Diseño e implantación de algoritmos para el problema multiparamétrico de programación lineal entera mixta 0-1

Fernando Crema

Octubre 2012

Fernando Crema

Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela.
E-mail: fcremarm@hotmail.com

Tutores: José Luis Quintero y Alejandro Crema

Facultad de Ingeniería, Universidad Central de Venezuela.
E-mail: quintero-jl@hotmail.com

Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela.
E-mail: alejandro.crema@ciens.ucv.ve

Contenido

Lista de figuras

Resumen: En este trabajo se presentan algoritmos para encontrar una solución aproximada para el problema multiparamétrico de programación lineal entera mixta 0-1 asociado a la función objetivo. El problema multiparamétrico surge al considerar la incertidumbre asociada a los parámetros que definen la función objetivo. Cada parámetro se supone perteneciente a un intervalo conocido. El problema multiparamétrico consiste en encontrar un conjunto de soluciones factibles, tal que: para cualesquiera valores de los parámetros una de ellas resuelve el problema aproximadamente. Una estrategia general conocida consiste en definir apropiadamente una secuencia de problemas no paramétricos (con función objetivo sin incertidumbre) cuyas soluciones definen al conjunto buscado. Se diseñan e implantan algoritmos alternativos basados en la misma estrategia general.

Palabras Clave: Programación entera, multiparametrización, tiempo real.

1 Introducción

Sean $l, u \in \Re^n$. Sea $\Omega = \{f : f \in \Re^n, l \leq f \leq u\}$. Sea $f \in \Omega$. Sea $P(f)$ un problema en (x, y) definido como sigue:

$$max \;\; c^t x + f^t y \quad s.a. \quad Ax + By \leq b, \;\; x \in X, \;\; y \in \{0,1\}^n$$

con $c \in \Re^p$, $b \in \Re^m$, $A \in \Re^{m \times p}$, $B \in \Re^{m \times n}$ y $X \subseteq \Re^p$. En este trabajo se consideran dos alternativas para el conjunto en donde se mueven las variables tipo x: $X = \{x : x \in \{0,1\}^p\}$ o $X = \{x : x \in \Re^p, \;\; x \geq 0\}$.

En todo el trabajo se usa la siguiente notación standard: si T es un problema de maximización entonces $F(T)$ denota a su conjunto de soluciones factibles y $v(T)$ a su valor óptimo (si $F(T) = \emptyset$ entonces se usa la convención $v(T) = -\infty$). Puesto que $F(P(f^1)) = F(P(f^2)) \quad \forall f^1, f^2 \in \Omega$, se usa $F(P)$ en lugar de $F(P(f))$. Para simplificar la exposición se asume que $F(P)$ no es vacío. Nótese que $P(f)$ puede ser un problema de Programación Lineal Entera Mixta 0-1 (PEM-0-1) o un problema de Programación Lineal Entera 0-1 (PE-0-1).

El análisis multiparamétrico puede ser considerado cuando hay incertidumbre sobre el valor de los datos. En este trabajo se considera la incertidumbre asociada al vector en la función objetivo correspondiente a las variables tipo y de tal manera que cada parámetro pertenece a un intervalo conocido. El problema multiparamétrico que estudiaremos es una familia de problemas en la cual un miembro es $P(f)$ con $f \in \Omega$.

Sea $\epsilon \geq 0$. Decimos que $(x^{(1)}, y^{(1)}), \cdots, (x^{(k)}, y^{(k)})$ es una solución multiparamétrica ϵ-optimal si: $(x^{(i)}, y^{(i)}) \in F(P)$ para todo $i = 1, \cdots, k$ y

$$v(P(f)) \geq max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, k\} \geq v(P(f)) - \epsilon \;\; \forall f \in \Omega$$

El objetivo de este trabajo es presentar e implantar tres algoritmos para encontrar una solución multiparamétrica ϵ-optimal.

Sea $\lambda \in (0,1)$. Nótese que si $v(P(l)) > 0$ y $\epsilon = \lambda v(P(l))$ entonces

$$v(P(f)) \geq max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, k\} \geq v(P(f))(1 - \lambda) \;\; \forall f \in \Omega$$

Así, ϵ y λ pueden ser interpretados como los errores *absoluto* y *relativo* respectivamente. Esa es la interpretación que asumiremos puesto que nuestra experiencia computacional se realizó sobre problemas para los cuales en forma natural $v(P(l)) > 0$.

Supongamos que tenemos suficiente tiempo como para hallar una solución multiparamétrica ϵ-optimal antes de que las decisiones tengan que ser tomadas.

Entonces, si el verdadero vector f se conoce al momento de la toma de decisiones podemos obtener una solución ϵ-optimal para $P(f)$ rápidamente usando $v(P(f)) \approx max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, k\}$, lo cual puede ser muy útil si la optimización debe hacerse en tiempo real. Adicionalmente una solución ϵ-optimal para $P(f)$ puede servir de guía para reoptimizar en la búsqueda de una mejor solución. Incluso si la estructura de $P(f)$ nos permite resolverlo rápidamente, el análisis multiparamétrico puede ser útil para analizar sistemáticamente los efectos de la incertidumbre.

