branching vs Cutting:* Decidir si se buscarán planos de corte.

  No: ir al paso 6 (*Branching*). Si: ir al paso 5 (*Separación*).
- **Paso 5-***Separación:* Buscar desigualdades válidas violadas por $\bar{X}$. Si no se encuentran, ir al paso 6 (*Branch*). Si se encuentran, agregarlas a la formulación e ir a paso 3 (*Bound*).
- **Paso 6-***Branching:* Generar subproblemas del nodo actual y agregarlos a la lista L. Volver al paso 2.

En la descripción del algoritmo quedan muchos puntos sin especificar, por ejemplo: ¿cuántas iteraciones realizar del algoritmo de *Planos de Corte*?, ¿cuántos cortes agregar por iteración?, ¿qué hacer con los cortes generados en los distintos nodos del árbol? La performance del algoritmo depende de estos factores y de muchos otros. En la práctica, lograr un equilibrio entre ellos no es tarea fácil y depende, en gran medida, del problema particular que se quiere resolver.

Por último, vale la pena señalar dos características que pueden ser explotadas al utilizar esta técnica.

- Usar planos de corte que, a pesar de ser generados en un nodo del árbol, sean válidos para todo el árbol. Esto permite aprovechar el trabajo de identificación de cortes y, de esa manera, disminuir los requerimientos de memoria necesario para cada nodo. En la práctica, se dispone de un espacio de memoria común "*pool de cortes*" donde se almacenan las desigualdades y a las que se hace referencia desde cada subproblema.
- Realizar un estudio de la estructura poliedral del problema para generar desigualdes válidas específicas de cada situación.

Los algoritmos descriptos en este capítulo se aplican actualmente en la resolución de PPLE y están presentes en la mayoría de los paquetes comerciales. A su vez, continuamente aparecen en la literatura del área, trabajos que aplican estas técnicas enriqueciendo la implementación con desigualdades válidas provenientes de estudios poliedrales específicos del conjunto de soluciones factibles.

Capítulo 2

Problemas de la Mochila

Dentro de los PPLE se destacan modelos conocidos bajo el nombre genérico de *Problemas de la Mochila*. Si bien tienen importancia propia, ya que surgen de aplicaciones en la vida real y como subproblemas de situaciones más complejas, también juegan un rol clave en la derivación de propiedades que luego pueden ser extendidas a problemas más generales. La literatura al respecto es muy vasta, lo que demuestra la importancia de estos problemas en el ámbito de la programación lineal entera. De aquí la necesidad de poder resolverlos eficientemente y es por eso que los utilizaremos para evaluar la performance de nuestro algoritmo. El objetivo de este capítulo es presentar estos modelos y algunas propiedades útiles que serán usadas más adelante. Una muy buena presentación de las distintas clases de problemas *mochila*, junto con resultados computacionales de la aplicación de diversas técnicas, puede ser consultada en [28] y [24].

2.1. Problema de la Mochila No Acotado (UKP)

Supongamos que disponemos de una mochila con capacidad de carga "c" y de una cantidad *no acotada* de objetos, de "n" clases distintas, para llenarla. Todos los objetos de la clase "i" poseen un valor (*profit*) "p_i" y un peso (*weight*) "w_i". El problema consiste en decidir cuántos objetos poner de cada tipo para maximizar el valor de la carga sin sobrepasar la capacidad de la mochila. Su formulación es la siguiente:

$$\text{maximizar} \quad f = \sum_{i=1}^{n} p_i x_i \quad \longleftarrow \text{Valor de la carga.}$$

$$\text{sujeta a} \quad \sum_{i=1}^{n} w_i x_i \leq c \quad \longleftarrow \text{Restricción de la capacidad de la mochila.}$$

$$x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, \ldots, n \quad \longleftarrow \text{Los objetos no se pueden fraccionar.}$$

donde la variable x_i representa la cantidad de objetos de clase i que cargamos en la mochila. Además, estamos suponiendo que todos los coeficientes del problema son números enteros positivos.

El UKP pertenece a la clase de problemas *NP-Hard* [12], sin embargo, puede resolverse en tiempo pseudo-polinomial –O(cn)– mediante *Programación Dinámica* [24]. Sus aplicaciones más reconocidas se describen a continuación.

- *Gestión Financiera*: se desea invertir una cantidad de dinero c –parcial o totalmente– en n tipos de acciones. Cada acción tiene un costo w_i y una retribución esperada p_i. El problema que se plantea es cómo elegir la mejor inversión posible.
- *Embarque de Cargas*: dado un conjunto de objetos, se debe elegir un subconjunto de ellos para cargar un avión, un barco, o un contenedor, maximizando el valor de la carga sin sobrepasar la capacidad.
- *Cutting Stock*: a partir de bobinas de cartón de ancho c se deben obtener n bobinas de tamaños distintos. Cada bobina i tiene ancho w_i y una demanda d_i. El problema consiste en decidir cómo cortar las bobinas madre (patrón de corte) de manera tal de minimizar el desperdicio. La caracterización de un patrón de corte responde a un problema mochila.

2.1.1. Propiedades del UKP

- Relajación lineal

  La relajación lineal del UKP $\Big($que notaremos "C(UKP)"$\Big)$ es, como dijimos, el PPL que resulta al eliminar las restricciones de integridad del modelo entero:

$$\text{maximizar} \quad f = \sum_{i=1}^{n} p_i x_i$$

$$\text{sujeta a} \quad \sum_{i=1}^{n} w_i x_i \leq c$$

$$x_i \geq 0,\ x_i \in \mathbb{R},\ i = 1, \ldots, n \quad \longleftarrow \text{Quitamos la restricción de integridad, ahora los objetos se pueden fraccionar.}$$

Proposición 2.1.1 *La solución óptima, $\bar{X}$, del C(UKP) es*

$$\bar{X} = \Big(0, ..., 0, \underset{\substack{\uparrow \\ j\text{-ésima coordenada}}}{\frac{c}{w_j}}, 0, ..., 0\Big), \quad \textit{donde j es la clase más valiosa: } \frac{p_j}{w_j} \geq \frac{p_i}{w_i} \ \ \forall\ i \neq j$$

Demostración:

Primero busquemos una cota superior para el valor de la función objetivo del C(UKP).

Si $X = (x_1, ..., x_n)$ es una solución factible

$$\Rightarrow \quad f(X) \ = \ \sum_{i=1}^{n} p_i x_i \ = \ \sum_{i=1}^{n} \frac{p_i}{w_i}\ w_i x_i \ \leq \ \sum_{i=1}^{n} \frac{p_j}{w_j}\ w_i x_i \ = \ \frac{p_j}{w_j}\ \sum_{i=1}^{n} w_i x_i \ \leq \ \underset{\substack{\uparrow \\ X \text{ factible}}}{\frac{p_j}{w_j}\ c.}$$

Luego, como nuestro candidato, $\bar{X}$, realiza la cota $\Big(f(\bar{X}) = \dfrac{c}{w_j}\ p_j\Big)$ y es factible, concluimos que es óptimo ■

- Cotas superiores

  Sean "f(UKP)" el valor óptimo de la función objetivo del UKP y "f(C(UKP))" el valor óptimo de la función objetivo del C(UKP). Busquemos, usando la relajación lineal, cotas superiores para f(UKP).

  Supongamos, para simplificar la notación, que las clases están numeradas cumpliendo:

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}.$$

  1. Como $f(\text{UKP}) \leq f(\text{C(UKP)})$ y f(UKP) es un número entero –tanto p_i como x_i son valores enteros–, entonces, la primera cota se obtiene de truncar el valor f(C(UKP)):

$$\underbrace{\bar{X} = \Big(\frac{c}{w_1}, 0, \ldots, 0\Big)}_{\text{óptimo de la relajación}} \Rightarrow f(\text{C(UKP)}) = f(\bar{X}) = \underbrace{\frac{c}{w_1}\, p_1}_{\text{evaluamos}} \Rightarrow U_1 = \lfloor f(\text{C(UKP)}) \rfloor = \underbrace{\Big\lfloor \frac{c}{w_1}\, p_1 \Big\rfloor}_{\text{truncamos}}.$$

  2. Imponiendo la condición "$\bar{x}_1 \leq \Big\lfloor \frac{c}{w_1} \Big\rfloor$", que debe valer para todas las soluciones factibles del problema entero, la solución continua resulta:

$$\bar{X} = \Big(\Big\lfloor \frac{c}{w_1} \Big\rfloor, \frac{\bar{c}}{w_2}, 0, \ldots, 0 \Big), \text{ donde } \bar{c} = c - \Big\lfloor \frac{c}{w_1} \Big\rfloor.$$

     Calculando $f(\bar{X})$ y tomando parte entera, conseguimos la segunda cota superior:

$$f(\bar{X}) = \Big\lfloor \frac{c}{w_1} \Big\rfloor\, p_1 + \frac{\bar{c}}{w_2}\, p_2 \Rightarrow U_2 = \Big\lfloor \frac{c}{w_1} \Big\rfloor\, p_1 + \Big\lfloor \frac{\bar{c}}{w_2}\, p_2 \Big\rfloor.$$

  Otras cotas más ajustadas –propuestas por Martello y Toth [28]– son las siguientes:

  3. $U_3 = max(A, B)$, donde

$$A = z' + \Big\lfloor c' \, \frac{p_3}{w_3} \Big\rfloor$$

$$B = z' + \Big\lfloor p_2 - (w_2 - c') \, \frac{p_1}{w_1} \Big\rfloor$$

$$z' = \Big\lfloor \frac{c}{w_1} \Big\rfloor\, p_1 + \Big\lfloor \frac{\bar{c}}{w_2} \Big\rfloor\, p_2$$

$$c' = \bar{c} \; (mod \; w_2)$$

  4. $U_4 = max(A, C)$, donde

$$A = z' + \Big\lfloor c' \, \frac{p_3}{w_3} \Big\rfloor$$

$$C = z' + \Big\lfloor \Big(c' + \Big\lceil \frac{(w_2 - c')}{w_1} \Big\rceil \, w_1 \Big) \, \frac{p_2}{w_2} - \Big\lceil \frac{(w_2 - c')}{w_1} \Big\rceil \, p_1 \Big\rfloor$$

$$z' = \Big\lfloor \frac{c}{w_1} \Big\rfloor\, p_1 + \Big\lfloor \frac{\bar{c}}{w_2} \Big\rfloor\, p_2$$

$$c' = \bar{c} \; (mod \; w_2)$$

  Se puede demostrar [28] que $U_4 \leq U_3 \leq U_2 \leq U_1$.

2.1.2. Estado del problema

El UKP es un problema sumamente estudiado en la literatura, se conocen algoritmos muy eficientes que permiten resolver instancias de decenas de miles de variables en pocos segundos [28]. En la mayoría de los casos, estos métodos realizan una primera fase de "*pre-solve*" que consiste en: 1) reducir el problema original a uno mucho más pequeño al que denominan *core problem*; 2) sobre el *core problem* aplican algún *criterio de dominancia* para simplificarlo aún más. Una vez finalizada esta primera etapa, buscan la solución del problema resultante empleando un algoritmo de búsqueda como puede ser *Branch-and-Bound* o *Programación Dinámica*.

2.2. Problema de la Mochila Multidimensional (MKP)

Consideremos ahora una mochila con "m" capacidades: "$c_1, \ldots, c_m$" (peso, volumen, etc.) y exactamente "n" objetos para llenarla. Cada objeto "i" tiene un valor "p_i" y "m" características: "$w_{i1}, \ldots, w_{im}$" (peso, volumen, etc.). El problema consiste en decidir qué objetos poner en la mochila, con el fin de maximizar el valor de la carga sin sobrepasar ninguna de sus capacidades. Formulación:

$$\begin{aligned} \text{maximizar} \quad & f = \sum_{i=1}^{n} p_i x_i \\ \text{sujeta a} \quad & \sum_{i=1}^{n} w_{ij} x_i \leq c_j, \quad j = 1, 2, \ldots, m \\ & x_i \in \{0, 1\} \ \forall i = 1, \ldots, n \end{aligned}$$

donde x_i es una variable binaria que vale 1 si el objeto i es cargado en la mochila y 0 si no. Además, estamos suponiendo que todos los coeficientes del problema son enteros positivos.

El MKP es el modelo más general con el que nos podemos enfrentar dentro de la familia de problemas *mochila* con variables binarias. Entre sus aplicaciones más reconocidas figuran: *Inversión de Capital*, *Selección de Proyectos*, *Cutting Stock* y *Embarque de Cargas*.

2.2.1. Estado del problema

La descripción más detallada acerca del estado del MKP la encontramos en el artículo de Fréville [10]. Este artículo señala al *Branch-and-Cut* (bajo una primera fase de pre-procesamiento) como el método exacto más eficiente a la hora de resolver instancias del MKP, y a los *solvers* comerciales CPLEX [20] y XPRESS-MP [9] como algunas de las mejores implementaciones disponibles.

En el trabajo de Cherbaka [5], se puede encontrar una descripción detallada de los distintos métodos exactos utilizados en la resolución del MKP junto con el tamaño de las instancias testeadas por parte de cada uno de los autores. A continuación, copiamos el cuadro donde se resume esta última información y una breve descripción de cada uno de los métodos.

Tabla 1. Tamaño de las instancias resueltas de tipo MKP en la literatura [5].

	Rango de variables		Rango de restricciones	
Autores	min.	max.	min.	max.
Balas (1965)		40		22
Soyster and Slivka (1977)	50	400	5	10
Shih (1979)	30	90		5
Gavish and Pirkul (1985)	20	500	3	5
Gabrel and Minoux (2002)		180		60

Balas [2] fue uno de los primeros en desarrollar un enfoque exacto para el MKP. Presentó un algoritmo de tipo *Branch-and-Bound* en el cual todas las variables son inicializadas en 0 y se incrementan a 1 basándose en un algoritmo *pseudo-dual*. A cada paso, el algoritmo identifica aquellas ramas que conducen a problemas infactibles y las poda. Un aspecto destacable del algoritmo es que no requiere de la solución de la relajación lineal para su funcionamiento.

Soyster y Slivka [34] proporcionaron un algoritmo que mejora el número de iteraciones del procedimiento propuesto por Balas. Este enfoque forma subproblemas usando la solución de la relajación lineal y luego resuelve cada subproblema utilizando el algoritmo de Balas. El tamaño de los subproblemas depende del número de restricciones del problema original, por lo que este algoritmo se comporta bien únicamente sobre modelos con pocas restricciones.

Shih [33] diseñó un procedimiento de tipo *Branch-and-Bound* definiendo sus propias estrategias de *poda* y *ramificación*. La cota superior correspondiente al nodo se calcula teniendo en cuenta cada uno de los problemas mochila de forma independiente y luego resolviendo la relajación lineal de cada uno de ellos. La más pequeña de estas cotas se utiliza como límite superior del nodo en cuestión. Para definir la estrategia de *branching* se utiliza la solución óptima del problema mochila cuya cota superior resultó mínima. Sobre problemas con un máximo de 90 variables y 5 restricciones, el método demostró ser superior al algoritmo de Balas en relación al número de iteraciones y a los tiempos de resolución.

Gavish y Pirkul [13] desarrollaron distintos tipos de relajaciones lineales (Lagrangeana, *surrogate* y *composite*) y reglas de *branching* que luego implementaron en un esquema de tipo *Branch-and-Bound*. Como resultado de esto, obtuvieron un algoritmo que supera al propuesto por Shih en lo que respecta al tiempo de CPU y al tamaño de las instancias que pudieron ser resueltas.

Gabrel y Minoux [11] trabajaron sobre un procedimiento de separación para identificar planos de corte que puedan ser aplicados en la resolución del MKP. En relación a los experimentos computacionales, mostraron una reducción en los tiempos de CPU –en comparación a la versión CPLEX 6.5– sobre instancias de hasta 180 variables y 60 restricciones.

Por último, el libro de Hans Kellerer [24] es otra buena referencia. Se trata de una recopilación de las publicaciones realizadas hasta el 2004 sobre los distintos problemas de tipo *mochila*. En palabras del autor: “La dificultad del MKP se muestra en el hecho de que el tamaño de las instancias que pueden ser resueltas en forma exacta está acotado a 500 variables y 10 restricciones. (. . .) Desde un punto de vista práctico, hay que tener en cuenta que la mayoría de los casos involucran sólo un número pequeño de restricciones ($m < 10$) pero posiblemente un número grande de variables.”

Capítulo 3

El Método PSA

En el capítulo 1, llevamos a cabo un repaso de los principales algoritmos empleados en la resolución de PPLE. En todos los casos, vimos que la estrategia para buscar el óptimo consistía en modificar el dominio del problema –habiendo considerado previamente su relajación– mediante el agregado de nuevas desigualdades lineales. En el caso del algoritmo de *Planos de Corte*, las *nuevas* desigualdades eran utilizadas para separar soluciones fraccionarias de la relajación y conservar el conjunto de soluciones enteras factibles del problema original. En el caso de los métodos *Branch-and-Bound* y *Branch-and-Cut*, las desigualdades eran usadas para particionar el dominio del problema y eliminar soluciones fraccionarias de la relajación. Facilitando, de esa manera, la búsqueda del óptimo. Con un enfoque distinto, en este capítulo vamos a presentar un algoritmo que no altera el dominio del problema y, como consecuencia, evita agregar restricciones adicionales a la formulación.