El análisis multiparamétrico, en el sentido antes expuesto, no es por supuesto la única manera de lidiar con la incertidumbre. Sin entrar en las formalidades matemáticas podemos describir otros enfoques conocidos como sigue:

(i) la Programación Robusta, que consiste en encontrar **una** solución que se desempeñe *razonablemente bien* para cualesquiera valores de los parámetros,

(ii) la Programación Estocástica, que asume conocidas distribuciones probabilísticas para los valores de los parámetros y que consiste entonces en encontrar **una** solución que maximice el valor esperado de la función objetivo y

(iii) el Análisis de Sensibilidad, que consiste en encontrar una *región de estabilidad* para la solución óptima asociada a un punto de referencia f^0 en Ω, de tal manera que si los valores de los parámetros definen un punto perteneciente a la región entonces la solución asociada a f^0 se comportará en forma *razonable*. En [1] se presenta un algoritmo para aproximar la región de estabilidad para problemas de PEM-0-1. Destacamos ese trabajo porque el trato de la incertidumbre, en el sentido de considerar intervalos asociados al vector f asociado a las variables tipo y, es exactamente el considerado en este trabajo.

Cada enfoque tiene sus ventajas y desventajas. En este trabajo no se pretende demostrar ni sugerir que el análisis multiparamétrico es la vía correcta para enfrentrar la incertidumbre. Se trata solo de un aporte que creemos es razonablemente útil.

Algunos trabajos recientes en el contexto de la programación multiparamétrica son los siguientes:

(i) en [2] el caso paramétrico es considerado con **un solo** parámetro afectando a la función objetivo, la matriz de restricciones y el vector del lado derecho simultáneamente,

(ii) en [3] el caso multiparamétrico es considerado cuando los parámetros inciertos afectan **solamente al vector del lado derecho** y

(iii) en [4] se presenta un algoritmo para el problema multiparamétrico general con los parámetros inciertos afectando a la la función objetivo, la matriz

de restricciones y el vector del lado derecho simultáneamente. La naturaleza del algoritmo que puede verse en [4] (basado en Ramificación y Acotación seguido de Multiparametrización Lineal) lo hace por ahora poco práctico, como los mismos autores señalan, para problemas de grandes dimensiones.

Los algoritmos en [2],[3] y [4] utilizan como herramienta, entre otras, a la programación multiparamétrica restringida al caso lineal (sin variables enteras) puesto que permiten incertidumbre asociada a variables continuas. Esto, porque el campo de trabajo de los autores es el del control de las operaciones de plantas que incluyen procesos químicos. Nuestro enfoque está dirigido a la incertidumbre concentrada en las variables 0-1.

Los algoritmos que se presentarán trabajan tomando como base una secuencia apropiada de problemas no-paramétricos (sin incertidumbre) de tal manera que las soluciones de esos problemas, posiblemente aproximadas, nos proveen de una solución multiparamétrica ϵ-optimal. Este tipo de enfoque fue introducido en [5] para el caso de un solo parámetro y fue usado en [6],[7],[8],[9],[10] y[11] para distintas situaciones. Los algoritmos que se presentan en este trabajo están basados en [7].

El trabajo está organizado como se indica a continuación.

En la **Sección 2** se presentan dos estrategias básicas para resolver problemas (no paramétricos) de PEM-0-1 y PE-0-1: ***Ramificar y Acotar*** y ***Relajar y Fijar***.

En la **Sección 3** se presentan los resultados teóricos fundamentales y los algoritmos desarrollados para el caso multiparamétrico: el ***Nuevo* algoritmo multiparamétrico**, el **algoritmo multiparamétrico de *Ramificación y Acotación*** y el **algoritmo multiparamétrico basado en *Relajar y Fijar***.

La **Sección 4** está dedicada a los detalles relacionados con el uso de Ilog Cplex (paquete para resolver problemas de programación matemática) que es utilizado como herramienta para resolver los sucesivos problemas que aparecen al ejecutar nuestros algoritmos. Es importante señalar de inmediato que Ilog Cplex, de ahora en adelante Cplex, **no puede** resolver problemas multiparamétricos. Todos los problemas que aparecen al ejecutarse nuestros algoritmos son de Programación Lineal (PL), de PE-0-1 o de PEM-0-1 con todos los datos conocidos y para resolver esos problemas puntuales (no paramétricos) usamos Cplex. Nuestros algoritmos se encargan del análisis de los resultados de cada problema puntual para generar una a una las soluciones a ser incorporadas a la solución multiparamétrica y/o para generar el próximo problema puntual a ser resuelto.

En la **Sección 5** están los resultados computacionales para dos problemas clásicos de la literatura: el problema de Ubicación de Plantas sin restricciones de

capacidad (Simple Plant Location Problem) y el problema de Múltiples Mochilas con Costos Fijos (Fixed-Charge Multiple Knapsack Problem).

La **Sección 6** tiene un resumen y algunas ideas para futuros trabajos.

2 Programación Entera Mixta 0-1: estrategias básicas

Esta sección está basada en los apuntes de clase de los cursos Programación Matemática I y II dictados en la Escuela de Computacin de la Facultad de Ciencias de la Universidad Central de Venezuela. Todos los problemas considerados en esta sección son no-paramétricos, esto es todos los datos son conocidos y fijos. Por comodidad el problema genérico T que se considerará está escrito como $P(f)$ con las variables tipo x continuas salvo en la subsección **2.3** en la cual las variables tipo x son 0-1.

Sea T un problema de Programación Entera Mixta 0-1 (PEM-0-1) en (x, y) definido como sigue:

$$max \;\; c^t x + f^t y \quad s.a. \quad Ax + By \leq b, \;\; x \in \Re^p, \;\; x \geq 0, \;\; y \in \{0,1\}^n$$

con $c \in \Re^p$, $f \in \Re^n$, $b \in \Re^m$, $A \in \Re^{m \times p}$ y $B \in \Re^{m \times n}$.

Si no aparece el vector x y todo lo relacionado con el mismo el problema es de Programación Entera 0-1 (PE-0-1).