3.1. Ideas Básicas y Motivación

Consideremos un problema general de optimización entera-*pura*. Supongamos que la función a maximizar "f" es una función continua y que el dominio sobre el que debemos trabajar es un conjunto compacto y convexo. Podemos pensar en la situación de la figura 3.1.

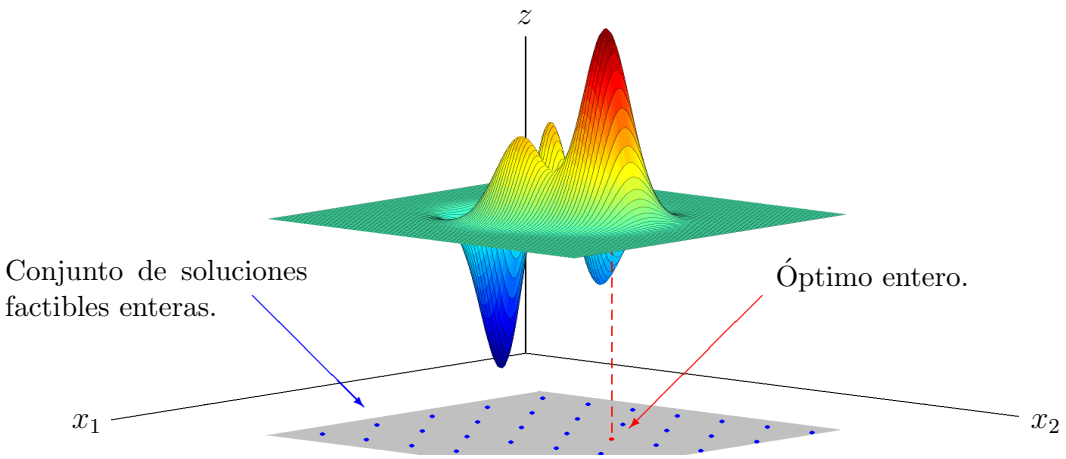

Figura 3.1: presentación del problema.

Nuestra propuesta para encontrar la solución entera del problema planteado es la siguientes. Supongamos que a partir de la función f fuéramos capaces de calcular las siguientes proyecciones:

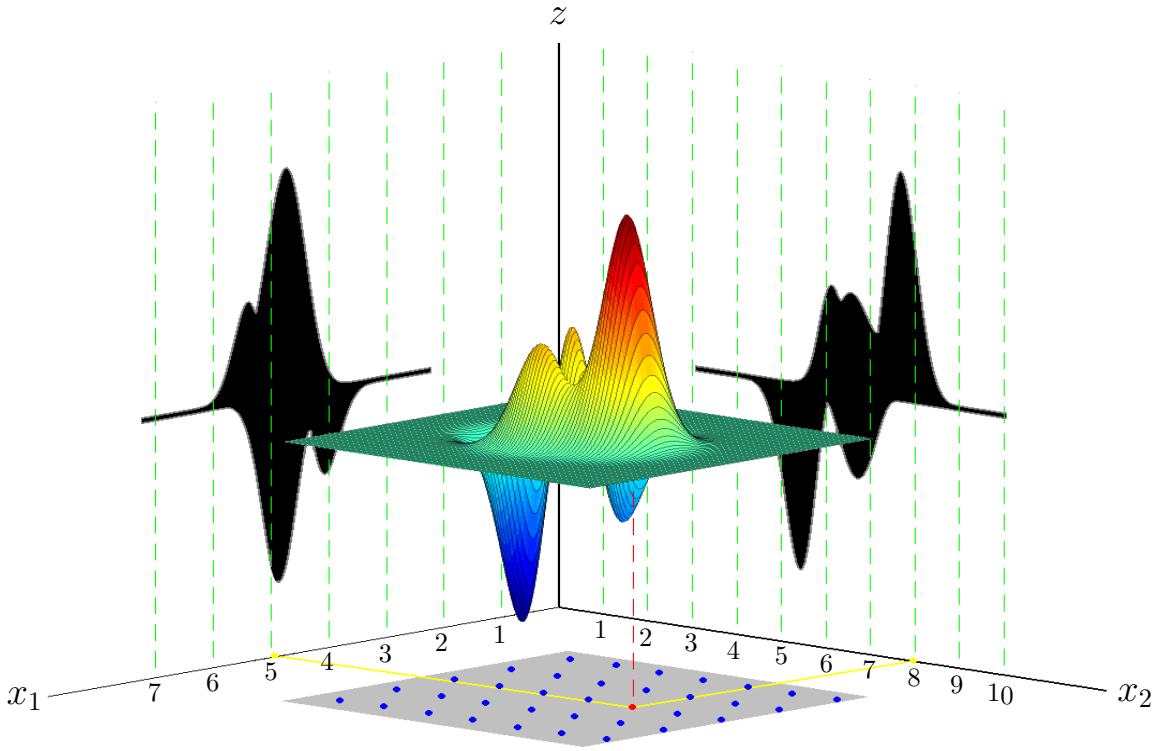

Figura 3.2: proyecciones.

Llamaremos "*proyección sobre la variable* x_1", y la notaremos "$\mathbf{P_{X_1}}$", a la *sombra* que proyecta la función sobre el plano x_1z al ser *iluminada* desde la derecha. De la misma manera, llamaremos "*proyección sobre la variable* x_2", "$\mathbf{P_{X_2}}$", a la *sombra* que proyecta la función sobre el plano x_2z al ser *iluminada* desde la izquierda (figura 3.2). Podemos pensar que estas proyecciones son el resultado de *aplastar* el gráfico de la función contra esos planos.

Luego, si tuviéramos que proponer un *candidato* para la solución entera del problema, usando *exclusivamente* la información que aportan $\mathbf{P_{X_1}}$ y $\mathbf{P_{X_2}}$, la opción más razonable sería elegir el valor entero 5 para la primera coordenada de nuestro candidato y el valor entero 8 para la segunda coordenada. Esta elección está basada en que, para esos valores enteros, las proyecciones alcanzan su mayor altura. De esta manera, el *candidato* que resulta de examinar las proyecciones –el punto (5,8)– coincide con la solución óptima del problema.

Si bien el ejemplo que acabamos de exponer es muy limitado (posee tan solo dos variables), es suficiente para mostrar la utilidad que podrían llegar a tener las *proyecciones* como herramientas para localizar soluciones enteras. El algoritmo que presentaremos al final de este capítulo no es otra cosa que una generalización –más sofisticada– del procedimiento que acabamos de describir. La estrategia del algoritmo PSA consiste en calcular las proyecciones del problema (tantas como variables contenga el problema original), para luego, utilizando esa información, generar una a una las coordenadas de la solución entera. Con este objetivo en mente, dedicaremos algo más de la mitad de este capítulo a: 1º) definir formalmente qué entendemos por *proyección* y 2º) deducir una serie de propiedades que nos serán útiles a la hora de construir el algoritmo.

3.2. Modelado y Definiciones

A partir de la figura 3.2, observamos que cada proyección es un conjunto de puntos limitado entre dos curvas. Llamaremos "*proyección superior*", y la notaremos "$P^{sup}_{X_i}(x_i)$", a la curva que lo acota por arriba, y "*proyección inferior*", "$P^{inf}_{X_i}(x_i)$", a la que lo acota por debajo (ver figura 3.3). Por lo tanto, el problema de calcular $\mathbf{P_{X_i}}$ se reduce a encontrar $P^{sup}_{X_i}(x_i)$ y $P^{inf}_{X_i}(x_i)$ y luego considerar la región encerrada entre ambas curvas.

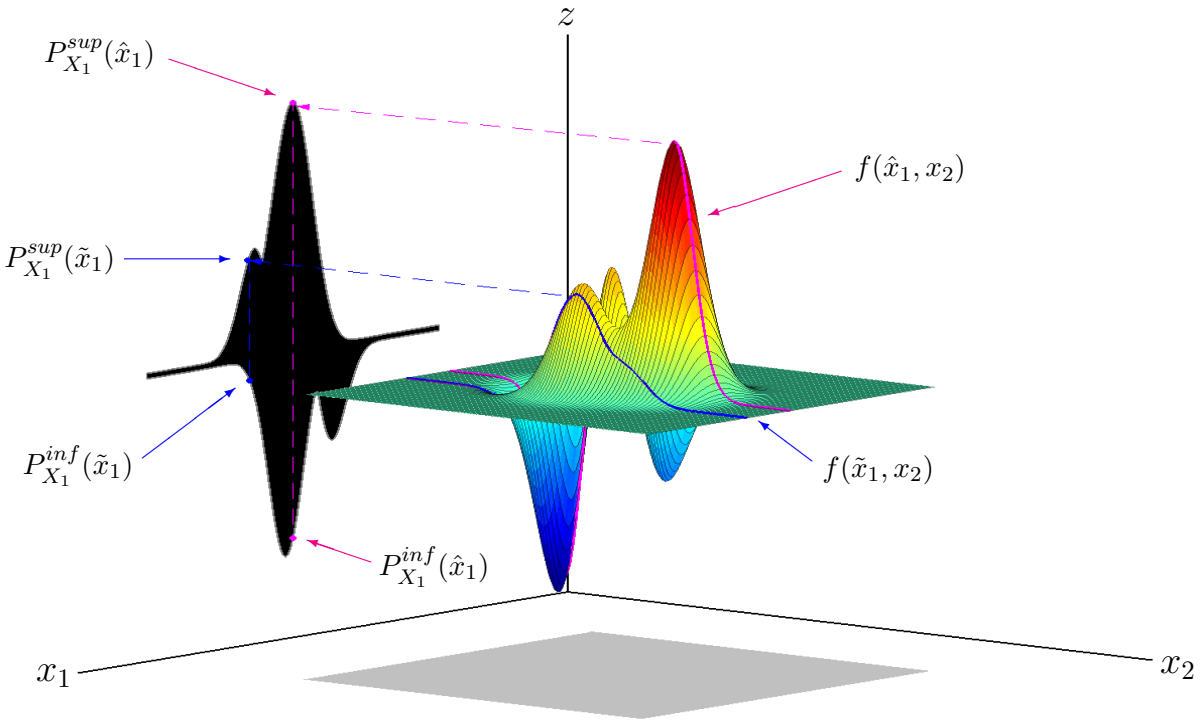

Figura 3.3: modelado.

En base a la figura 3.3 definimos:

$$P^{sup}_{X_1}(x_1) := \max_{x_2} f(x_1, x_2), \text{ con } x_2 \in Dom\Big(f(x_1,\ .\ )\Big)$$

$$P^{inf}_{X_1}(x_1) := \min_{x_2} f(x_1, x_2), \text{ con } x_2 \in Dom\Big(f(x_1,\ .\ )\Big).$$

$$\text{Luego, } \mathbf{P_{X_1}} := \Big\{(x_1, z) : x_1 \in Dom(f)\ \wedge\ P^{inf}_{X_1}(x_1) \leq z \leq P^{sup}_{X_1}(x_1)\Big\}.$$

Análogamente definimos $P^{sup}_{X_2}(x_2)$, $P^{inf}_{X_2}(x_2)$ y $\mathbf{P_{X_2}}$ intercambiando los roles de las variables x_1 y x_2 en las expresiones anteriores.

La extensión natural de estas definiciones para el caso general es la siguiente.

Definición 3.2.1 *Sea $f : D \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ una función continua y D un dominio compacto y convexo, definimos:*

$$P_{X_i}^{sup}(x_i) := \max_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n} f(x_1,\ldots,x_i,\ldots,x_n),\ con\ (x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)\ \in Dom\Big(f(\ldots,x_i,\ldots)\Big)$$

$$P_{X_i}^{inf}(x_i) := \min_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n} f(x_1,\ldots,x_i,\ldots,x_n),\ con\ (x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)\ \in Dom\Big(f(\ldots,x_i,\ldots)\Big)$$

Entonces la proyección sobre x_i queda definida como

$$\mathbf{P_{X_i}} := \Big\{(x_i,z) : x_i \in Dom(f)\ \wedge\ P_{X_i}^{inf}(x_i) \leq z \leq P_{X_i}^{sup}(x_i)\Big\}.$$

Ejemplo: consideremos una función de tres variables, $f(x_1,x_2,x_3)$, definida sobre el intervalo [2,12]×[2,12]×[1,4]. Por el momento no nos preocupemos por la fórmula de f, simplemente imaginemos una superficie en movimiento (figura 3.4). Si aplicáramos la definición anterior sobre cada una de las variables para calcular $P_{X_i}^{sup}(x_i)$ y $P_{X_i}^{inf}(x_i)$, y luego consideráramos la región encerrada entre ambas curvas, obtendríamos las proyecciones $\mathbf{P_{X_1}}$, $\mathbf{P_{X_2}}$ y $\mathbf{P_{X_3}}$ presentadas en la figura 3.5.

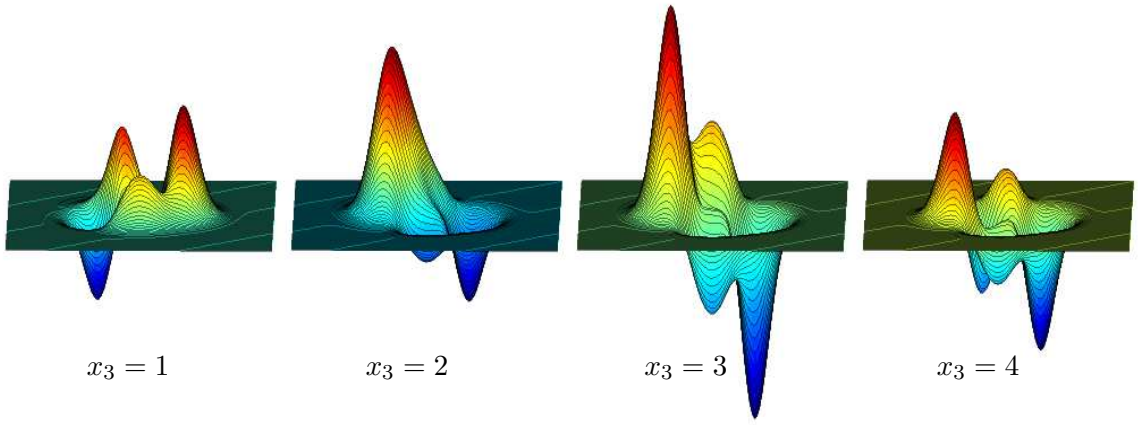

Figura 3.4: $f(x_1,x_2,x_3)$.

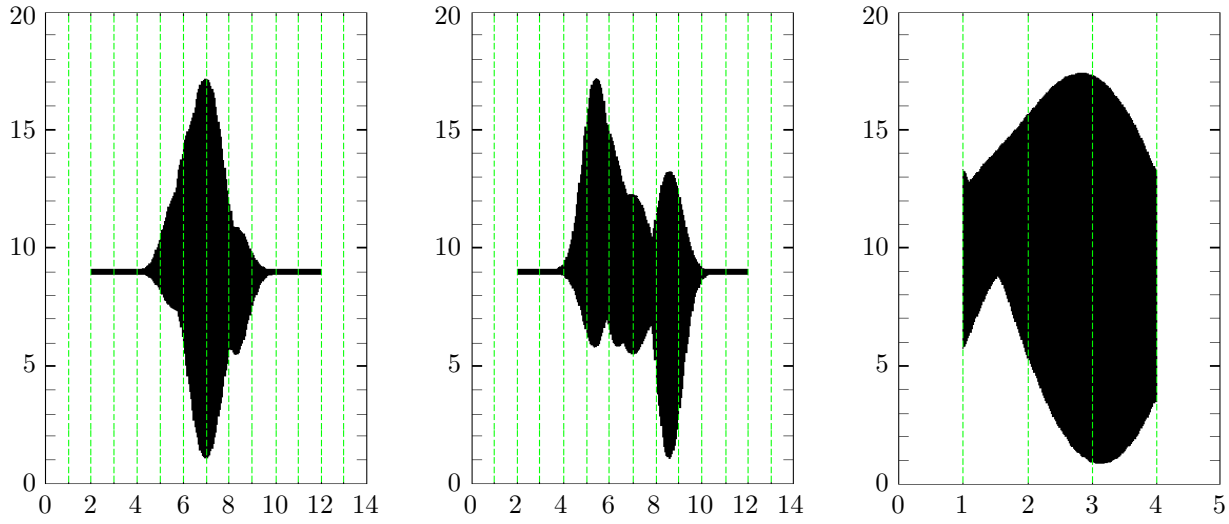

Figura 3.5: proyecciones de una función de tres variables.

Algunas observaciones:

1. Las proyecciones "$\mathbf{P_{X_i}}$" son siempre conjuntos de $\mathbb{R}^2$ independientemente del número de variables que contenga el problema. Cualquier punto dentro de este conjunto es de la forma "(x_i, z)", la primera coordenada corresponde a la variable sobre la que estamos proyectando y la segunda a la altura de la función.

2. ¿Qué información aportan las proyecciones? Nos muestran el comportamiento de la función en términos de los valores de cada una de las variables de manera independiente. Esto nos permite descomponer el problema original en subproblemas de una sola variable y, de esa manera, reducir la dificultad del planteo inicial.

3. En el caso en que f sea una función de una sola variable, la proyección coincide con el gráfico de la función $\Big($porque $P^{sup}_{X_1}(x_1) = P^{inf}_{X_1}(x_1) = f(x_1)\Big)$.