2.1 Los conceptos fundamentales: Ramificación, Relajación y Clausura

Ramificación: Se dice que T ha sido ramificado en los problemas T_0 y T_1 si:

$F(T) = F(T_0) \cup F(T_1)$ con $F(T_0) \cap F(T_1) = \emptyset$.

Lo usual, y así se hace en este trabajo, es que la ramificación esté definida con base en alguna de las variables binarias, digamos y_j, como sigue: T_0 es un problema que resulta de añadir a T la restricción adicional $y_j = 0$ y T_1 es un problema que resulta de añadir a T la restricción adicional $y_j = 1$. Decimos que T_0 y T_1 son descendientes de T. Los problemas T_0 y T_1 pueden ser a su vez ramificados (usando y_k con $k \neq j$) y sus descendientes también son descen-

dientes de T y así sucesivamente.

Para identificar un descendiente de T usamos una tripleta de conjuntos denotada $(K0, K1, K2)$. En $K0$ están los índices de las variables 0-1 que han sido seleccionadas durante el proceso de ramificación para tomar el valor cero y en $K1$ están los índices de las variables 0-1 que han sido seleccionadas durante el proceso de ramificación para tomar el valor uno. El resto de los índices de las variables 0-1 están en $K2$. El correspodiente descendiente se denotará $T_{(K0,K1,K2)}$.

Relajación: La relajación lineal de T, denotada $\bar{T}$, es un problema que resulta de sustituir $y \in \{0,1\}^n$ por $y \in \Re^n,\ 0 \leq y_j \leq 1\ \ \forall j$. Se puede considerar la Relajación Lineal de cualquier descendiente de T, digamos $T_{(K0,K1,K2)}$ sustituyendo $y \in \{0,1\}^n$ por $y \in \Re^n,\ 0 \leq y_j \leq 1\ \ \forall j \in K2$.

Los siguientes son resultados elementales válidos para $T = T_{(\emptyset,\emptyset,\{1,\cdots,n\})}$ y cualquiera de sus descendientes:

(i) Como $F(T_{(K0,K1,K2)}) \subseteq F(\bar{T}_{(K0,K1,K2)})$ entonces si $F(\bar{T}_{(K0,K1,K2)}) = \emptyset$ se infiere que $F(T_{(K0,K1,K2)}) = \emptyset$. Esto es: si la relajación es infactible el problema es infactible.

(ii) Supongamos que $F(\bar{T}_{(K0,K1,K2)})$ es no vacío y acotado (esto es válido para todos los problemas con los cuales se experimentó en este trabajo) con lo cual existe solución óptima para la relajación. Como $F(T_{(K0,K1,K2)}) \subseteq F(\bar{T}_{(K0,K1,K2)})$ entonces $(v(\bar{T}_{(K0,K1,K2)}) \geq v(T_{(K0,K1,K2)})$. Esto es: el valor óptimo de la relajación es mayor o igual al valor del problema. El resultado incluye el caso en el cual $F(T_{(K0,K1,K2)}) = \emptyset$ al usar la convención $v(T_{(K0,K1,K2)}) = -\infty$ en ese caso.

(iii) Supongamos que $F(\bar{T}_{(K0,K1,K2)})$ es no vacío y acotado y sea $(\bar{x}, \bar{y})$ una solución óptima. Como $F(T_{(K0,K1,K2)}) \subseteq F(\bar{T}_{(K0,K1,K2)})$ se infiere de inmediato que si $(\bar{x}, \bar{y}) \in F(T_{(K0,K1,K2)})$ (esto es si $\bar{y} \in \{0,1\}^n$) entonces $(\bar{x}, \bar{y})$ es óptima en $T_{(K0,K1,K2)}$. Esto es: si la solución óptima de la relajación es factible en el problema entonces es óptima en el problema.

Es importante mencionar que existen otras maneras de relajar que escapan a los objetivos de este trabajo.

Clausura:

(i) Si $F(\bar{T}_{(K0,K1,K2)}) = \emptyset$ entonces no tiene sentido considerar descendientes de $T_{(K0,K1,K2)}$ porque todos son infactibles. Se dice que $T_{(K0,K1,K2)}$ es clausurado por infactibilidad.

(ii) Sea *zmejor* el mejor valor conocido para una solución factible de T.

Supongamos que $v(\bar{T}_{(K0,K1,K2)} \leq zmejor$ entonces ni $T_{(K0,K1,K2)}$ ni ninguno de sus descendientes puede tener una solución óptima con valor superior a $zmejor$ de donde no tiene sentido considerar descendientes de $T_{(K0,K1,K2)}$. Se dice que $T_{(K0,K1,K2)}$ es clausurado por valor o por acotamiento.

(iii) Sea $(\bar{x}, \bar{y})$ una solución óptima de $\bar{T}_{(K0,K1,K2)}$ con $v(\bar{T}_{(K0,K1,K2)}) > zmejor$ que resulta factible en $T_{(K0,K1,K2)}$ (porque $\bar{y} \in \{0,1\}^n$). En tal caso $T_{(K0,K1,K2)}$ ha sido resuelto y no tiene sentido considerar a sus descendientes. Se dice que $T_{(K0,K1,K2)}$ es clausurado por factibilidad y se actualiza $zmejor$ haciendo $zmejor = v(\bar{T}_{(K0,K1,K2)}) = v(T_{(K0,K1,K2)})$.

2.2 Algoritmos de Ramificación y Acotación

Se inicia el proceso con $zmejor = -\infty$. Se resuelve la Relajación Lineal de T y si T es clausurado se finaliza con una solución óptima igual a la solución óptima obtenida para la relajación o con el mensaje de infactibilidad según el caso.

Si T no es clausurado se escoge una variable para ramificar y los descendientes ingresan a una lista de problemas pendientes.

Se selecciona un problema de la lista de problemas pendientes, denominado problema candidato, y se resuelve su relajación. Si el problema seleccionado es clausurado se selecciona el siguiente problema pendiente. Si el problema no es clausurado sus descendientes son añadidos a la lista de problemas pendientes y así sucesivamente.

Cada vez que un problema es clausurado por factibilidad se actualiza el valor de $zmejor$. Cuando la lista de problemas pendientes esté vacía el problema ha sido resuelto bien sea por haber obtenido una solución óptima con valor $zmejor$ o por por haber detectado que el problema es infactible.

Distintos criterios para seleccionar el próximo problema pendiente y para seleccionar la variable de ramificación conducen a distintos algoritmos, todos dentro de la misma estrategia general.

Puesto que el problema es 0-1 en lo que respecta a las variables enteras, y el conjunto de soluciones factibles se asume acotado, la finitud del algoritmo está garantizada finalizando con una solución óptima o con el mensaje de infactibilidad si ninguna solución factible fue encontrada ($zmejor = -\infty$).

El esquema general de un algoritmo de Ramificación y Acotación basado en relajaciones lineales es el siguiente:

Haga $zmejor = -\infty$ y haga $lista = \{(\emptyset, \emptyset, \{1, \cdots, n\})\}$.

mientras $lista \neq \emptyset$.

Seleccione el próximo problema candidato definido por $(K0, K1, K2)$.

Haga $lista = lista - \{(K0, K1, K2)\}$

Resuelva $\bar{T}_{(K0,K1,K2)}$.

Si $F(\bar{T}_{(K0,K1,K2)}) \neq \emptyset$ y $v(\bar{T}_{(K0,K1,K2)}) > zmejor$.

Sea $(\bar{x}, \bar{y})$ una solución óptima para $\bar{T}_{(K0,K1,K2)}$.

Si $\bar{y} \in \{0, 1\}^n$ entonces:

zmejor $= v(\bar{T}_{(K0,K1,K2)})$ y almacene $(\bar{x}, \bar{y})$.

si no

Seleccione $j : 0 < \bar{y}_j < 1$ y haga:

$lista = lista \cup \{(K0+j, K1, K2-j), (K0, K1+j, K2-j)\}\}$.

finsi

finsi

finmientras.

Mas adelante generalizaremos este esquema para hallar una solución multiparamétrica ϵ-optimal.

2.3 Heurística de Relajar y Fijar

La heurística Relajar y Fijar está diseñada para problemas de PE-0-1 con dos conjuntos de variables 0-1. Sea T un problema de PE-0-1 en (x, y) con la misma estructura definida anteriormente. Supongamos que $F(T)$ no es vacío. Primero se resuelve la relajación en la cual la condición 0-1 de las variables tipo x se elimina como antes pero no la de las variables tipo y (se trata entonces de una Relajación Lineal *parcial*. Sea $(\hat{x}, \hat{y})$ una solución óptima (que siempre existe porque $F(T)$ es no vacío). A continuación se fijan los valores de las variables tipo y en sus valores en $\hat{y}$ y el problema restringido de esa manera es resuelto. El problema así restringido podría resultar infactible. Si el problema restringido

resulta factible sea $(x^*, \hat{y})$ una solución óptima. En tal caso tenemos:

$$c^t x^* + f^t \hat{y} \leq v(T) \leq c^t \hat{x} + f^t \hat{y}$$

Si las cotas superior e inferior están suficientemente cerca entonces la heurística tuvo un resultado satisfactorio.

Mas adelante utilizaremos esta heurística para el caso multiparamétrico como herramienta para generar soluciones en el marco de un algoritmo que garantiza la obtención de una solución multiparamétrica ϵ-optimal.

3 Resultados Teóricos y los Algoritmos

En esta sección regresamos a la consideración de los problemas $P(f)$ con $f \in \Omega$.

Supongamos que $(x^{(1)}, y^{(1)}), \cdots, (x^{(r)}, y^{(r)})$ son tales que: $(x^{(i)}, y^{(i)}) \in F(P)$ para todo $i = 1, \cdots, r$. Si $(x^{(1)}, y^{(1)}), \cdots, (x^{(r)}, y^{(r)})$ no es una solución multiparamétrica ϵ-optimal entonces debe existir $f \in \Omega$ tal que:

$v(P(f)) - max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\} > \epsilon.$

Sea $Q^{(r)}$ un problema en $((x, y), f)$ definido como sigue:

$$max\ c^t x + f^t y - max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\} \quad s.a.$$

$$f \in \Omega, \quad (x, y) \in F(P)$$

Nótese que con $Q^{(r)}$ estamos buscando la máxima diferencia entre $v(P(f))$ y $max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\}$.

$Q^{(r)}$ fue considerado en [7] para resolver el problema multiparamétrico de PE-0-1. Los resultados presentados en [7] se extienden de inmediato al caso que nos ocupa y se presentan a continuación sin las demostraciones correspondientes. Para la tranquilidad del lector en este trabajo se presentan demostraciones sobre la correctitud de los algoritmos que son independientes de [7].

Observación 1 *(Crema, A. [7])*
Por construcción de $Q^{(r)}$ se tiene que :

(i) $F(Q^{(r)}) \neq \emptyset$.

(ii) Existe solución óptima para $Q^{(r)}$.

(iii) Si $((x,y),f)$ *es una solución óptima para* $Q^{(r)}$ *entonces* (x,y) *es una solución óptima para* $P(f)$.