3.3. Propiedades

Proposición 3.3.1 *Sea f una función continua definida sobre un dominio convexo y compacto. Entonces, para cada punto $(x_i, z) \in \mathbf{P_{X_i}}$, existe un punto $X \in Dom(f)$ tal que $(X)_i = x_i$ y $f(X) = z$.*

Esta propiedad nos asegura que cada punto de la proyección es el *reflejo*, vía f, de algún punto del dominio.

Demostración:

Sea $(\mathbf{x_i}, \mathbf{z}) \in \mathbf{P_{X_i}}$. Primero veamos que los puntos $\Big(\mathbf{x_i}, P^{sup}_{X_i}(\mathbf{x_i})\Big)$ y $\Big(\mathbf{x_i}, P^{inf}_{X_i}(\mathbf{x_i})\Big)$, pertenecientes a la proyección, son el reflejo de algún punto del dominio.

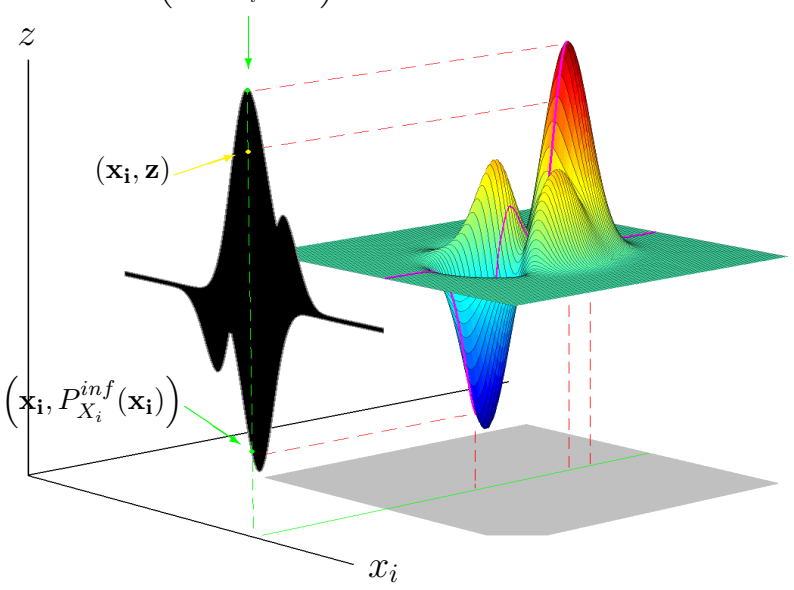

Figura 3.6: demostración.

Como f es continua sobre un dominio compacto y convexo, entonces la función "$f|_{x_i=\mathbf{x_i}}$" alcanza su valor máximo. Es decir, existe $\hat{X} \in Dom(f)$ tal que

$$(\hat{X})_i = \mathbf{x_i} \text{ y } f(\hat{X}) = \text{máx } f(x_1, \ldots, \mathbf{x_i}, \ldots, x_n) = P_{X_i}^{sup}(\mathbf{x_i}).$$

De esta manera, probamos que el punto $\left(\mathbf{x_i}, P_{X_i}^{sup}(\mathbf{x_i})\right)$ es la imagen de un punto del dominio. Análogamente, se puede demostrar esta misma propiedad para $\left(\mathbf{x_i}, P_{X_i}^{inf}(\mathbf{x_i})\right)$.

Ahora veamos que el punto $(\mathbf{x_i}, \mathbf{z})$, con $P_{X_i}^{inf}(\mathbf{x_i}) < \mathbf{z} < P_{X_i}^{sup}(\mathbf{x_i})$, también es el reflejo de un punto del dominio.

Como f es continua, se sigue que "$f|_{x_i=\mathbf{x_i}}$" también es una función continua. Además, el dominio restringido a $x_i = \mathbf{x_i}$ sigue siendo convexo y compacto. Por lo tanto, $f|_{x_i=\mathbf{x_i}}$ recorre todos los valores entre $P_{X_i}^{inf}(\mathbf{x_i})$ y $P_{X_i}^{sup}(\mathbf{x_i})$ y, para cada uno de esos valores, existe un punto $\mathbf{X} \in Dom(f)$ tal que $(\mathbf{X})_i = \mathbf{x_i}$ y $f(\mathbf{X}) = \mathbf{z}$. Es decir, $(\mathbf{x_i}, \mathbf{z})$ también es la imagen de un punto del dominio ■

Proposición 3.3.2 *Sea f un función lineal definida sobre un dominio compacto y convexo. Luego, las proyecciones "$\mathbf{P_{X_i}}$" son conjuntos compactos y convexos.*

Demostración:

Sean $(\tilde{x}_i, \tilde{z})$ y $(\hat{x}_i, \hat{z})$ dos puntos cualesquiera en $\mathbf{P_{X_i}}$. Para ver que la proyección es un conjunto convexo, debemos probar que todos los puntos de la forma $\alpha(\tilde{x}_i, \tilde{z}) + (1-\alpha)(\hat{x}_i, \hat{z}),\ \alpha \in [0,1]$, también pertenecen a $\mathbf{P_{X_i}}$.

Por la proposición anterior, existen $\tilde{X}$ y $\hat{X} \in Dom(f)$ tales que

$$(\tilde{X})_i = \tilde{x}_i \text{ y } f(\tilde{X}) = \tilde{z} \quad \text{y} \quad (\hat{X})_i = \hat{x}_i \text{ y } f(\hat{X}) = \hat{z}.$$

Luego, como el dominio de f es convexo, se sigue que:

$$\alpha\tilde{X} + (1-\alpha)\hat{X} \in Dom(f) \ \ \forall\ \alpha \in [0,1].$$

Consideremos un punto genérico del segmento que va de $(\tilde{x}_i, \tilde{z})$ a $(\hat{x}_i, \hat{z})$:

$$(\mathbf{x_i}, \mathbf{z}) = \beta(\tilde{x}_i, \tilde{z}) + (1-\beta)(\hat{x}_i, \hat{z}), \quad \text{con } \beta \in [0,1].$$

Debemos mostrar que para este punto existe $\mathbf{X} \in Dom(f)$ tal que $(\mathbf{X})_i = \mathbf{x_i}$ y $f(\mathbf{X}) = \mathbf{z}$ $\Big($es decir, $(\mathbf{x_i}, \mathbf{z}) \in \mathbf{P_{X_i}}\Big)$.

Sea $\mathbf{X} := \beta\tilde{X} + (1-\beta)\hat{X}$; veamos que este punto verifica todas las condiciones.

1. $\mathbf{X} \in Dom(f)$ pues pertenece al segmento $\alpha\tilde{X} + (1-\alpha)\hat{X},\ \alpha \in [0,1]$.
2. $(\mathbf{X})_i = \Big(\beta\tilde{X} + (1-\beta)\hat{X}\Big)_i = \beta\tilde{x}_i + (1-\beta)\hat{x}_i = \mathbf{x_i}$.
3. Por la linealidad de f, $f(\mathbf{X}) = f\Big(\beta\tilde{X} + (1-\beta)\hat{X}\Big) = \beta f(\tilde{X}) + (1-\beta)f(\hat{X}) = \beta\tilde{z} + (1-\ \beta)\hat{z} = \ \mathbf{z}$.

Extendiendo este procedimiento para todos los puntos del segmento $\alpha(\tilde{x}_i, \tilde{z}) + (1-\alpha)(\hat{x}_i, \hat{z}),\ \alpha \in\ [0,1]$, concluimos que $\mathbf{P_{X_i}}$ es un conjunto convexo.

Por último, $\mathbf{P_{X_i}}$ es un conjunto compacto porque es la imagen de un compacto vía la composición de dos funciones continuas (f y max) ■

Proposición 3.3.3 *Bajo las hipótesis de la propiedad anterior. Si $\bar{x}_i \in \mathbb{R}$ es el máximo de $\mathbf{P_{X_i}}$ y la proyección está definida en $[a_i, b_i]$, entonces, $P^{sup}_{X_i}(x_i)$ es estrictamente creciente (o constante) en $[a_i, \bar{x}_i)$ y estrictamente decreciente (o constante) en $(\bar{x}_i, b_i]$.*

Demostración:

Consecuencia de la convexidad de $\mathbf{P_{X_i}}$ ■

Definición 3.3.4 *Sea $f : \mathbb{R}^n \mapsto \mathbb{R}$. Llamaremos* nivel *o* altura *a cada uno de los valores que puede alcanzar la función.*

Definición 3.3.5 *Sea $f : \mathbb{R}^n \mapsto \mathbb{R}$. Si $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_n)$ es una solución óptima de f tal que $f(\bar{X}) = N$, entonces diremos que N es el* nivel óptimo *de la función.*

Proposición 3.3.6 *Sea $f : D \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ continua, D un dominio convexo y compacto. Si $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_n)$ es una solución óptima del problema (vale tanto para problemas continuos, enteros-puros o enteros-mixtos) y N es el nivel óptimo de f, entonces, $(\bar{x}_i, N) \in \mathbf{P_{X_i}}\ \forall i$.*

Demostración:

Por definición, debemos probar que $P^{inf}_{X_i}(\bar{x}_i) \leq N \leq P^{sup}_{X_i}(\bar{x}_i)$.
$P^{inf}_{X_i}(\bar{x}_i) :=$ mín valor $f(x_1, \ldots, \bar{x}_i, \ldots, x_n) \leq f(\bar{x}_1, \ldots, \bar{x}_i, \ldots, \bar{x}_n) = N$
$P^{sup}_{X_i}(\bar{x}_i) :=$ máx valor $f(x_1, \ldots, \bar{x}_i, \ldots, x_n) \geq f(\bar{x}_1, \ldots, \bar{x}_i, \ldots, \bar{x}_n) = N$ ■

Definición 3.3.7 *Fijado un nivel N, llamaremos $Rango^N_i := \{\underline{x_i \in \mathbb{Z}}$ tales que $(x_i, N) \in \mathbf{P_{X_i}}\}$.*

Es decir, al analizar la proyección $\mathbf{P_{X_i}}$ restringida al nivel N, $Rango^N_i$ indica el conjunto de valores enteros para los cuales la variable x_i alcanza dicho nivel.

Proposición 3.3.8 *Sea $f : D \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ continua, D un dominio convexo y compacto. Si $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_n)$ es una solución óptima entera del problema (entera-pura o entera-mixta), N es el nivel óptimo de f y la coordenada i-ésima de la solución debe ser entera, entonces, $\bar{x}_i \in Rango_i^N$.*

Demostración:

Consecuencia de la proposición 3.3.6 y de la definición 3.3.7 ■

3.4. El Método PSA-*puro*

Para introducir el método en el caso entero-*puro*, consideremos el ejemplo anterior de tres variables –para el cual hemos calculado las proyecciones– y supongamos que los coeficientes de la función son todos números enteros. Luego, debido a que la solución óptima debe tener todas sus coordenadas enteras, se sigue que el valor óptimo de la función también será un número entero. Por lo tanto, de todos los niveles que puede alcanzar la función, sólo nos interesarán aquellos correspondientes a valores enteros.

Si estamos buscando el máximo del problema, en la figura 3.7 pueden observarse los primeros cuatro niveles enteros *candidatos* a ser el nivel óptimo de la función.

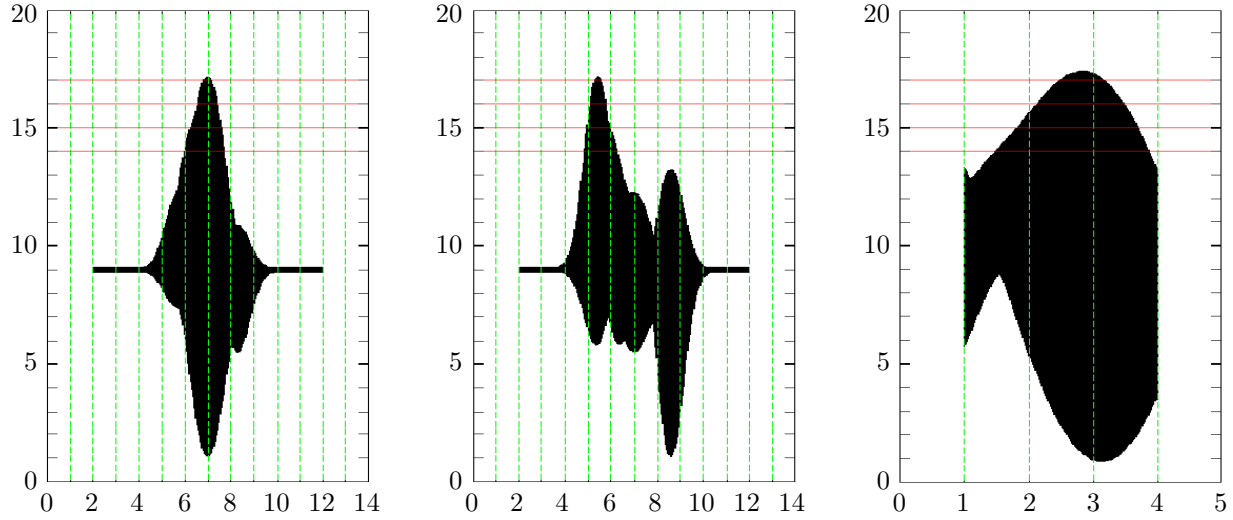

Figura 3.7: niveles 17, 16, 15 y 14.

Para buscar la solución óptima entera del problema consideremos el conjunto de proyecciones restringidas al primero de los *posibles* niveles óptimos (el 17). Si en efecto, 17 es la altura óptima de la función, se sigue, por la proposición 3.3.8, que cada una de las coordenadas de la solución óptima deben formar parte de los conjuntos $Rango_i^{17}$. Por lo tanto, de existir una solución en este nivel, debería poder formarse a partir de la información que aportan dichos conjuntos. En este caso, $Rango_1^{17} = \{7\}$, $Rango_2^{17} = \emptyset$ y $Rango_3^{17} = \{3\}$. Con lo cual, deducimos que no puede existir ninguna solución entera tal que al ser evaluada en f de como resultado 17. Conclusión: descartamos al nivel 17 como el nivel óptimo de la función y analizamos el siguiente de los *posibles* valores.

Repetimos el procedimiento sobre el nivel 16. En este caso, $Rango_1^{16} = \{7\}$, $Rango_2^{16} = \emptyset$ y $Rango_3^{16} = \{3\}$. Al igual que en el nivel anterior, concluimos que no puede existir ninguna solución entera en este nivel.

Consideramos el nivel 15. Ahora $Rango_1^{15} = \{7\}$, $Rango_2^{15} = \{5, 6\}$ y $Rango_3^{15} = \{2, 3\}$. En este caso, deberíamos calcular todos los posibles candidatos a solución ($X = (7, 5, 2)$, $X = (7, 5, 3)$, $X = (7, 6, 2)$ y $X = (7, 6, 3)$) y verificar si alguno de ellos es factible y si al ser evaluado en f da como resultado 15. De ser así, estaríamos frente al óptimo del problema (porque ya vimos que en los niveles superiores no existe ninguna solución factible). Ahora bien, como en general el número de candidatos producidos de esta manera resulta exponencial, intentaremos simplificar un poco el problema. De existir una solución óptima en el nivel actual, como $|Rango_1^{15}| = 1$, sabemos que debe ser de la forma "$X = (7, -, -)$". Esto nos permite reemplazar el valor $x_1 = 7$ en la función original y obtener una nueva función, $f(7, x_2, x_3)$, sobre la cual podemos volver a calcular $\mathbf{P_{X_2}}$ y $\mathbf{P_{X_3}}$ (siempre mirando el nivel 15). Al actualizar los conjuntos $Rango_2^{15}$ y $Rango_3^{15}$ para la nueva función, se da una de las siguientes alternativas:

(*a*) $|Rango_i^{15}| = 1$ para $i = 2, 3$. Podemos completar nuestro candidato a solución, $X = (7, -, -)$, con los valores que proveen los conjuntos $Rango_2^{15}$ y $Rango_3^{15}$. Si el candidato así generado es factible y al ser evaluado en f da como resultado 15, entonces hallamos el óptimo del problema. En caso contrario, como este candidato es el único posible, deducimos que 15 no es la altura óptima.

(*b*) $\exists\ i\ /\ |Rango_i^{15}| = 0$. Concluimos que el nivel 15 no es óptimo.

(*c*) $\exists\ i\ /\ |Rango_i^{15}| = 1$. Reemplazamos en "$f(7, x_2, x_3)$" y en "$X = (7, -, -)$" el valor de la variable cuyo rango tiene cardinal 1 y volvemos a calcular la proyección de la variable restante.

(*d*) $|Rango_i^{15}| > 1\ \forall i$. En este caso, no hay ninguna variable que pueda asumir un único valor entero; entonces trabajamos de la siguiente manera. Elegimos una de las dos variables bajo algún criterio, por ejemplo x_2. Reemplazamos en la función y en el candidato "X" que estamos construyendo el último (o el primero) de los valores del rango, resulta: $f(7, 6, x_3)$, $X = (7, 6, -)$. Y calculamos nuevamente $\mathbf{P_{X_3}}$ y $Rango_3^{15}$. Guardamos en el conjunto "*Pendientes*" los datos de la función evaluada en el valor de x_2 que no fue utilizado; esto lo hacemos para analizar dicho problema si el espacio de soluciones generado por el primer valor no arroja ninguna solución óptima. En este caso, $Pendientes = \{f(7, 5, x_3)\}$.

De esta manera, el algoritmo recorre cada uno de los *posibles* niveles óptimos y genera, a partir de la información que aportan las proyecciones, los distintos *candidatos* a solución.

La primera diferencia que encontramos entre nuestro algoritmo y el resto de los algoritmos utilizados para resolver PPLE, es que PSA genera los candidatos a solución en términos del valor de la función. Recordemos que los algoritmos de tipo *Branch-and-Bound* y *Branch-and-Cut* los fabrican en base a las coordenadas fraccionarias de la solución óptima de la relajación lineal. Otra diferencia importante es que PSA no agrega restricciones adicionales a la formulación del problema. Por último, al trabajar en base a la información que aportan las proyecciones, PSA utiliza esa información para reducir, sistemáticamente, el número de variables del problema. Cosa que no sucede, como regla general, en el resto de los algoritmos.

El esquema del algoritmo es el siguiente:

$Pendientes = \{\ \}$, $X = (-, \ldots, -)$.

0. Inicialización. Calcular el conjunto de proyecciones del problema y el primer nivel entero "N".

1. Inspección. Analizar las proyecciones restringidas al nivel actual y calcular los conjuntos $Rango_i^N\ \forall i = 1, \ldots, n$.

(*a*) Si $|Rango_i^N| = 1\ \forall i$ (*todas las variables toman un único valor entero*), ir al *Paso 2.*

(*b*) Si $\exists\ i\ /\ |Rango_i^N| = 0$ (*hay al menos una variable a la que no se le puede asignar ningún valor entero*), analizar el último problema agregado en *Pendientes* –y quitarlo del conjunto– o, si $Pendientes = \emptyset$, bajar un nivel: $N = N - 1$. Actualizar el vector X e ir al *Paso 1.*

(*c*) Si $\exists\ i\ /\ |Rango_i^N| = 1$ (*hay al menos una variable obligada a tomar un único valor entero*), ir al *Paso 3.*

(*d*) Si $|Rango_i^N| > 1\ \forall i$ (*todas las variables pueden tomar al menos dos valores enteros*), ir al *Paso 4.*

2. Comprobación. Completar el candidato "X" con los valores sugeridos por las proyecciones (definir $x_i = Rango_i^N\ \forall i\ /\ |Rango_i^N| = 1$), y comprobar si es un punto factible y si al evaluarlo en f coincide con el nivel actual.

(*e*) "*Si el candidato es factible y al evaluarlo en f coincide con el nivel actual*", fin del algoritmo (se alcanzó el óptimo del problema).

(*f*) "*Si el candidato es factible, pero al ser evaluado en f está por debajo de nivel actual*", guardar esa solución si supera la mejor solución encontrada hasta el momento y, a continuación, analizar el último problema agregado en *Pendientes* –quitándolo del conjunto– o, si $Pendientes = \emptyset$, bajar un nivel: $N = N - 1$. Actualizar X e ir al *Paso 1.*

(*g*) "*Si el candidato es infactible*", analizar el último problema agregado en *Pendientes* –y quitarlo del conjunto– o, en su defecto, bajar un nivel: $N = N - 1$. Actualizar X e ir al *Paso 1.*

3. Reducción. Reemplazar las variables que toman un único valor entero en el problema actual y en el candidato a solución "X" que se está construyendo. Calcular las nuevas proyecciones y regresar al *Paso 1.*

4. Elección. Escoger, bajo algún criterio, una de las variables del problema actual. Reemplazar cada uno de los valores del rango en dicho problema y guardar los subproblemas generados de esta manera en el conjunto *Pendientes*. Considerar el último subproblema agregado en *Pendientes*, quitarlo del conjunto, y actualizar X. Calcular las proyecciones y regresar al *Paso 1.*

3.5. Validez del Algoritmo

Proposición 3.5.1 *Consideremos un problema general de programación lineal entera-pura (con coeficientes enteros), llamémoslo "*PLE*", y supongamos que las restricciones determinan una región no vacía y acotada.*

$$
\begin{array}{ll}
\text{PLE:} & \\
\textit{maximizar} & f = c_1x_1 + \cdots + c_nx_n \\
\textit{sujeta a} & a_{11}x_1 + \cdots + a_{1n}\ x_n \leq b_1 \\
 & \qquad\qquad\qquad\qquad\ \vdots \\
 & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\
 & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, \ldots, n
\end{array}
$$

Veamos que el algoritmo "PSA-puro" converge a una solución óptima y además lo hace en un número finito de pasos.

Demostración:

Por inducción en el número de variables.

Si $n = 1$, nuestro problema es

$$
\begin{array}{ll}
\text{PLE:} & \\
\text{maximizar} & f = c_1x_1 \\
\text{sujeta a} & a_{11}x_1 \leq b_1 \\
 & \qquad\ \vdots \\
 & a_{m1}x_1 \leq b_m \\
 & x_1 \geq 0,\ x_1 \in \mathbb{Z}
\end{array}
$$

Llamaremos $[r_1, r_2]$ al intervalo que queda determinado al intersecar el conjunto de restricciones lineales y supondremos, sin pérdida de generalidad, que $c_1 > 0$. Luego, el óptimo del problema se alcanza en el punto $x_1 = \lfloor r_2 \rfloor$ con nivel óptimo $c_1\lfloor r_2 \rfloor$.

Inicio del algoritmo, ***Paso 0.***

En este caso, como el problema es unidimensional, la proyección coincide con el gráfico de la función: $\mathbf{P_{X_1}}$ es un segmento de recta de pendiente c_1, con ordenada al origen 0, definida en el intervalo $[r_1, r_2]$. Por lo tanto, el primer nivel entero que debemos analizar es $\lfloor c_1 r_2 \rfloor$.

Observación: siempre que el algoritmo examine un nivel superior a "$c_1\lfloor r_2 \rfloor$" –el nivel óptimo–, a lo sumo deberá probar un número *finito* de candidatos (ya que el rango de valores de cada una de las variables es acotado) antes de concluir que el intervalo analizado no es óptimo. Por lo tanto, podemos suponer que el algoritmo llega al nivel "$c_1\lfloor r_2 \rfloor$" en un número finito de operaciones.

Nivel $c_1\lfloor r_2 \rfloor$.

Paso 1. Dado que la proyección es un segmento de recta, a x_1 le corresponde un único valor entero para el nivel actual: $Rango_1^{c_1\lfloor r_2 \rfloor} = \{\lfloor r_2 \rfloor\}$ (caso (a)).

Paso 2. $x_1 = \lfloor r_2 \rfloor$ es un candidato factible y al ser evaluado en f coincide con el nivel actual. Fin del algoritmo.

Conclusión: el método encuentra el óptimo del problema en un número finito de iteraciones.

Hipótesis inductiva: suponiendo que el algoritmo resuelve todas las instancias de hasta $n-1$ variables en un número finito de pasos, probemos el caso de n variables.

Ahora nuestro problema es

$$
\begin{array}{ll}
\text{PLE:} & \\
\text{maximizar} & f = c_1x_1 + \cdots + c_nx_n \\
\text{sujeta a} & a_{11}x_1 + \cdots + a_{1n}\ x_n \leq b_1 \\
 & \qquad\qquad\qquad\qquad\ \vdots \\
 & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\
 & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, \ldots, n
\end{array}
$$

Sean $\bar{X} = (e_1, \ldots, e_n)$ una solución óptima entera de PLE y N el nivel óptimo de f.

Inicio del algoritmo, ***Paso 0.***

Supongamos que el método calcula las n proyecciones del problema y el nivel inicial. Luego, analiza uno a uno los niveles enteros –en orden descendente– empezando por el más grande.

Observación: al igual que en el problema anterior de una variable, podemos suponer que el algoritmo llega al nivel óptimo en un número finito de operaciones.

Nivel N.

Paso 1. Al examinar las proyecciones restringidas al nivel actual, ocurre una de las siguientes alternativas.

(a) $|Rango_i^N| = 1\ \forall i$. Por la proposición 3.3.8, $Rango_i^N = \{e_i\}\ \forall i$.

Paso 2. "$X = (e_1, \ldots, e_n)$" es un candidato factible, y al ser evaluado en f coincide con el nivel actual; hallamos el óptimo del problema. Fin del algoritmo.

(b) $\exists\ i\ /\ |Rango_i^N| = 0$. Absurdo, la proposición 3.3.8 nos asegura que, para este nivel, existe al menos un valor para cada variable.

(*c*) $\exists\ i\ /\ |Rango_i^N| = 1$. Supongamos, sin pérdida de generalidad, que esta variable es x_n. Luego, por la proposición 3.3.8, $Rango_n^N = \{e_n\}$.

Paso 3. Actualizamos el candidato a solución: $X = (-, \ldots, -, e_n)$. Y reemplazamos el valor de x_n en PLE; como resultado obtenemos "PLE-*reducido*", un problema de $n-1$ variables.

PLE-*reducido*:
maximizar $\quad f^* = c_1x_1 + \cdots + c_{n-1}x_{n-1} + c_ne_n$

$$\text{sujeta a} \quad \begin{aligned} a_{11}x_1 + \cdots + a_{1n-1}\,x_{n-1} &\leq b_1 - a_{1n}e_n \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mn-1}x_{n-1} &\leq b_m - a_{mn}e_n \end{aligned}$$

$$x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, \ldots, n-1$$

Observación: si $\bar{X} = (e_1, \ldots, e_n)$ es una solución óptima de PLE, entonces, $\hat{X} = (e_1, \ldots, e_{n-1})$ es una solución óptima de PLE-*reducido*. En efecto, si $(d_1, \ldots, d_{n-1})$ fuera una solución factible de PLE-*reducido*, tal que $f^*(d_1, \ldots, d_{n-1}) > f^*(e_1, \ldots, e_{n-1})$, entonces, $(d_1, \ldots, d_{n-1}, e_n)$ sería una solución factible de PLE tal que $f(d_1, \ldots, d_{n-1}, e_n) > f(e_1, \ldots, e_{n-1}, e_n)$, absurdo.

Paso 1. El problema que resulta de la operación anterior posee $n-1$ variables y nivel óptimo $f^*(e_1, \ldots, e_{n-1}) = N$. Por HI, el algoritmo devuelve una solución óptima del problema reducido en un número finito de pasos. Agregando el valor e_n a dicha solución, conseguimos el óptimo de PLE. Fin del algoritmo.

(*d*) $|Rango_i^N| > 1\ \forall i$.

Paso 4. Elegimos una de las variables de problema actual, por ejemplo x_n. Reemplazamos cada uno de los valores del $Rango_n^N$ en dicho problema (entre estos valores debe figurar e_n), y guardamos los subproblemas así generados en el conjunto *Pendientes*. Consideramos el último problema agregado en *Pendientes*, lo quitamos del conjunto, y comenzamos a analizarlo.

Podemos suponer, en el peor de los casos, que el algoritmo analiza todos los subproblemas correspondientes a valores de $x_n \neq e_n$ sin hallar ninguna solución óptima. Luego, después de un número finito de operaciones, el algoritmo analiza el subproblema correspondiente al valor $x_n = e_n$.

Paso 1. El subproblema generado a partir del valor "e_n" posee $n-1$ variables y nivel óptimo N. Por HI, el algoritmo encuentra una solución óptima del problema reducido en un número finito de pasos. Agregando el valor e_n a dicha solución, conseguimos el óptimo de PLE ■

3.6. El Método PSA-*mixto*

En esta sección presentaremos una nueva versión del algoritmo PSA que nos permitirá trabajar sobre problemas enteros-*mixtos*.

Al intentar reproducir las ideas utilizadas en el caso entero-*puro* para esta otra clase de problemas, nos encontramos con la siguiente dificultad: *sobre problemas mixtos ya no podemos suponer que el óptimo se ubique en un nivel entero.* Con lo cual, el conjunto de *posibles* niveles óptimos –que debemos analizar– está compuesto por infinitos elementos. Para solucionar este problema incorporamos la siguiente definición.

Definición 3.6.1 *Diremos que dos niveles, N_1 y N_2, forman parte del mismo* intervalo de niveles, *si $Rango_i^{N_1} = Rango_i^{N_2}$ para todas las variables x_i enteras.*

Esta definición nos permitirá trabajar con infinitos niveles de manera simultánea y, de esa manera, reducir nuestro problema a analizar un número *finito* de casos.

Notación *Notaremos "$Rango_j^I$" al rango de valores enteros que puede asumir la variable x_j para cualquiera de los niveles que componen el intervalo de niveles "I".*

Para motivar el algoritmo en el caso entero-*mixto*, consideremos nuevamente el ejemplo de tres variables y supongamos que estamos buscando una solución *mixta* "$X = (x_1, x_2, x_3)$" de la forma: $x_1, x_2 \in \mathbb{Z}$ y $x_3 \in \mathbb{R}$. Como en este caso solo estamos interesados en buscar valores enteros para las dos primeras coordenadas de la solución, nos limitaremos a calcular, únicamente, las proyecciones correspondientes a esas variables.

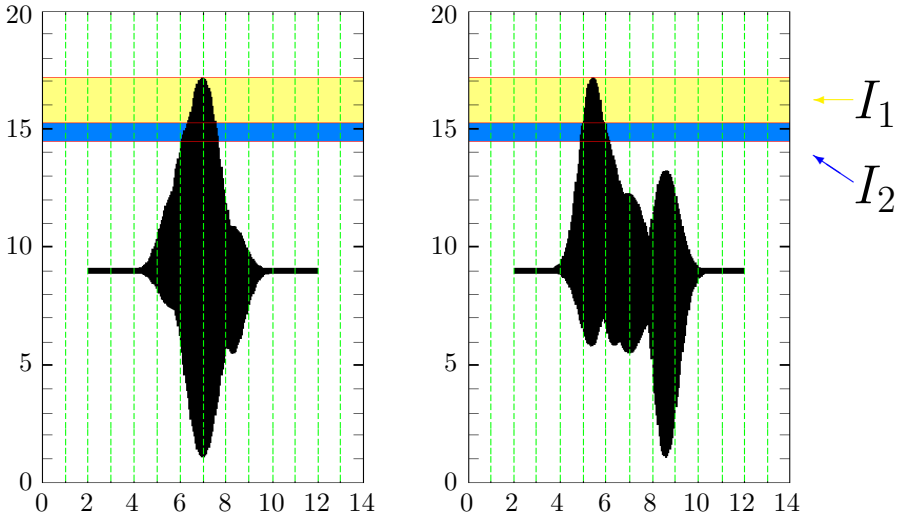

Figura 3.8: intervalos de niveles I_1 e I_2.

En la figura 3.8, se exhiben los dos primeros *intervalos de niveles* en los que pueden ser descompuestas las proyecciones $\mathbf{P_{X_1}}$ y $\mathbf{P_{X_2}}$. Observar que, para cualquiera de los niveles que componen esos intervalos, el rango de valores de las variables enteras siempre es el mismo.

Para todos los niveles "N" comprendidos en el primer intervalo: $Rango_1^N = \{7\}$ y $Rango_2^N = \emptyset$. Con la nueva notación: $Rango_1^{I_1} = \{7\}$ y $Rango_2^{I_1} = \emptyset$. Luego, no puede existir ninguna solución *mixta* –cuya segunda coordenada sea entera– tal que al ser evaluada en f dé como resultado alguno de los niveles comprendidos en el intervalo I_1. Conclusión: I_1 no contiene al nivel óptimo del problema. Analizamos el siguiente intervalo de niveles.

En el caso del intervalo I_2: $Rango_1^{I_2} = \{7\}$ y $Rango_2^{I_2} = \{5, 6\}$. Luego, dentro de este intervalo tenemos dos *candidatos* a solución: $X = (7, 5, -)$ y $X = (7, 6, -)$. Evaluando ambos puntos en la función original, resultan dos problemas de optimización *continua* de una variable cada uno (en el caso de estar resolviendo un PPLE-*mixta*, el problema resultante es de programación lineal). Si alguno de estos subproblemas alcanza su valor máximo dentro del intervalo I_2, entonces, completando el candidato "X" con el valor óptimo que provee el problema *continuo*, obtenemos el óptimo del problema *mixto*. Ahora bien, como en general el número de candidatos producidos de esta manera resulta exponencial, volvemos a trabajar como lo hicimos en el caso entero-*puro*. Consideramos aquellas variables que pueden asumir un único valor entero; reemplazamos dichos valores en la función y en el candidato a solución que estamos construyendo; y volvemos a calcular las proyecciones de las variables restantes (sólo de las variables enteras). En nuestro caso, la única variable a la que se le puede asignar un único valor entero es x_1, con valor 7. Reemplazando este valor en la función y en el candidato a solución que estamos construyendo, resulta: $f(7, x_2, x_3),\ x_2 \in \mathbb{Z},\ x_3 \in \mathbb{R}$ y $X = (7, -, -)$. Calculando nuevamente $\mathbf{P_{X_2}}$, y actualizando el conjunto $Rango_2^{I_2}$, se da una de las siguientes alternativas[1]:

(a) $|Rango_2^{I_2}| = 1$. Podemos completar nuestro candidato a solución, $X = (7, -, -)$, con el valor que provee el conjunto $Rango_2^{I_2}$. De esta manera, logramos reducir el problema original a un problema de optimización *continua* de una sola variable. Si este nuevo problema alcanza su valor máximo en alguno de los niveles comprendidos en el intervalo I_2, entonces, completando nuestro candidato "X" con el valor óptimo que provee el problema *continuo*, obtenemos el óptimo del problema *mixto*. En caso contrario, como este candidato es el único posible, deducimos que el intervalo I_2 no contiene al nivel óptimo (y, por consiguientes, a la solución óptima).