(iv) $v(Q^{(r)}) \geq 0$.

(v) $v(Q^{(i)}) \geq v(Q^{(i+1)}) \;\; i = 1, \cdots, r-1$

(vi) $v(Q^{(r)}) \leq \epsilon$ *si y solo si* $(x^{(1)}, y^{(1)}), \cdots, (x^{(r)}, y^{(r)})$ *es una solución multiparamétrica* ϵ*-optimal.*

La presencia del término no lineal $f^t y$ en la función objetivo trae como consecuencia que para reescribir $Q^{(r)}$ como un problemaq de PEM-0-1 (con la finalidad de poderlo resolver) necesitamos un vector w de variables auxiliares para escribir $\sum_{j=1}^{n} w_j$ en lugar de $f^t y$ y $4n$ restricciones auxiliares, usando técnicas de linealización standard, para asegurarnos de: $w_j = f_j y_j \;\; \forall j$.

$Q^{(r)}$ puede ser reescrito como un problema de PEM-0-1 en $((x,y),z,w,f)$ como sigue (Crema A., ver [7]):

$$max\ c^t x + \sum_{j=1}^{n} w_j - z \quad s.a.$$

$$f \in \Omega, \quad (x,y) \in F(P)$$

$$z \geq c^t x^{(i)} + f^t y^{(i)} \;\; (i = 1, \cdots, r)$$

$$l_j y_j \leq w_j \leq u_j y_j \;\; (j = 1, \cdots, n)$$

$$l_j(1 - y_j) \leq f_j - w_j \leq u_j(1 - y_j) \;\; (j = 1, \cdots, n)$$

Sea $y \in [0,1]^n$. Sea $f^+(y)$ definido como sigue: $f^+(y)_j = u_j y_j + l_j(1 - y_j)$. Nótese que $f^+(y) \in \Omega$. Si $y \in \{0,1\}^n$ entonces $f^+(y)$ es conocido como *el escenario mas favorable para* y. Si $y, \hat{y} \in \{0,1\}$ sean $J_{0,1} = \{j \in \{1, \cdots, n\} : y_j = 0, \hat{y}_j = 1\}$ y $J_{1,0} = \{j \in \{1, \cdots, n\} : y_j = 1, \hat{y}_j = 0\}$.

El siguiente lema, conocido en la literatura de Programación Robusta, asegura que podemos considerar un subconjunto discreto de Ω.

Lema 1 *Sea* $(x,y) \in F(P)$. *Si* (x,y) *no es una solución óptima para* $P(f^+(y))$ *entonces* (x,y) *no es una solución óptima para* $P(f)$ *para todo* $f \in \Omega$.

Si bien el lema es conocido, las demostraciones que pueden verse en la literatura están restringidas a problemas específicos cuando en realidad es válido en general. La demostración que sigue es elemental y general.

Demostración (Crema,A.):
Supongamos que (x,y) no es solución óptima para $P(f^+(y))$. Sea $(\hat{x}, \hat{y})$ una

solución óptima para $P(f^+(y))$. Puesto que en tal caso $c^t\hat{x} + f^+(y)^t\hat{y} > c^t x + f^+(y)^t y$ entonces $c^t\hat{x} + \sum_{J_{0,1}} l_j > c^t x + \sum_{J_{1,0}} u_j$, en consecuencia si $f \in \Omega$ se tiene que:

$$c^t\hat{x} + f^t\hat{y} - c^t x - f^t y = c^t\hat{x} + \sum_{J_{0,1}} f_j - c^t x - \sum_{J_{1,0}} f_j \geq$$

$$c^t\hat{x} + \sum_{J_{0,1}} l_j - c^t x - \sum_{J_{1,0}} u_j > 0$$

Se sigue que (x, y) no es solución óptima para $P(f)$.•

Corolario 1 *(x, y) es una solución óptima para $P(f))$ para algún $f \in \Omega$ si y solo si (x, y) es una solución óptima para $P(f^+(y))$.*

En consecuencia vale la pena considerar un nuevo problema que considere únicamente escenarios que pertenecen a

$$\Omega^+ = \{f : f = f^+(y) \text{ para algún } y \in \{0,1\}^n\}$$

Sea $Q^{+(r)}$ un problema en (x, y) definido como sigue:

$$max\ c^t x + f^+(y)^t y - max\{c^t x^{(i)} + f^+(y)^t y^{(i)} : i = 1, \cdots, r\} \quad s.a.$$

$$(x, y) \in F(P)$$

Mas adelante se demostrará la equivalencia entre $Q^{+(r)}$ y $Q^{(r)}$ con lo cual el algoritmo presentado en [7] es válido para el caso que nos ocupa reemplazando $Q^{(r)}$ por $Q^{+(r)}$. Sin embargo podemos justificar un algoritmo basado en $Q^{+(r)}$ independientemente de [7].

Nótese que $Q^{+(r)}$ puede reescribirse como un problema de PEM-0-1 en $((x, y), z)$ como sigue:

$$max \quad c^t x + f^+(y)^t y - z \quad s.a.$$

$$z \geq c^t x^{(i)} + f^+(y)^t y^{(i)} \quad (i = 1, \cdots, r)$$

$$(x, y) \in F(P)$$

con $f^+(y)^t y = u^t y$ y $f^+(y)^t y^{(i)} = \sum_{j=1}^n ((u_j y_j + l_j(1 - y_j))y_j^{(i)})\ \forall i$.

Observación 2 *(Crema, A. y Peraza, E. [12]) Por construcción de* $Q^{+(r)}$ *tenemos que :*

(i) $F(Q^{+(r)}) \neq \emptyset$.