(b) $|Rango_2^{I_2}| = 0$. Concluimos que el intervalo I_2 no contiene al nivel óptimo.

(d) $|Rango_2^{I_2}| > 1$. En esta situación, no nos queda otra alternativa que analizar los dos subproblemas "$f(7, 5, -)$" y "$f(7, 6, -)$". Al igual que en el caso entero-*puro*, comenzamos trabajando con uno de los subproblemas y guardamos la información del problema restante en el conjunto "*Pendientes*". Si alguno de estos dos subproblemas alcanza su valor máximo dentro del intervalo I_2, entonces estamos frente al óptimo del problema *mixto*[2]. En caso contrario, concluimos que I_2 no contiene al nivel óptimo.

De esta manera, el algoritmo recorre cada uno de los intervalos de niveles y genera, a partir de la información que aportan las proyecciones, los distintos candidatos a solución. El procedimiento anterior se repite hasta alcanzar la solución óptima del problema.

[1] Eventualmente, en esta operación, el intervalo sobre el que estamos trabajando deberá ser dividido en intervalos más pequeños, "*refinado*". Esto se debe a que el rango de valores de cada una de las variables enteras se puede ver alterado al calcular el nuevo conjunto de proyecciones. En situaciones como éstas, descartamos el intervalo que estamos analizando y continuamos trabajando con el subintervalo que contiene a los niveles de mayor altura.

[2] Observar que, llegado este punto, se deben analizar *todos* los subproblemas contenidos en el conjunto "*Pendientes*" antes de hacer alguna afirmación acerca de la optimalidad de una solución encontrada.

Esquema del algoritmo:

$Pendientes = \{\ \}$, $X = (-, \ldots, -)$.

0. Información inicial. Calcular únicamente las proyecciones de las variables enteras y el primer intervalo de niveles "I".

1. Inspección. Analizar las proyecciones restringidas al intervalo actual y calcular $Rango_i^I$ para todas las variables enteras x_i.

- (a) Si $|Rango_i^I| = 1\ \forall i$, ir al *Paso 2*.
- (b) Si $\exists\ i\ /\ |Rango_i^I| = 0$, ir al *Paso 3*.
- (c) Si $\exists\ i\ /\ |Rango_i^I| = 1$, ir al *Paso 4*.
- (d) Si $|Rango_i^I| > 1\ \forall i$, ir al *Paso 5*.

2. Comprobación. Completar el candidato "X" a partir de los valores sugeridos por las proyecciones (definir $x_i = Rango_i^I\ \forall i\ /\ |Rango_i^I| = 1$). Evaluar este candidato en el problema original y resolver el problema de optimización *continua* resultante (en el caso de estar resolviendo un PPLE-*mixta*, el problema resultante es de programación lineal).

- (e) "*Si el óptimo del problema reducido se alcanza en uno de los niveles contenidos en el intervalo actual*". Asignar a las variables no enteras del vector "X" las coordenadas de la solución óptima del problema de optimización *continua*. Guardar X si es factible y supera la mejor solución encontrada hasta ese momento. Ir al *Paso 3*.
- (f) "*Si el óptimo del problema reducido no pertenece al intervalo actual*". Asignar a las variables no enteras del vector "X" las coordenadas de la solución óptima del problema de optimización *continua*. Guardar X si es factible y supera la mejor solución encontrada hasta el momento. Ir al *Paso 3*.
- (g) "*Si el problema reducido es infactible*", ir al *Paso 3*.

3. Pendientes. Analizando el conjunto *Pendientes*, se determina la acción a seguir.

- Si $Pendientes \neq \emptyset$, considerar el último subproblema agregado a este conjunto y quitarlo del mismo. Actualizar X e ir al *Paso 1*.
- Si $Pendientes = \emptyset$ y ya se conoce una solución factible cuyo valor objetivo pertenece al intervalo actual, fin del algoritmo. El óptimo del problema *mixto* es la última solución factible guardada.
- En caso contrario, bajar al siguiente intervalo de niveles (por abuso de notación, volveremos a llamarlo "I"). Actualizar X e ir al *Paso 1*.

4. Reducción. Reemplazar las variables que toman un único valor entero en el problema actual. Calcular las nuevas proyecciones (sólo las proyecciones de las variables enteras) y comprobar si el intervalo que está siendo analizado debe ser refinado. Ir al *Paso 1*.

5. Elección. Escoger –bajo algún criterio– una de las variables enteras y reemplazar cada uno de los valores del rango en el problema actual. Guardar los subproblemas generados de esta manera en el conjunto *Pendientes*. Considerar el último subproblema agregado en *Pendientes* y quitarlo del conjunto. Actualizar el vector X, calcular las proyecciones, refinar el intervalo de ser necesario, y regresar al *Paso 1*.

Capítulo 4

Problema de la Mochila No Acotado

A partir de este capítulo y en adelante, aplicaremos el método construido en el capítulo anterior para resolver PPLE-*pura*. En esta tesis nos concentraremos sobre problemas del tipo *mochila*, pero con algún esfuerzo adicional, nuestro desarrollo puede ser extendido a problemas más generales. Empecemos por un ejemplo de tipo *Mochila No Acotado* (UKP).

$$\begin{aligned} \text{maximizar} \quad & f = 2x_1 + 5x_2 + x_3 + 8x_4 \\ \text{sujeta a} \quad & 79x_1 + 53x_2 + 45x_3 + 45x_4 \leq 178 \\ & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, 2, 3, 4 \end{aligned}$$

Inicio del algoritmo, ***Paso 0.***

En este caso tenemos que calcular cuatro proyecciones (una por cada variable); para hacer esto, recordemos, primero debemos encontrar las proyecciones superior "$P^{sup}_{X_i}(x_i)$" e inferior "$P^{inf}_{X_i}(x_i)$" y después considerar la región encerrada entre ambas curvas. A modo de ejemplo, mostremos cómo se calcula $\mathbf{P_{X_1}}$.

Por definición, $P^{sup}_{X_1}(x_1) :=$ máx valor $\quad 2x_1 + 5x_2 + x_3 + 8x_4$

$$\begin{aligned} \text{sujeta a} \quad & 79x_1 + 53x_2 + 45x_3 + 45x_4 \leq 178 \\ & x_i \geq 0,\ x_i \in \mathbb{R},\ i = 1, 2, 3, 4 \end{aligned}$$

$\longleftarrow$ ¡Atención! relajamos la condición de integridad.

donde estamos pensando que la variable x_1 toma un valor fijo y que el resto de las variables están *libres* dentro del dominio de definición de $f(x_1, \dots)$.

Si comparamos los cocientes $\dfrac{p_i}{w_i}$ de cada variable:

	x_2	x_3	x_4
$\dfrac{p_i}{w_i}$	0.0943	0.0222	0.1778

advertimos que el óptimo se alcanza en el punto $\left(x_1, 0, 0, \frac{178 - 79x_1}{45}\right)$. Evaluando este punto en la función, obtenemos el valor que estamos buscando:

$$P^{sup}_{X_1}(x_1) = -12{,}0444x_1 + 31{,}6444, \quad \text{con } x_1 \in \left[0, \frac{178}{79}\right].$$

De la misma manera averiguamos la proyección inferior sobre x_1:

Por definición, $P^{inf}_{X_1}(x_1) :=$ mín valor $\quad 2x_1 + 5x_2 + x_3 + 8x_4$

$$\text{sujeta a} \quad 79x_1 + 53x_2 + 45x_3 + 45x_4 \leq 178$$

$$x_i \geq 0,\ x_i \in \mathbb{R},\ i = 1, 2, 3, 4$$

Como a x_1 la estamos pensando fija y los coeficientes de la función objetivo son todos valores positivos, entonces, el mínimo se alcanza en el punto $(x_1, 0, 0, 0)$. Evaluando en f, obtenemos:

$$P^{inf}_{X_1}(x_1) = 2x_1, \quad \text{con } x_1 \in \left[0, \frac{178}{79}\right].$$

Repitiendo el procedimiento anterior se pueden calcular el resto de las proyecciones:

$$P^{sup}_{X_2}(x_2) = \text{- } 4.4222x_2 + 31.6444 \quad \text{y} \quad P^{inf}_{X_2}(x_2) = 5x_2, \quad \text{con } x_2 \in \left[0, \frac{178}{53}\right]$$

$$P^{sup}_{X_3}(x_3) = \text{- } 7.0000x_3 + 31.6444 \quad \text{y} \quad P^{inf}_{X_3}(x_3) = x_3, \quad \text{con } x_3 \in \left[0, \frac{178}{45}\right]$$

$$P^{sup}_{X_4}(x_4) = 3.7547x_4 + 16.7925 \quad \text{y} \quad P^{inf}_{X_4}(x_4) = 8x_4, \quad \text{con } x_4 \in \left[0, \frac{178}{45}\right]$$

A continuación, graficamos el conjunto de proyecciones junto con los cuatro primeros niveles enteros *candidatos* a ser el nivel óptimo del problema.

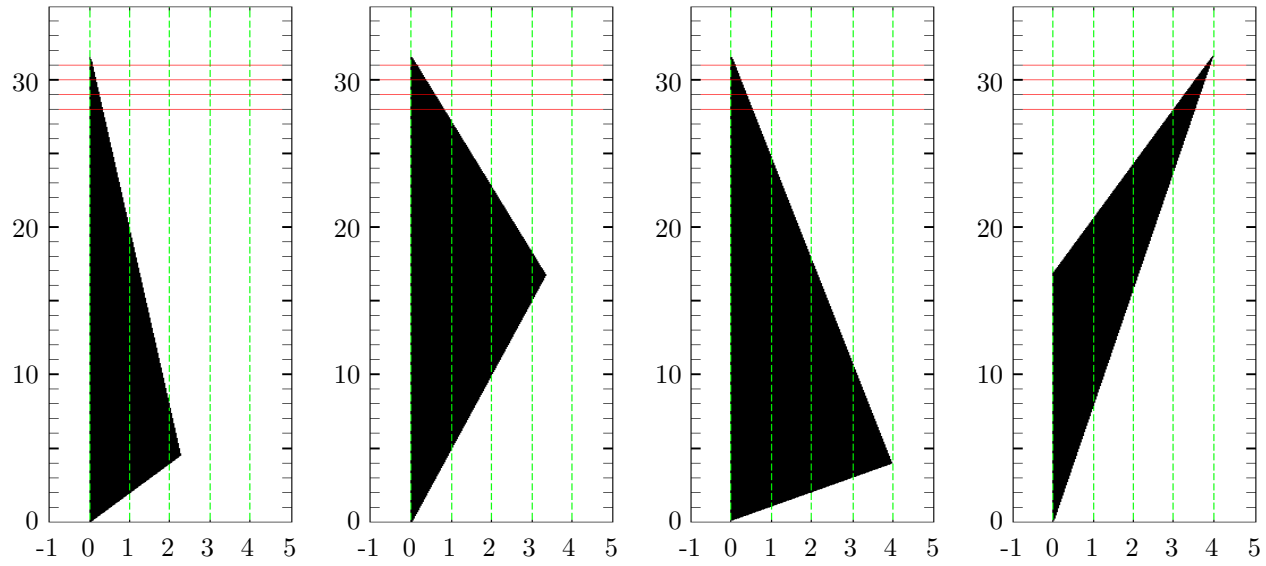

Figura 4.1: niveles 31, 30, 29 y 28.

A partir de la figura 4.1, el primer nivel entero que debemos analizar es el 31.

Nivel 31.

Paso 1. Observando las proyecciones "$\mathbf{P_{X_i}}$" restringidas al nivel 31, advertimos que: $Rango_1^{31} = \{0\}$, $Rango_2^{31} = \{0\}$, $Rango_3^{31} = \{0\}$ y $Rango_4^{31} = \emptyset$ (caso (b)). Es decir, no puede existir ningún punto con todas sus coordenadas enteras tal que al ser evaluado en f de como resultado 31. Conclusión: el óptimo no se ubica en esta altura. Dado que $Pendientes = \emptyset$, bajamos un nivel.

Niveles 30 y 29.

Paso 1. Los analizamos en conjunto porque en ambos casos ocurre exactamente lo mismo. Al igual que en el nivel anterior, en este caso: $Rango_1^N = \{0\}$, $Rango_2^N = \{0\}$, $Rango_3^N = \{0\}$ y $Rango_4^N = \emptyset$, para $N = 30$ y 29 (caso (b)). Luego, no existe ninguna solución entera que alcance los niveles 30 o 29 al ser evaluada en f. Como el conjunto *Pendientes* no sufrió modificaciones, continuamos con el próximo nivel.

Nivel 28.

Paso 1. Recién en este nivel tenemos un primer candidato a solución. Al analizar las proyecciones restringidas al nivel actual, obtenemos: $Rango_1^{28} = \{0\}$, $Rango_2^{28} = \{0\}$, $Rango_3^{28} = \{0\}$ y $Rango_4^{28} = \{3\}$ (caso (a)).

Paso 2. $X = (0, 0, 0, 3)$ es el único candidato que resulta de analizar el conjunto de proyecciones; es una solución factible, sin embargo, al ser evaluado en f da como resultado 24. Luego, deducimos que 28 no es la altura óptima.

Antes de pasar al siguiente nivel (pues el conjunto *Pendientes* sigue siendo vacío), guardamos la solución encontrada por si no conseguimos una mejor antes de llegar al nivel 24.

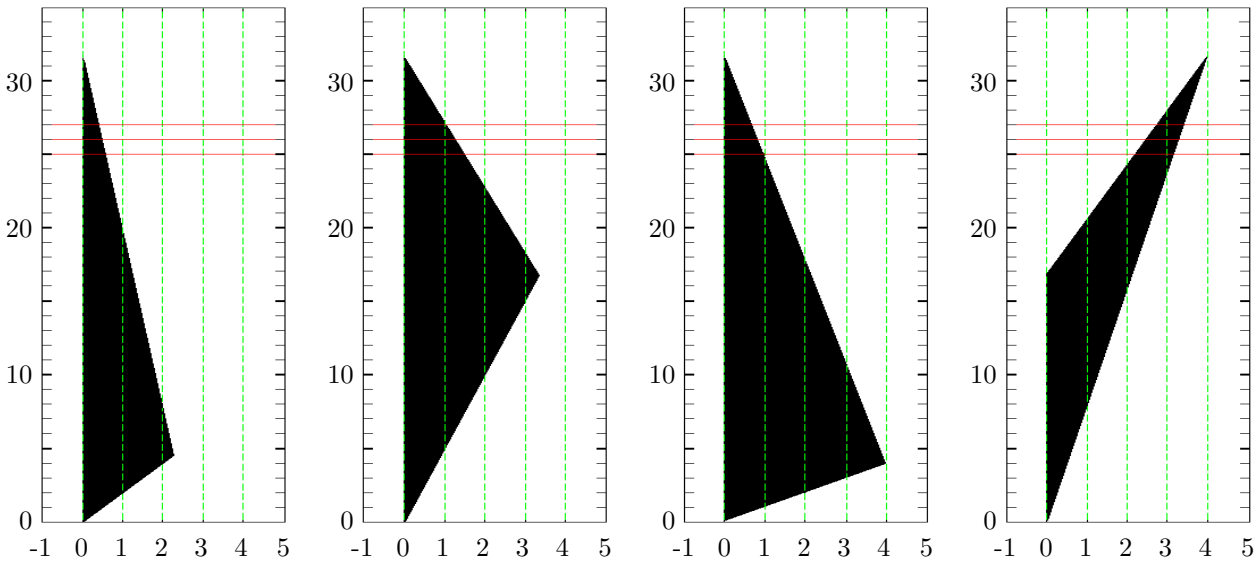

Figura 4.2: niveles 27, 26 y 25.

Nivel 27.

Paso 1. Observando el conjunto de proyecciones restringidas al nivel 27, tenemos: $Rango_1^{27} = \{0\}$, $Rango_2^{27} = \{0, 1\}$, $Rango_3^{27} = \{0\}$ y $Rango_4^{27} = \{3\}$ (caso(c)). Luego, sobre el nivel actual se presentan dos candidatos a solución: $X = (0, 0, 0, 3)$ y $X = (0, 1, 0, 3)$. Para no analizar uno a uno estos candidatos –ya que, en general, vamos a tener una cantidad exponencial de ellos– trabajamos como indica el *Paso 3*.

Paso 3. Actualizamos nuestro candidato a solución: $X = (0, -, 0, 3)$. Y reemplazamos en el problema actual las variables que toman un único valor (para luego volver a calcular $\mathbf{P_{X_2}}$).