(ii) Existe solución óptima para $Q^{+(r)}$.

(iii) $v(Q^{+(r)}) \geq 0$.

(iv) $v(Q^{+(i)}) \geq v(Q^{+(i+1)}) \;\; i = 1, \cdots, r-1$

Se necesita un lema auxiliar, también conocido en la literatura de Programación Robusta, para demostrar el resultado fundamental que sustenta a todos los algoritmos que presentaremos:

Lema 2 *Sea* $f \in \Omega$ *y sean* $y, \hat{y} \in \{0,1\}^n$ *entonces*

$$f^t\hat{y} - f^t y \leq f^+(\hat{y})^t\hat{y} - f^+(\hat{y})^t y$$

Demostración:
$f^t\hat{y} - f^t y = \sum_{J_{0,1}} f_j - \sum_{J_{1,0}} f_j \leq \sum_{J_{0,1}} u_j - \sum_{J_{1,0}} l_j = f^+(\hat{y})^t\hat{y} - f^+(\hat{y})^t y$ •

El siguiente lema sustenta a todos los algoritmos que se presentarán. La demostración original puede verse en Peraza E. y Crema A. ([12]). La demostración que se presenta a continuación (de Crema,A.) es mucho mas simple que la original:

Lema 3 *(Crema, A. y Peraza, E. [12])* $v(Q^{+(r)}) \leq \epsilon$ *si y solo si* $(x^{(1)}, y^{(1)}), \cdots, (x^{(r)}, y^{(r)})$ *es una solución multiparamétrica* ϵ*-optimal.*

Demostración (Crema, A.):
Si $(x^{(1)}, y^{(1)}), \cdots, (x^{(r)}, y^{(r)})$ es una solución multiparamétrica ϵ-optimal entonces

$$c^t x + f^t y - max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\} \leq \epsilon$$

para todo $f \in \Omega$ y para todo $(x, y) \in F(P)$. Entonces:

$$c^t x + f^+(y)^t y - max\{c^t x^{(i)} + f^+(y)^t y^{(i)} : i = 1, \cdots, r\} \leq \epsilon$$

para todo $(x, y) \in F(P)$. Se sigue que $v(Q^{+(r)}) \leq \epsilon$.

Supongamos que $v(Q^{+(r)}) \leq \epsilon$. Sea $f \in \Omega$ y sea (x, y) una solución óptima para $P(f)$. Usando el lema 2 y algunas manipulaciones algebráicas elementales se tiene que:

$$c^t x + f^t y - max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\} =$$

$$min\{c^t x + f^t y - c^t x^{(i)} - f^t y^{(i)} : i = 1, \cdots, r\} \leq$$

$$min\{c^t x + f^+(y)^t y - c^t x^{(i)} - f^+(y)^t y^{(i)} : i = 1, \cdots, r\} =$$

$$c^t x + f^+(y)^t y - max\{c^t x^{(i)} + f^+(y)^t y^{(i)} : i = 1, \cdots, r\} \leq v(Q^{+(r)}) \leq \epsilon$$

En consecuencia,

$$v(P(f)) \geq max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\} \geq v(P(f)) - \epsilon \ \bullet$$

En el siguiente lema se presentan algunas relaciones entre $Q^{(r)}$ y $Q^{+(r)}$. El resultado en (iv) resultó una total sorpresa y fue ***descubierto*** mientras se experimentaba para este TEG.

Lema 4 *(Crema,A.)*
(i) $v(Q^{(r)}) = v(Q^{+(r)})$.

(ii) Si $((x, y), f)$ *es una solución óptima para* $Q^{(r)}$ *entonces* (x, y) *es una solución óptima para* $Q^{+(r)}$ *y para* $P(f^+(y))$.

(iii) Si (x, y) *es una solución óptima para* $Q^{+(r)}$ *entonces* $((x, y), f^+(y))$ *es una solución óptima para* $Q^{(r)}$ *y* (x, y) *lo es para* $P(f^+(y))$.

(iv) Supongamos que $Q^{(r)}$ *y* $Q^{+(r)}$ *están escritos como problemas de PEM-0-1 y sean* $\bar{Q}^{(r)}$ *y* $\bar{Q}^{+(r)}$ *sus relajaciones lineales entonces* $v(\bar{Q}^{+(r)}) = v(\bar{Q}^{(r)})$.

Demostración:
(i) Puesto que $\Omega^+ \subseteq \Omega$ se tiene que $v(Q^{(r)}) \geq v(Q^{+r})$. Sea $((x, y), f)$ una solución óptima para $Q^{(r)}$. Utilizando la definición de los problemas, la optimalidad de $((x, y), f)$, el lema 2 y algunas manipulaciones algebráicas elementales se tiene que:

$$v(Q^{(r)}) = c^t x + f^t y - max\{c^t x^{(i)} + f^t y^{(i)} : i = 1, \cdots, r\} =$$

$$min\{c^t x + f^t y - c^t x^{(i)} - f^t y^{(i)} : i = 1, \cdots, r\} \leq$$

$$min\{c^t x + f^+(y)^t y - c^t x^{(i)} - f^+(y)^t y^{(i)} : i = 1, \cdots, r\} =$$
$$c^t x + f^+(y)^t y - max\{c^t x^{(i)} + f^+(y)^t y^{(i)} : i = 1, \cdots, r\} \leq v(Q^{+(r)})$$

(ii) Sea $((x, y), f)$ una solución óptima para $Q^{(r)}$. De la demostración de (i) tenemos que (x, y) es una solución óptima para $Q^{+(r)}$. De la observación 1 (x, y) es una solución óptima para $P(f)$ y del lema 1 (x, y) es una solución óptima para $P(f^+(y))$.