Problema original:

$$\begin{aligned} \text{maximizar} \quad & f = 2x_1 + 5x_2 + x_3 + 8x_4 \\ \text{sujeta a} \quad & 79x_1 + 53x_2 + 45x_3 + 45x_4 \leq 178 \\ & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, 2, 3, 4 \end{aligned}$$

Reemplazando los valores $x_1 = 0$, $x_3 = 0$ y $x_4 = 3$, se deriva el siguiente problema reducido:

$$\begin{aligned} \text{maximizar} \quad & 5x_2 + 24 \\ \text{sujeta a} \quad & 53x_2 \leq 43 \\ & x_2 \geq 0,\ x_2 \in \mathbb{Z} \end{aligned}$$

Llegada esta instancia, hagamos un comentario antes de calcular la proyección: si el nuevo problema tiene una solución entera en el nivel 27, entonces el problema original también posee una solución factible en ese nivel. La solución del problema original se obtiene completando la solución parcial, $X = (0, -, 0, 3)$, con el valor de x_2 que proporciona el problema reducido.

Proyección $\mathbf{P_{X_2}}$:

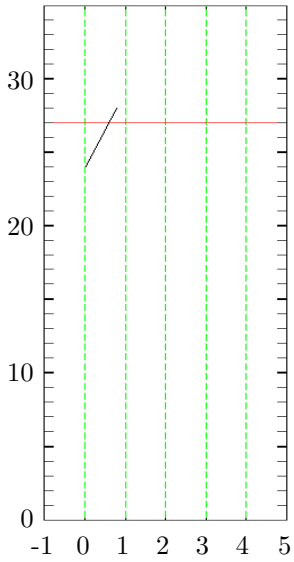

Figura 4.3: nivel 27, problema reducido.

Paso 1. Examinando la figura 4.3, advertimos que: $Rango_2^{27} = \emptyset$ (caso(b)). Por lo tanto, no existe ningún valor entero de x_2 con el cual f pueda alcanzar el nivel 27. Luego, el problema reducido no posee ninguna solución entera en este nivel, por ende, tampoco la tiene el problema original. Dado que $Pendientes = \emptyset$, bajamos un nivel.

Niveles 26 y 25.

Ocurre exactamente lo mismo que en el nivel 27. Continuamos.

Nivel 24.

No es necesario analizar este nivel, ya que: a) conocemos una solución factible que llega a esta altura y b) vimos que en los niveles superiores no hay ninguna otra solución factible. Conclusión: $X = (0, 0, 0, 3)$ es una solución óptima.

Sin embargo, y no obstante haber alcanzado una solución óptima, analizaremos este nivel porque aparece una situación no contemplada hasta el momento (tendremos que aplicar el *Paso 4* del algoritmo).

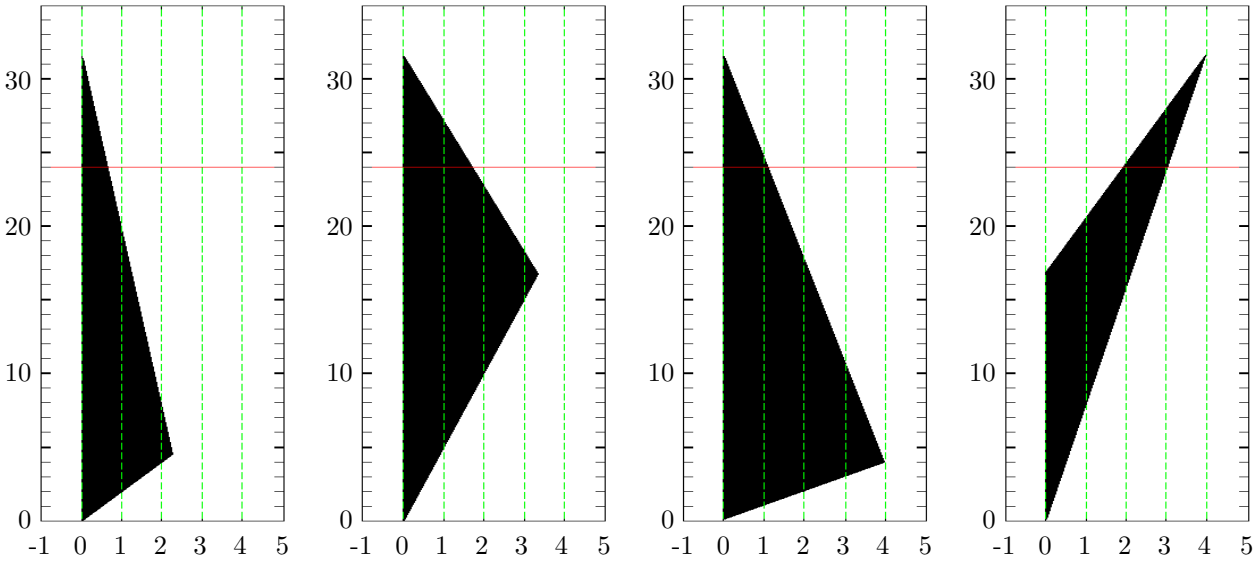

Figura 4.4: nivel 24.

Paso 1. Observando las proyecciones restringidas al nivel 24: $Rango_1^{24} = \{0\}$, $Rango_2^{24} = \{0, 1\}$, $Rango_3^{24} = \{0, 1\}$ y $Rango_4^{24} = \{2, 3\}$ (caso (c)).

Paso 3. Actualizamos el candidato a solución: $X = (0, -, -, -)$. Y reemplazamos el valor de x_1 en el problema original; obtenemos el siguiente problema reducido:

$$\begin{aligned} \text{maximizar} \quad & 5x_2 + x_3 + 8x_4 \\ \text{sujeta a} \quad & 53x_2 + 45x_3 + 45x_4 \leq 178 \\ & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 2, 3, 4 \end{aligned}$$

Como achicamos el problema, podemos volver a calcular las proyecciones –siempre restringidas al nivel actual–. En este caso, al calcular las proyecciones del problema reducido, nuevamente obtenemos las proyecciones 2, 3 y 4 del problema original (figura 4.4).

Paso 1. $Rango_2^{24} = \{0, 1\}$, $Rango_3^{24} = \{0, 1\}$ y $Rango_4^{24} = \{2, 3\}$ (caso (d)).

Paso 4. Elegimos una variable, por ejemplo x_4 $\left(\text{es la de mayor relación } \frac{p_i}{w_i}\right)$, y reemplazamos cada uno de los valores del rango en el problema anterior. Guardamos el primero de los subproblemas en el conjunto *Pendientes* y continuamos trabajando con el subproblema que resulta de reemplazar el valor $x_4 = 3$. Ahora $X = (0, -, -, 3)$.

El subproblema que se deriva de elegir "$x_4 = 3$" es el siguiente:

$$\begin{aligned} \text{maximizar} \quad & 5x_2 + x_3 + 24 \\ \text{sujeta a} \quad & 53x_2 + 45x_3 \leq 43 \\ & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 2, 3 \end{aligned}$$

Calculando las proyecciones $\mathbf{P_{X_2}}$ y $\mathbf{P_{X_3}}$ del problema reducido, obtenemos:

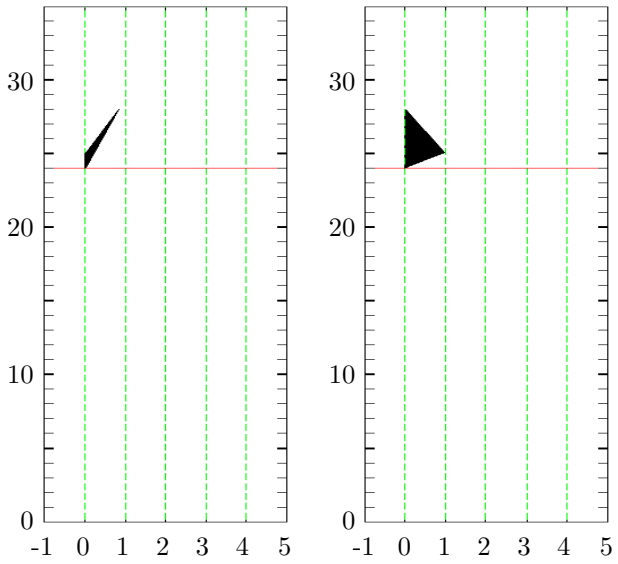

Figura 4.5: nivel 24, problema dos veces reducido.

Paso 1. Hasta el momento tenemos un candidato de la forma "$X = (0, -, -, 3)$". Examinando las nuevas proyecciones restringidas al nivel 24, obtenemos: $Rango_2^{24} = \{0\}$ y $Rango_3^{24} = \{0\}$ (caso (a)).

Paso 2. El candidato que resulta de observar las proyecciones, el punto $X = (0, 0, 0, 3)$, es un candidato factible y al evaluarlo en f coincide con el nivel actual. Hallamos el óptimo del problema, fin del algoritmo.

4.1. Número de Operaciones

Consideremos el siguiente problema de tipo UKP y veamos cuántas operaciones son necesarias para calcular el conjunto de proyecciones.

$$\begin{aligned} \text{maximizar} \quad & f = p_1x_1 + \cdots + p_nx_n \\ \text{sujeta a} \quad & w_1x_1 + \cdots + w_nx_n \leq c \\ & x_i \geq 0,\ x_i \in \mathbb{Z},\ i = 1, \ldots, n \end{aligned}$$

Para simplificar la notación, supondremos que las clases están numeradas cumpliendo:

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}.$$

Empecemos viendo cuántas operaciones son necesarias para calcular las n proyecciones inferiores:

$$\left.\begin{array}{rl} P^{inf}_{X_j}(x_j) := \text{mín valor} & p_1x_1 + \cdots + p_jx_j + \cdots + p_nx_n \\ \text{sujeta a} & w_1x_1 + \cdots + w_jx_j + \cdots + w_nx_n \leq c \\ & x_i \geq 0,\ x_i \in \mathbb{R},\ i \neq j \end{array}\right\} \Rightarrow \quad P^{inf}_{X_j}(x_j) = p_jx_j, \text{ con } x_j \in \Big[0, \frac{c}{w_j}\Big].$$

Es decir: sólo debemos realizar n operaciones (calcular los cocientes "$\frac{c}{w_j}$") para determinar las n proyecciones inferiores.

Para conocer las proyecciones superiores, separemos el problema en dos casos. Por un lado, calculemos $P^{sup}_{X_1}(x_1)$ (la proyección de la clase más valiosa), y, por otro, $P^{sup}_{X_j}(x_j)$ para $j \geq 2$.

$$\left.\begin{array}{rl} P^{sup}_{X_1}(x_1) := \text{máx valor} & p_1x_1 + \cdots + p_nx_n \\ \text{sujeta a} & w_1x_1 + \cdots + w_nx_n \leq c \\ & x_i \geq 0,\ x_i \in \mathbb{R},\ i \geq 2 \end{array}\right\} \Rightarrow \quad P^{sup}_{X_1}(x_1) = \Big(p_1 - p_2\frac{w_1}{w_2}\Big)x_1 + \frac{c}{w_2}p_2, \text{ con } x_1 \in \Big[0, \frac{c}{w_1}\Big].$$

$$\left.\begin{array}{rl} P^{sup}_{X_j}(x_j) := \text{máx valor} & p_1x_1 + \cdots + p_nx_n \\ \text{sujeta a} & w_1x_1 + \cdots + w_nx_n \leq c \\ & x_i \geq 0,\ x_i \in \mathbb{R},\ i \neq j \end{array}\right\} \Rightarrow \quad P^{sup}_{X_j}(x_j) = \Big(p_j - p_1\frac{w_j}{w_1}\Big)x_j + \frac{c}{w_1}p_1, \text{ con } x_j \in \Big[0, \frac{c}{w_j}\Big].$$

Con lo cual: primero debemos realizar $O(n)$ operaciones para identificar las dos clases más valiosas. Y luego, calcular los coeficientes "$p_1 - p_2\frac{w_1}{w_2}$" y "$\frac{c}{w_2}p_2$" para conocer la pendiente y la ordenada de la recta $P^{sup}_{X_1}(x_1)$, y, por otro lado, "$p_j - p_1\frac{w_j}{w_1}$" y "$\frac{c}{w_1}\ p_1$" para conocer la pendiente y la ordenada de las rectas $P^{sup}_{X_j}(x_j)$ para $j \geq 2$.

Conclusión: son necesarias $O(n)$ operaciones para averiguar el primer grupo de proyecciones del problema.

4.2. Experimentos Computacionales

Para medir la eficiencia de nuestro método, hemos utilizado –para ser consistentes con la mayoría de los trabajos presentados en la literatura– una serie de instancias *test* generadas aleatoriamente según los procedimientos sugeridos por Martello & Toth [28] y David Pinsinger [32]. Cada una de las instancias propuestas está pensada para representar ciertos modelos de la realidad y para revelar falencias de los distintos métodos.

4.2.1. Instancias *Test*

Instancias *Uncorrelated:* p_i se genera aleatoriamente en el intervalo [1,R] y w_i en el intervalo [10,R]. Este tipo de instancias sirve para modelar situaciones en las que se puede suponer que el valor y el peso de los objetos no guardan relación, puede haber objetos con gran valor y poco peso, y viceversa. En general son instancias fáciles de resolver para todos los métodos.

Instancias *Weakly Correlated:* los pesos w_i se generan aleatoriamente en el intervalo [10,R] y los valores p_i en $[w_i - 100, w_i + 100]$. En este caso, el valor y el peso sí guardan cierta relación, típicamente el valor sólo difiere del peso en un porcentaje pequeño. Instancias como éstas modelan muy bien situaciones de gestión o administración donde se dispone de cierto capital y varias opciones de inversión. El valor de retorno de una inversión se supone proporcional al capital invertido más o menos alguna variación.

Instancias *Strongly Correlated:* los w_i se generan aleatoriamente en el intervalo [10,R] y los $p_i = w_i + 100$. Estas instancias representan situaciones de la vida real donde el retorno es proporcional a la inversión más algún cargo extra por cada proyecto. En general las instancias de este tipo son difíciles de resolver.

Instancias *Sub-set Sum:* los w_i se generan aleatoriamente en el intervalo [10,R] y los $p_i = w_i$. Estas instancias reflejan situaciones donde el valor y el peso de cada objeto son iguales. Por lo tanto, el problema es equivalente a llenar la mochila con la mayor cantidad de peso. En general, son instancias difíciles de resolver para todos los métodos, porque todas las cotas superiores resultan en el valor trivial c (capacidad de la mochila).

4.2.2. Resultados y Conclusiones

Comparamos el rendimiento de nuestro método, PSA, contra el *solver* comercial CPLEX [20] de IBM (versión 10.1.0). En una primera etapa, llevamos a cabo la comparación sin modificar los parámetros por *default* de CPLEX; es decir, con todas las herramientas activas: pre-procesamiento, cortes, heurísticas, etc.. Trabajamos sobre instancias de tipo *Uncorrelated*, *Weakly Correlated*, *Strongly Correlated* y *Sub-set Sum* con 5000, 20000, 50000 y 70000 variables. Luego, en una segunda etapa, repetimos las pruebas realizadas sobre las dos instancias más grandes –50000 y 70000 variables– desactivando las opciones de *pre-solve*: "set pre pre n" y "set pre red 0". Esto lo hicimos para determinar el grado de incidencia que tiene el sistema de pre-procesamiento sobre el resultado final del *solver*. Todas las mediciones fueron realizadas en una computadora *SUN UltraSparc III workstation with a CPU running at 1GHz and 2GB of RAM memory*. El límite de tiempo para cada corrida fue fijado en 200 segundos. A continuación, se muestran los resultados computacionales de las pruebas mencionadas junto con las conclusiones del experimento. Notaremos con la letra "n" a la cantidad de variables de cada una de las instancias.

Cómo leer las tablas: cada casillero de la tabla contiene 5 valores, por ejemplo

CPLEX	PSA	niveles
(10) 0.256 seg.	(1) 0.02 seg.	1

En las columnas 1 y 2, se indica, entre paréntesis, el número de instancias resueltas en forma óptima por parte de cada uno de los métodos (sobre un total de 10 problemas *test*). Y, a la derecha de ese valor, se muestra el tiempo promedio consumido por cada uno de los métodos sobre las instancias que pudieron ser resueltas. En la columna 3, se exhibe la cantidad de niveles (promedio) recorridos por PSA hasta alcanzar la solución óptima.

Tabla 2. Instancias *Uncorrelated*.

	w_i = random(10,100) p_i = random(1,100) $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,1000) p_i = random(1,1000) $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,10000) p_i = random(1,10000) $c = 0.5 \sum_{i=1}^{n} w_i$		
n	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**
5000	**(10) 0.256 s.**	(1) 0.02 s.	1	(10) 0.083 s.	**(10) 0.021 s.**	1	**(10) 0.096 s.**	(10) 0.326 s.	109.1
20000	**(10) 2.393 s.**	(1) 0.05 s.	1	(10) 0.455 s.	**(10) 0.051 s.**	1	(10) 0.361 s.	**(10) 0.045 s.**	1
50000	(10) 8.442 s.	**(10) 0.124 s.**	1	(10) 1.934 s.	**(10) 0.102 s.**	1	(10) 1.012 s.	**(10) 0.094 s.**	1
70000	(10) 12.138 s.	**(10) 0.172 s.**	1	(10) 3.409 s.	**(10) 0.145 s.**	1	(10) 1.489 s.	**(10) 0.120 s.**	1
50000	(10) 10.062s.	**(10) 0.124 s.**	1	(10) 76.8650s.	**(10) 0.102 s.**	1	(10) 85.1430s.	**(10) 0.094 s.**	1
70000	(10) 14.161s.	**(10) 0.172 s.**	1	(10) 145.966s.	**(10) 0.145 s.**	1	(10) 166.742s.	**(10) 0.120 s.**	1

Tabla 3. Instancias *Strongly Correlated*.

	w_i = random(10,100) $p_i = w_i + 100$ $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,1000) $p_i = w_i + 100$ $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,10000) $p_i = w_i + 100$ $c = 0.5 \sum_{i=1}^{n} w_i$		
n	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**
5000	**(10) 0.032 s.**	(1) 0.03 s.	1	(10) 0.038 s.	**(10) 0.02 s.**	1	**(10) 0.073 s.**	(9) 0.028 s.	1
20000	**(10) 0.128 s.**	(1) 0.05 s.	1	(10) 0.128 s.	**(10) 0.051 s.**	1	(10) 0.224 s.	**(10) 0.059 s.**	1
50000	(10) 0.320 s.	**(10) 0.137 s.**	1	(10) 0.333 s.	**(10) 0.126 s.**	1	(10) 0.447 s.	**(10) 0.113 s.**	1
70000	(10) 0.464 s.	**(10) 0.189 s.**	1	(10) 0.476 s.	**(10) 0.174 s.**	1	(10) 0.642 s.	**(10) 0.172 s.**	1
50000	(10) 10.056 s.	**(10) 0.137 s.**	1	(10) 49.251 s.	**(10) 0.126 s.**	1	(10) 76.362 s.	**(10) 0.113 s.**	1
70000	(10) 14.173 s.	**(10) 0.189 s.**	1	(10) 84.433 s.	**(10) 0.174 s.**	1	(10) 145.014 s.	**(10) 0.172 s.**	1

Tabla 4. Instancias *Weakly Correlated*.

	w_i = random(10,100) $p_i = w_i$ + random(-100,100) si $p_i \leq 10$ ponemos: $p_i = 10$ $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,1000) $p_i = w_i$ + random(-100,100) si $p_i \leq 10$ ponemos: $p_i = 10$ $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,10000) $p_i = w_i$ + random(-100,100) si $p_i \leq 10$ ponemos: $p_i = 10$ $c = 0.5 \sum_{i=1}^{n} w_i$		
n	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**
5000	**(10) 0.226 s.**	(1) 0.020 s.	1	(10) 0.126 s.	**(10) 0.026 s.**	1	(10) 0.093 s.	**(10) 0.054 s.**	6.4
20000	**(10) 2.122 s.**	(1) 0.08 s.	1	(10) 1.069 s.	**(10) 0.053 s.**	1	(10) 0.388 s.	**(10) 0.055 s.**	1
50000	(10) 9.009 s.	**(10) 0.122 s.**	1	(10) 5.761 s.	**(10) 0.118 s.**	1	(10) 1.671 s.	**(10) 0.091 s.**	1
70000	(10) 16.471 s.	**(10) 0.164 s.**	1	(10) 10.45 s.	**(10) 0.164 s.**	1	(10) 2.655 s.	**(10) 0.137 s.**	1
50000	(10) 13.241 s.	**(10) 0.122 s.**	1	(10) 58.859 s.	**(10) 0.118 s.**	1	(10) 79.589 s.	**(10) 0.091 s.**	1
70000	(10) 19.277 s.	**(10) 0.164 s.**	1	(10) 105.659 s.	**(10) 0.164 s.**	1	(10) 153.278 s.	**(10) 0.137 s.**	1

Tabla 5. Instancias *Sub-set Sum*.

	w_i = random(10,100) $p_i = w_i$ $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,1000) $p_i = w_i$ $c = 0.5 \sum_{i=1}^{n} w_i$			w_i = random(10,10000) $p_i = w_i$ $c = 0.5 \sum_{i=1}^{n} w_i$		
n	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**	**CPLEX**	**PSA**	**niveles**
5000	**(10) 0.078 s.**	(10) 0.097 s.	1	(10) 0.072 s.	**(10) 0.071 s.**	1	**(10) 0.061 s.**	(10) 0.081 s.	1
20000	(10) 0.476 s.	**(10) 0.220 s.**	1	**(10) 0.298 s.**	(10) 0.303 s.	1	(10) 0.280 s.	**(10) 0.265 s.**	1
50000	**(10) 0.819 s.**	(10) 0.850 s.	1	(10) 0.829 s.	**(10) 0.704 s.**	1	(10) 0.822 s.	**(10) 0.609 s.**	1
70000	**(10) 1.187 s.**	(10) 1.288 s.	1	**(10) 1.223 s.**	(10) 2.109 s.	1	**(10) 1.211 s.**	(10) 1.239 s.	1
50000	(9) 11.418 s.	**(10) 0.850 s.**	1	(10) 1.292 s.	**(10) 0.704 s.**	1	(10) 2.388 s.	**(10) 0.609 s.**	1
70000	(10) 82.89 s.	**(10) 1.288 s.**	1	(10) 12.72 s.	**(10) 2.109 s.**	1	(10) 6.635 s.	**(10) 1.239 s.**	1

A partir de las tablas presentadas en este capítulo, observamos que PSA es más veloz que CPLEX sobre la mayor parte de las instancias testeadas. Si comparamos los resultados obtenidos en la primera etapa de nuestro experimento (utilizando CPLEX con *pre-solve*), advertimos que PSA redujo los tiempos de resolución en el 70 % de los casos estudiados. Más marcada aún es la diferencia obtenida al comparar los valores de la segunda etapa. En ese caso, el porcentaje de las instancias sobre las que PSA logró mejorar el rendimiento de CPLEX alcanza el 100 % de los casos examinados. En base a los resultados conseguidos en ambas etapas, concluimos que el desempeño de CPLEX merma significativamente cuando se desactivan las opciones de *pre-solve*. En esos casos, los tiempos de resolución pueden pasar de "algunos segundos" –incluso décimas de segundo– a "minutos". Estos valores demuestran que la fase de pre-procesamiento desempeña un rol clave en la performance final del *solver* cuando se trabaja sobre problemas de tipo UKP.

También vale la pena comentar los requerimientos de memoria de cada uno de los métodos. En instancias con 70000 variables, por ejemplo, el consumo de memoria de PSA oscila entre 800MB y 1.8GB; mientras que CPLEX utiliza tan sólo 50MB. El alto consumo de memoria por parte de PSA es producto de almacenar, a lo largo de todo el proceso, la información correspondiente a 70000 proyecciones inferiores y 70000 proyecciones superiores.

Sobre instancias de tamaño *pequeño* y *mediano* (estos datos no figuran en las tablas), PSA resultó muy ineficiente. En dichos casos, el óptimo se ubica profundo –por lo tanto, el algoritmo debe recorrer una gran cantidad de niveles hasta alcanzarlo– y el rango de valores de cada una de las variables suele ser grande. Para mejorar el rendimiento del método en situaciones de este tipo, proponemos, para un futuro trabajo, llevar a cabo las siguientes operaciones:

- Implementar cotas superiores (por ejemplo, la cota U_4 que presentamos en el capítulo 2) para evitar el análisis de los primeros niveles del problema.
- Agrupar todos los niveles que tengan el mismo rango de valores para todas las variables y analizarlos en conjunto. Así podríamos aprovechar las operaciones realizadas en un nivel para todos aquellos niveles que tengan las mismas características.
- Implementar sistemas de pre-procesamiento para reducir el tamaño del problema original.

El balance de este experimento es bueno. Logramos mejorar los tiempos de un *solver* comercial con más de 20 años de desarrollo, empleando un procedimiento que no tiene implementado ningún sistema de *pre-solve*. Vale recordar que, en la mayoría de los algoritmos diseñados específicamente para resolver problemas de tipo UKP, la esencia del algoritmo es, justamente, el sistema de pre-procesamiento (ver capítulo 2).

Capítulo 5

Mochila Multidimensional

En este capítulo presentaremos detalles y resultados de la aplicación del algoritmo PSA sobre problemas de tipo *Mochila Multidimensional* (MKP). Comenzaremos con un ejemplo sencillo para mostrar cómo se calculan las proyecciones en este caso y, posteriormente, expondremos los resultados computacionales de la comparación entre CPLEX y PSA sobre instancias de este tipo.

$$\begin{aligned} \text{maximizar} \quad & f = 36x_1 + 25x_2 + 6x_3 \\ \text{sujeta a} \quad & 43x_1 + 28x_2 + 17x_3 \leq 44 \\ & 38x_1 + 51x_2 + 43x_3 \leq 66 \\ & x_i \in \{0, 1\},\ i = 1, 2, 3 \end{aligned}$$

Inicio del algoritmo, ***Paso 0.***

En problemas de clase MKP o, más generalmente, en PPLE que contengan al menos dos restricciones, difícilmente podamos deducir fórmulas para $P_{X_i}^{sup}(x_i)$ y $P_{X_i}^{inf}(x_i)$ para valores genéricos de x_i (como sí lo hicimos para el UKP). Por lo tanto, en situaciones de este tipo, nos limitaremos a calcular los valores estrictamente necesarios para poder utilizar las proyecciones. Estos valores son: $P_{X_i}^{sup}(e_i)$ y $P_{X_i}^{inf}(e_i)$ para todos los valores enteros e_i de x_i. En el caso particular del MKP, el problema de calcular las proyecciones se reduce, por ende, a encontrar $P_{X_i}^{sup}(1)$, $P_{X_i}^{sup}(0)$, $P_{X_i}^{inf}(1)$ y $P_{X_i}^{inf}(0)$ para cada una de las variables x_i. Una vez conseguidas estas pseudo-proyecciones, continuamos con el *Paso 1* del algoritmo.

Veamos cómo se calcula $P_{X_1}^{sup}(1)$.

Por definición, $P_{X_1}^{sup}(1) :=$ máx valor $\quad f = 36 \,.\, 1 + 25x_2 + 6x_3$

$$\begin{aligned} \text{sujeta a} \quad & 43 \,.\, 1 + 28x_2 + 17x_3 \leq 44 \\ & 38 \,.\, 1 + 51x_2 + 43x_3 \leq 66 \\ & 0 \leq x_i \leq 1,\ i = 2, 3 \quad \longleftarrow \text{ Relajamos esta condición.} \end{aligned}$$

Resolviendo este PPL a través del método *Simplex*, obtenemos: $P_{X_1}^{sup}(1) = 36.8929$.

Repitiendo el procedimiento anterior para cada uno de los valores enteros y para cada una de las variables restantes, resulta:

$P^{sup}_{X_1}(1) = 36.8929$ $P^{sup}_{X_2}(1) = 38.3953$ $P^{sup}_{X_3}(1) = 27.7895$

$P^{sup}_{X_1}(0) = 27.0930$ $P^{sup}_{X_2}(0) = 36.3529$ $P^{sup}_{X_3}(0) = 38.3953$

$P^{inf}_{X_1}(1) = 36$ $P^{inf}_{X_2}(1) = 25$ $P^{inf}_{X_3}(1) = 6$

$P^{inf}_{X_1}(0) = 0$ $P^{inf}_{X_2}(0) = 0$ $P^{inf}_{X_3}(0) = 0$

A continuación, graficamos el conjunto de proyecciones y analizamos los tres primeros niveles enteros *candidatos* a ser el nivel óptimo de la función.

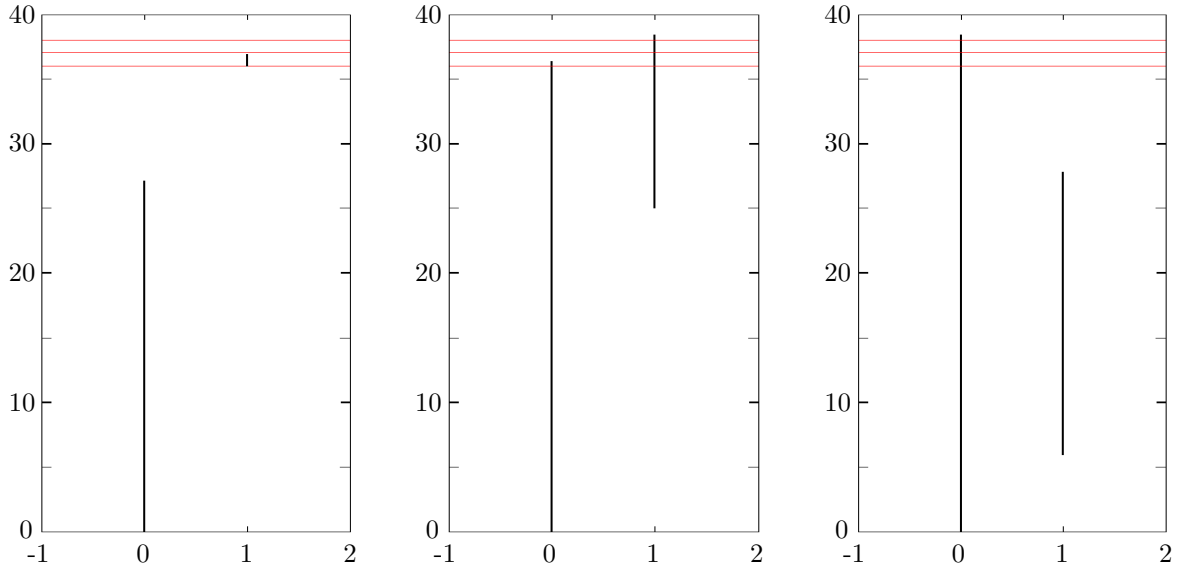

Figura 5.1: niveles 38, 37 y 36.

Niveles 38 y 37.

Paso 1. Examinando las proyecciones restringidas a cualquiera de estos dos niveles, obtenemos: $Rango^N_1 = \emptyset$, $Rango^N_2 = \{1\}$ y $Rango^N_3 = \{0\}$ para $N = 38$ y 37 (caso (b)). Luego, no puede existir ninguna solución binaria tal que al ser evaluada en f alcance los niveles 38 o 37. Como $Pendientes = \emptyset$, analizamos el próximo nivel.

Nivel 36.

Paso 1. $Rango^{36}_1 = \{1\}$, $Rango^{36}_2 = \{0, 1\}$ y $Rango^{36}_3 = \{0\}$ (caso (c)).

Paso 3. Actualizamos nuestro candidato a solución: $X = (1, -, 0)$. Y reemplazamos los valores de x_1 y x_3 en el problema original; se deriva el siguiente problema reducido:

$$\begin{aligned} \text{maximizar} \quad & f = 25x_2 + 36 \\ \text{sujeta a} \quad & 28x_2 \leq 1 \\ & 51x_2 \leq 28 \\ & x_2 \in \{0, 1\} \end{aligned}$$

Como ahora el problema contiene una sola variable, la proyección coincide con el gráfico de la función: $\mathbf{P_{X_2}}$ es un segmento de recta de pendiente 25, con ordenada al origen 36, y está definida en el intervalo $\left[0, \frac{1}{28}\right]$.

Paso 1. $Rango_2^{36} = \{0\}$ (caso(a)).

Paso 2. El candidato "$X = (1, 0, 0)$" es factible y al ser evaluado en f coincide con el nivel actual (satisface las condiciones de optimalidad). Fin del algoritmo.

5.1. Detalles de Implementación

Hagamos algunas precisiones acerca de la implementación del método PSA en lo que se refiere al cálculo de proyecciones para problemas de tipo MKP.

En primer lugar, no es necesario realizar ninguna operación para calcular las proyecciones inferiores. Dado que todos los coeficientes del problema son positivos, se sigue que: $P_{X_i}^{inf}(x_i) = p_i x_i$ $\forall\, x_i \in [0, 1]$. Con lo cual, calcular las proyecciones de un MKP se reduce a hallar $P_{X_i}^{sup}(0)$ y $P_{X_i}^{sup}(1)$ para cada variable x_i.

En cuanto a las proyecciones superiores, en el ejemplo anterior utilizamos el método *Simplex* para hallar la solución exacta de cada uno de los PPL asociados a la definición de $P_{X_i}^{sup}(e_i),\ e_i \in \{0, 1\}$. En la práctica, resolver 2 PPL por cada variable incluida en el modelo resulta prohibitivo; con lo cual, se torna necesario simplificar el número de operaciones. Con este objetivo en mente, introdujimos dos modificaciones en la implementación del método para reducir el número de operaciones cada vez que calculamos las proyecciones superiores:

1. El algoritmo comienza resolviendo la relajación lineal asociada al problema entero. Supongamos que el óptimo de la relajación se alcanza en el punto "$\bar{X}$" con valor máximo "f_{max}". Si la coordenada j-ésima del vector "$\bar{X}$" es un valor entero "e_j" (0 o 1), entonces, deducimos que $P_{X_j}^{sup}(e_j) = f_{max}$. Es decir, suponiendo que el problema que estemos intentando resolver posea n variables, mediante esta primera operación podemos calcular –en el mejor de los casos– hasta $n - 1$ de las $2n$ proyecciones superiores. Y esto se logra resolviendo un sólo PPL.

2. Para averiguar los restantes valores, utilizamos la tabla *Simplex-final* conseguida en el paso anterior para, agregando una restricción adicional a la formulación del problema ($x_i \leq 0$ o $x_i \geq 1$), calcular cotas superiores para $P_{X_i}^{sup}(1)$ y $P_{X_i}^{sup}(0)$. Esta operación nos permite completar rápidamente el conjunto de proyecciones y, de esa manera, asignar un único valor entero a un gran porcentaje de variables en muy poco tiempo. En contraposición, este procedimiento puede incrementar el rango de valores de algunas de las variables. Resta, para un futuro trabajo, estudiar en qué momento es conveniente reemplazar los valores aproximados, obtenidos en esta etapa, por valores exactos.