(iii) Sea (x, y) una solución óptima para $Q^{+(r)}$. De la demostración de (i) y del lema 1 tenemos que $((x, y), f^+(y))$ es una solución óptima para $Q^{(r)}$ y (x, y) lo es para $P(f^+(y))$.

(iv) Sea $((\bar{x}, \bar{y}), \bar{z})$ una solución óptima para $\bar{Q}^{+(r)}$ entonces $((\bar{x}, \bar{y}), \bar{z}, \bar{w}, f^+(\bar{y})) \in F(\bar{Q}^{(r)})$ con $\bar{w}_j = u_j \bar{y}_j \ \forall j$. En consecuencia:

$$v(\bar{Q}^{+(r)}) = c^t \bar{x} + u^t \bar{y} - \bar{z} = c^t \bar{x} + \sum_{j=1}^{n} \bar{w}_j - \bar{z} \leq v(\bar{Q}^{(r)})$$

Sea $((\bar{x}, \bar{y}), \bar{z}, \bar{w}, \bar{f}))$ una solución óptima para $\bar{Q}^{(r)}$ entonces $\bar{w}_j \leq u_j \bar{y}_j$ y $\bar{w}_j - \bar{f}_j \leq -l_j(1 - \bar{y}_j) = u_j \bar{y}_j - f^+(\bar{y})_j$. Se sigue que

$\bar{w}_j - \bar{f}_j y_j^{(i)} \leq u_j \bar{y}_j - f^+(\bar{y})_j y_j^{(i)}$ y $\sum_{j=1}^{n} \bar{w}_j - \bar{f}^t y^{(i)} \leq u^t \bar{y} - f^+(\bar{y})^t y^{(i)}$.

Se tiene entonces que:

$$v(\bar{Q}^{(r)}) = c^t \bar{x} + \sum_{j=1}^{n} \bar{w}_j - \bar{z} = c^t \bar{x} + \sum_{j=1}^{n} \bar{w}_j - max\{c^t x^{(i)} + \bar{f}^t y^{(i)} : i = 1, \cdots, r\} =$$

$$min\{c^t \bar{x} + \sum_{j=1}^{n} \bar{w}_j - c^t x^{(i)} - \bar{f}^t y^{(i)} : i = 1, \cdots, r\} \leq$$

$$min\{c^t \bar{x} + u^t \bar{y} - c^t x^{(i)} - f^+(\bar{y})^t y^{(i)} : i = 1, \cdots, r\} =$$

$$c^t \bar{x} + u^t \bar{y} - max\{c^t x^{(i)} + f^+(\bar{y})^t y^{(i)} : i = 1, \cdots, r\} \leq v(\bar{Q}^{+(r)}) \ \bullet$$

Nótese que el lema 3 puede ser visto como una consecuencia del lema 4.

Puesto que $\{0, 1\}^n$ es un conjunto finito, la observación 2 y el lema 3 demuestran que el siguiente algoritmo encuentra una solución multiparamétrica

ϵ-optimal en un número finito de pasos. Toda solución multiparamétrica ϵ-optimal debe ser finita, de tal manera que encontrarla en un número finito de pasos no parece una tarea digna de ser mencionada. Debe aclararse, sin embargo, que se puede demostrar pero escapa a los alcances de este trabajo, que si $\epsilon = 0$ entones está garantizado que el número de pasos es exactamente el mínimo con un algoritmo basado en el presentado pero eliminando apropiada e implícitamente soluciones innecesarias por lo cual cabe esperar un buen comportamiento con $\epsilon > 0$.

El *nuevo* algoritmo multiparamétrico (Peraza,E. y Crema,A. ([12])

Encuentre $(x^{(1)}, y^{(1)}) \in F(P)$, sea $r = 1$ y haga $\delta = 2\epsilon$.

mientras $\delta > \epsilon$

Resuelva $Q^{+(r)}$ para obtener (x, y).

Sea $\delta = v(Q^{+(r)})$.

Si $\delta > \epsilon$ haga $(x^{(r+1)}, y^{(r+1)}) = (x, y)$ y $r = r + 1$.

finmientras

Puesto que $v(\bar{Q}^{+(r)}) = v(\bar{Q}^{(r)})$ y ya que $Q^{+(r)}$ no tiene las n variables adicionales y las $4n$ restricciones adicionales que se usan para reescribir $Q^{(r)}$ como un problema de PEM-0-1 entonces podemos esperar que el ***nuevo*** algoritmo multiparamétrico tenga mucho mejor rendimiento que el ***viejo*** algoritmo ([7]) definido usando $Q^{(r)}$. La experiencia computacional confirma esa presunción: ahora podemos resolver problemas con dimensiones mucho mas grandes que aquellas usadas anteriormente.

Sin embargo, $Q^{+(r)}$ puede ser un problema dificil de resolver y en consecuencia algoritmos alternativos basados en relajaciones podrían ser útiles. A continuación presentamos dos de ellas. Otras alternativas serán bosquejadas en la sección 6.

3.1 El algoritmo multiparamétrico de *Ramificación y Acotación*

Supongamos que $X = \{x : x \in \Re^p, \quad x \geq 0\}$. Un problema en el arbol de búsqueda estará definido por la tripleta $(K0, K1, K2)$ con $K0 = \{j : y_j = 0\}$, $K1 = \{j : y_j = 1\}$ y $K2 = \{1, \cdots, n\} - (K0 \cup K1)$. Sea $Q^{+(r)}_{(K0,K1,K2)}$ la restricción de $Q^{+(r)}$ definida por $(K0, K1, K2)$ y sea $\bar{Q}^{+(r)}_{(K0,K1,K2)}$ su relajación

lineal.