Por lo tanto, las proyecciones superiores utilizadas en nuestra implementación son una combinación entre valores exactos y aproximados. Lo cual no implica que las soluciones obtenidas a través de este procedimiento no sean exactas.

5.2. Experimentos Computacionales

En este caso, hemos trabajado en base a los artículos de Arnaud Frèville [36, 10] para fabricar las instancias *test* utilizadas para medir la perfomance de nuestro algoritmo. Las instancias de tipo MKP son una generalización de las de tipo UKP adaptadas a problemas con numerosas restricciones.

5.2.1. Instancias *Test*

Instancias *Uncorrelated:* los w_{ij} se generan aleatoriamente en el intervalo [1,1000] y los coeficientes c_j y p_i de la siguiente manera:

$$c_j := 0.5 \sum_{i=1}^{n} w_{ij} \quad \text{y} \quad p_i := \text{random}[1, 1000].$$

Donde n indica la cantidad de variables del problema y m el número de restricciones.

Instancias *Weakly Correlated:* los w_{ij} se generan aleatoriamente en el intervalo [1,1000] y los coeficientes c_j y p_i de la siguiente manera:

$$c_j := 0.5 \sum_{i=1}^{n} w_{ij} \quad \text{y} \quad p_i := \frac{\sum_{j=1}^{m} w_{ij}}{m} + \text{random}[-100, 100].$$

5.2.2. Resultados y Conclusiones

Nuevamente, comparamos el rendimiento de PSA contra el *solver* comercial CPLEX [20] de IBM (versión 10.1.0). En todos los casos, corrimos CPLEX sin modificar los parámetros por *default*; es decir, con todas las herramientas activas: pre-procesamiento, cortes, heurísticas, etc.. Trabajamos sobre instancias de tipo *Uncorrelated* y *Weakly Correlated* con 3000, 5000, 10000 y 20000 variables y 1, 2 y 3 restricciones. Todas las mediciones fueron realizadas en una computadora *SUN UltraSparc III workstation with a CPU running at 1GHz and 2GB of RAM memory*. En este caso, no impusimos ningún límite de tiempo para las corridas, con lo cual, las instancias que no pudieron ser resueltas corresponden a problemas donde se superaron los 2GB de memoria. A continuación, exponemos los resultados obtenidos a través de estas pruebas junto con las conclusiones del experimento. Notaremos con las letras "n" y "m" al número de variables y restricciones, respectivamente, que contienen cada una de las instancias.

Cómo leer las tablas: cada casillero de la tabla contiene 7 valores, por ejemplo

CPLEX	PSA	niveles	1ª iteración
(10) 0.924 seg.	**(10) 0.153 seg.**	4.6	2973.8 (99.12 %)

En las columnas 1 y 2 se indica, entre paréntesis, el número de instancias resueltas en forma óptima por parte de cada uno de los algoritmos (sobre un total de 10 problemas *test*). Y, a la derecha de ese valor, se muestra el tiempo promedio consumido por cada uno de los métodos sobre las instancias que pudieron ser resueltas. En la columna 3, se exhibe la cantidad de niveles (promedio) recorridos por PSA hasta alcanzar la solución óptima. Por último, en la columna 4, se indica el número de variables (promedio) a las que PSA le asigna un único valor entero en la primera iteración del nivel óptimo. A la derecha de este último valor, figura el porcentaje que representa sobre el total de variables.

Tabla 6. Resultados computacionales sobre instancias *Uncorrelated*.

n	m	CPLEX	PSA	niveles	1ª iteración
3000	1	(10) 0.924 seg.	**(10) 0.153 seg.**	4.6	2973.8 (99.12 %)
3000	2	(10) 44.733 seg.	**(10) 15.883 seg.**	10.3	2936.7 (97.89 %)
3000	3	(10) 25.84 min.	**(10) 10.652 min.**	18.9	2888.1 (96.27 %)
3000	4	(9) 6.548 hs.	**(10) 6.63 hs.**[3]	25.1	2842.6 (94.75 %)
5000	1	(10) 1.733 seg.	**(10) 0.168 seg.**	2.8	4977.6 (99.55 %)
5000	2	(10) 2.612 min.	**(10) 26.802 seg.**	7.6	4925.2 (98.50 %)
5000	3	(10) 1.009 hs.	**(10) 12.38 min.**	11.3	4882.6 (97.65 %)
10000	1	(10) 2.831 seg.	**(10) 0.254 seg.**	1.8	9974.4 (99.74 %)
10000	2	(10) 5.398 min.	**(10) 47.414 seg.**	4.3	9919.3 (99.19 %)
10000	3	(8) 3.42 hs.	**(10) 53.73 min.**	6.9	9874.4 (98.74 %)
20000	1	(10) 7.373 seg.	**(10) 0.419 seg.**	1.5	19961.2 (99.80 %)
20000	2	(10) 24.953 min.	**(10) 1.119 min.**	2.8	19908.2 (99.54 %)
20000	3	(1) 17.148 hs.[4]	**(10) 4.387 hs.**	4.6	19830.0 (99.15 %)

Tabla 7. Resultados computacionales sobre instancias *Weakly Correlated*.

n	m	CPLEX	PSA	niveles	1ª iteración
3000	1	(10) 0.873 seg.	**(10) 0.072 seg.**	1.4	2975.0 (99.16 %)
3000	2	(10) 2.396 min.	**(10) 15.087 seg.**	3.0	2904.3 (96.81 %)
3000	3	(5) 3.27 hs.	**(10) 1.096 hs.**	4.6	2852.0 (95.06 %)
5000	1	(10) 1.486 seg.	**(10) 0.079 seg.**	1.3	4953.4 (99.06 %)
5000	2	(10) 3.731 min.	**(10) 26.631 seg.**	2.1	4900.3 (98.00 %)
5000	3	(1) 11.27 hs.[5]	**(10) 1.945 hs.**	3.7	4812.4 (96.24 %)
10000	1	(10) 3.274 seg.	**(10) 0.159 seg.**	1.0	9941.7 (99.41 %)
10000	2	(10) 8.342 min.	**(10) 16.414 seg.**	2.0	9845.8 (98.45 %)
10000	3	(0) ———	**(10) 4.73 hs.**	2.0	9814.4 (98.14 %)
20000	1	(10) 10.511 seg.	**(10) 0.308 seg.**	1.0	19868.9 (99.34 %)
20000	2	(10) 14.235 min.	**(10) 20.749 seg.**	1.2	19839.0 (99.19 %)
20000	3	(0) ———	**(10) 10.39 hs.**	1.6	19720.6 (98.60 %)

[3]Sobre las 9 instancias resueltas por CPLEX, PSA consumió, en promedio, 4.518 horas.

[4]El tiempo consumido por PSA sobre esta misma instancia fue de 7.29 horas.

[5]El tiempo consumido por PSA sobre esta misma instancia fue de 1.87 horas.

En base a los resultados computacionales presentados en este capítulo, concluimos:

- PSA es más veloz que CPLEX en *absolutamente todas* las instancias testeadas.
- Sobre todas las instancias testeadas con dos o más restricciones, el consumo de memoria de PSA resultó inferior al de CPLEX. Por ejemplo, si consideramos las pruebas realizadas sobre instancias *Weakly Correlated* con 10000 variables y 3 restricciones, PSA consumió un promedio de 160.46MB de memoria (consumo máximo), mientras que CPLEX agotó, en todos los casos, los 2GB disponibles. Es decir, PSA utilizó *menos* del 8.023 % de la memoria requerida por CPLEX.
- Es interesante destacar el número de variables a las que PSA le asigna un único valor entero en la primera iteración del nivel óptimo. En todos los casos, este porcentaje asciende a más del 94 %. Con lo cual, luego de la primera iteración, PSA reduce el problema original a un nuevo problema que contiene menos del 6 % de las variables. De aquí los buenos resultados obtenidos tanto en tiempos de resolución como en consumo de memoria.

En este caso, el balance del experimento es muy bueno. Logramos mejorar los tiempos y los requerimientos de memoria de CPLEX sobre instancias de tamaño *grande* no publicadas en la literatura del área. Resta evaluar, en un futuro trabajo, el comportamiento del algoritmo sobre instancias de mayor tamaño (experimento que no podemos hacer con la computadora actual debido al alto consumo de memoria –sobre todo por parte de CPLEX– y a los tiempos de resolución requeridos). Por último, también sería interesante medir la performance del algoritmo aplicando, previamente, algún sistema de pre-procesamiento.

Capítulo 6

Conclusiones

Al comenzar esta tesis, hicimos un repaso (en el capítulo 1) de los principales métodos empleados en la resolución de PPLE en forma exacta: el algoritmo de *Planos de Corte* y los métodos *Branch-and-Bound* y *Branch-and-Cut*. En todos los casos, vimos que la estrategia para buscar el óptimo consistía en modificar el dominio del problema –habiendo considerado previamente su relajación lineal– mediante el agregado de nuevas desigualdades lineales. En el caso del algoritmo de *Planos de Corte*, las nuevas desigualdades eran utilizadas para separar soluciones fraccionarias de la relajación lineal y conservar el conjunto de soluciones factibles enteras del problema original. En el caso de los métodos *Branch-and-Bound* y *Branch-and-Cut*, las desigualdades eran usadas para particionar el dominio del problema y eliminar soluciones fraccionarias de la relajación.

Si bien la estrategia que acabamos de comentar es la más difundida entre los algoritmos exactos; es una forma de trabajar que presenta una contra importante: añadir restricciones adicionales a la formulación del problema, implica un aumento del número de operaciones realizadas cada vez que se debe resolver una nueva relajación lineal. El método que presentamos en el capítulo 3 evita, justamente, esta dificultad. Al tratarse de un algoritmo que no actúa sobre el dominio del problema, "PSA" no requiere del agregado de nuevas restricciones lineales para su funcionamientos. Además, podemos destacar otras dos características que también lo distinguen de los algoritmos mencionados anteriormente: 1º) genera los candidatos a solución en términos del valor de la función; y 2º) reduce sistemáticamente el número de variables del problema.

A partir de los experimentos computacionales presentados en los capítulos 4 y 5 (producto de comparar el rendimiento de nuestro método contra el *solver* comercial CPLEX de IBM sobre instancias de tipo UKP y MKP), concluimos:

- Sobre instancias con más de 3000 variables y una única restricción, PSA es más veloz que CPLEX (en la mayoría de los casos) pero el consumo de memoria es mucho más elevado.
- En instancias con *al menos* dos restricciones y 3000 variables, PSA es más veloz que CPLEX en *absolutamente todas las instancias testeadas*. Y, además, reduce considerablemente el consumo de memoria.

Con respecto a las instancias de tamaño *pequeño* (estos datos no figuran en las tablas), no obtuvimos buenos resultados al aplicar el método PSA. En general, a medida que disminuye el número de variables del problema y/o aumenta la cantidad de restricciones, la distancia entre el nivel óptimo del problema entero y el de la relajación lineal es cada vez mayor. Esto se traduce en un aumento del número de niveles analizados por parte de nuestro algoritmo y, como consecuencia, de la cantidad de operaciones realizadas. Para mejorar la performace del método para esta clase de problemas, proponemos, para un futuro trabajo, implementar las siguientes herramientas:

- Implementar cotas superiores –siempre que la situación lo permita– para evitar el análisis de los primeros niveles del problema.
- Agrupar todos los niveles que tengan el mismo rango de valores para todas las variables y analizarlos en conjunto. De esa manera, podríamos aprovechar las operaciones realizadas en un nivel para todos aquellos niveles que tengan las mismas características (es el caso de los niveles 31, 30 y 29, por un lado, y de los niveles 27, 26 y 25, por otro, en la instancia resuelta de tipo UKP en el capítulo 3).
- Implementar sistemas de pre-procesamiento para reducir el tamaño del problema original.

Con todo lo expresado, pensamos que este trabajo aporta un nuevo enfoque en la resolución de problemas de programación lineal entera. Resta, para un futuro trabajo, evaluar el comportamiento del algoritmo sobre instancias de mayor tamaño (principalmente, con mayor número de restricciones); como así también, sobre otra clase de problemas. Por último, también sería interesante medir la performance del algoritmo aplicando, previamente, algún sistema de pre-procesamiento.

Bibliografía

[1] D. L. Applegate, R. E. Bixby, V. Chvatal and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princenton University Press, 2006.

[2] Balas, E., *An Additive Alogrithm for Solving Linear Programs with Zero-One Vari- ables*, Operations Research, 13, 4, 517-549 (1965).

[3] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh and P. Vance. *Branch-and-Price: Column Generation for Solving Integer Programs*. Operations Research 46, 316-329, 1998.

[4] H. Crowder, E. Johnson, and M. Padberg.*Solving Large-Scale Zero-One Linear Programming Problems*. Operations Research 31, 803, 1983.

[5] N. S. Cherbaka. *Solving Single and Multiple Plant Sourcing Problems with a Multidimensional Knapsack Model*. Ph.D thesis, Faculty of Virginia Polytechnic Institute, 2004.

[6] G.B. Dantzig. *Linear Programming and Extensions*. Elsevier, 2004.

[7] G.B. Dantzig and M. N. Thapa. *Linear Programming, 1: Introduction*. Princeton University Press, 1963.

[8] G. Dantzig, R. Fulkerson and S. Johnson. *Solution of a large-scale traveling salesman problem*. Journal of Operation Research American Mathematical Society vol 2, nro. 4, 1954.

[9] FICO^{TM} Xpress Optimization Suite, http://optimization.fico.com/, 2011.

[10] A. Fréville. *The multidimensional 0–1 knapsack problem: an overview*. Elsevier, 2004.

[11] Gabrel, V., and Minoux, M., *A Scheme for Exact Separation of Extended Cover Inequalities and Application to Multidimensional Knapsack Problems*, Operations Re- search Letters, 30, 252-264 (2002).

[12] M. Garey and D. Johnson *Computers and Intractability: A Guide to the Theory of NP-Completeness*.W. H. Freeman and Company, San Francisco, 1979.

[13] Gavish, B., and Pirkul, H., *Eficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality*, Mathematical Programming, 31, 78-105 (1985).

[14] B.L. Golden, S. Raghawan and E.A. Wasil, *The Vehicle Routing Problem*. Springer, New York, 2008.

[15] R.E. Gomory. *Outline of an algorithm for integer solutions to linear programs*. Bulletin of the American Mathematical Society 64, 275-278, 1958.

[16] Gurobi Optimizer, http://www.gurobi.com/, 2011.

[17] M. Grostschel, M. Junger and G. Reinelt. *A cutting plane algorithm for the Linear Ordering Problem*. Operations Research, 32, 1195-1220, 1984.

[18] M.Grotschel, M. Junger and G. Reinelt.*Facets of the linear ordering polytope*. Mathematical Programming 33, 43–60, 1985.

[19] M. Grotschel and M.W. Padberg. *On the symmetric travelling salesman problem I: inequalities*. Mathematical Programming 16,265-280, 1979.

[20] IBM ILOG CPLEX Optimizer, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, 2011.

[21] M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi and L. Wolsey. *50 Years of Integer Programming 1958-2008*. Springer, 2010.

[22] N. Karmarkar. *A new polynomial-time algorithm for linear programming*. Combinatorica 4, 373-395, 1984.

[23] L.G. Khachiyan. *A polynomial algorithm in linear programming*. Soviet Mathematics Doklady 20, 191-194, 1979.

[24] H. Kellerer, U. Pferschy and D. Pisinger.*Knapsack Problems*. Springer, 2004.

[25] V. Klee and G.J. Minty. *How good is the simplex algorithm?*. Inequalities (O. Shisha, ed.), vol. III, Academic Press, New York, 159-175, 1972.

[26] A.H. Land and A.G. Doig. *An Automatic Method for Solving Discrete Programming Problems*. Econometrica 28, 497-520, 1960.

[27] J. Linderoth and M. Savelsberg. *A computational study of search strategies for mixed integer programming*. INFORMS Journal on Computing, 11(2), 173–187, 1999.

[28] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[29] G. Nemhauser and L. Wolsey. *Integer Programming and Combinatorial Optimization*. Wiley, 1988.

[30] M. W. Padberg. *On the facial structure of set packing polyhedra*. Mathematical Programming, 5: 199-215, 1973.

[31] M. Padberg and G. Rinaldi. *Optimization of a 532-City Symmetric Traveling Salesman Problem by Branch and Cut*. Oper. Res. Letters, 6, 1-7, 1987.

[32] D. Pisinger. *Algorithms for Knapsack Problems.* Ph.D thesis, University of Copenhagen, Denmark, 1995.

[33] Shih, W., *A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem*, Journal of the Operational Research Society, 30, 369-378 (1979).

[34] Soyster, A.L. Lev, B., and Slivka, W., *Zero-One Programming with many variables and few constraints*, European Journal of Operational Research, 2, 195-201 (1977).

[35] F. Vanderbeck and L. Wolsey. *An Exact Algorithm for IP Column Generation*. Operations Research Letters 19, 151–160, 1996.

[36] C. Wilbaut, S. Hanafi, A. Fréville and S. Balev. *Tabu search: global intensification using dynamic programming.* Université de Valenciennes, France, 2004.

[37] L. Wolsey. *Integer Programming.* Wiley, 1998.