Nuestro algoritmo multiparamétrico de ***Ramificación y Acotación*** se rige por las siguientes reglas:

(1) Si $F(\bar{Q}^{+(r)}_{(K0,K1,K2)}) = \emptyset$ entonces $(K0, K1, K2)$ es clausurado (no tiene descendientes).

(2) Si $v(\bar{Q}^{+(r)}_{(K0,K1,K2)}) \leq \epsilon$ entonces $(K0, K1, K2)$ es clausurado (no tiene descendientes).

(3) Sea $(\bar{x}, \bar{y})$ una solución ópima para $\bar{Q}^{+(r)}_{(K0,K1,K2)}$ con

$v(\bar{Q}^{+(r)}_{(K0,K1,K2)}) > \epsilon$ y $\bar{y} \in \{0,1\}^n$ entonces: $(K0, K1, K2)$ **NO es clausurado** y hacemos $(x^{(r+1)}, y^{(r+1)}) = (\bar{x}, \bar{y})$. Nótese que con esta regla el problema que se está resolviendo pasa de $Q^{+(r)}$ a $Q^{+(r+1)}$ pero **se mantiene el mismo arbol de búsqueda, esto es: la lista de problemas pendientes se mantiene y todos los problemas que fueron clausurados se mantienen en la misma condición**.

Algunas observaciones son suficientes para justificar nuestras reglas:

Observación 3 *Puesto que* $F(Q^{+(r+j)}_{(K0,K1,K2)}) \subseteq F(Q^{+(r)}_{(K0,K1,K2)})$ *para todo* $j \geq 1$ *entonces se tiene que:*

(i) Si $Q^{+(r)}_{(K0,K1,K2)}$ *es clausurado por infactibilidad lo mismo debe hacerse con* $Q^{+(r+j)}_{(K0,K1,K2)}$ *para todo* $j \geq 1$ *lo cual justifica la regla 1.*

(ii) Si $v(\bar{Q}^{+(r)}_{(K0,K1,K2)}) \leq \epsilon$ *entonces* $v(\bar{Q}^{+(r+j)}_{(K0,K1,K2)}) \leq \epsilon$ *para todo* $j \geq 1$ *lo cual justifica la regla 2.*

(iii) Sea $(\bar{x}, \bar{y})$ *una solución óptima para* $\bar{Q}^{+(r)}_{(K0,K1,K2)}$ *con*

$v(\bar{Q}^{+(r)}_{(K0,K1,K2)}) > \epsilon$ *y* $\bar{y} \in \{0,1\}^n$ *entonces:*

podría ocurrir $v(Q^{+(r+j)}_{(K0,K1,K2)}) > \epsilon$ *y en consecuencia problemas descendientes de* $(K0, K1, K2)$ *pueden contener soluciones factibles necesarias para obtener una solución multiparamétrica* ϵ*-optimal lo cual justifica la regla 3.*

El algoritmo multiparamétrico de ***Ramificación y Acotación*** está definido como sigue respetando la terminología presentada en la sección 2:

Encuentre $(x^{(1)}, y^{(1)}) \in F(P)$, sea $r = 1$ y haga $lista = \{(\emptyset, \emptyset, \{1, \cdots, n\})\}$.

mientras $lista \neq \emptyset$.

Seleccione el próximo problema candidato definido por $(K0, K1, K2)$.

Resuelva $\bar{Q}^{+(r)}_{(K0,K1,K2)}$.

Si $F(\bar{Q}^{+(r)}_{(K0,K1,K2)}) \neq \emptyset$ y $v(\bar{Q}^{+(r)}_{(K0,K1,K2)}) > \epsilon$.

Sea $((\bar{x}, \bar{y}), \bar{z})$ una solución óptima para $\bar{Q}^{+(r)}_{(K0,K1,K2)}$.

Si $\bar{y} \in \{0, 1\}^n$ entonces:

Haga $(x^{(r+1)}, y^{(r+1)}) = (\bar{x}, \bar{y})$ y $r = r + 1$.

si no

Seleccione $j : 0 < \bar{y}_j < 1$ y haga:

$lista = lista - \{(K0, K1, K2)\}$

$lista = lista \cup \{(K0+j, K1, K2-j), (K0, K1+j, K2-j)\}\}$.

finsi

si no

Haga $lista = lista - \{(K0, K1, K2)\}$.

finsi

finmientras.

3.2 El algoritmo multiparamétrico basado en *relajar y fijar*

El algoritmo multiparamétrico basado en *relajar y fijar* está basado en la heurística para problemas de PE-0-1 con dos conjuntos de variables 0-1 presentada en la sección 2 ([13]). Proponemos una idea similar para obtener una solución multiparamétrica ϵ-optimal.

Supongamos que $X = \{0, 1\}^p$. Sea $\hat{Q}^{+(r)}$ la relajación lineal parcial de $Q^{+(r)}$ en la cual la condición 0-1 de las variables tipo x es eliminada. Sea $(\hat{x}, \hat{y})$ la

correspondiente solución. Sea $P(u)/\hat{y}$ el correspondiente problema restringido de $P(u)$ añadiendo las restricciones $y = \hat{y}$.

Se define el algoritmo multiparamétrico basado en ***relajar y fijar*** como sigue:

Encuentre $(x^{(1)}, y^{(1)}) \in F(P)$, haga $r = 1$ y sea $\delta = 2\epsilon$.

mientras $\delta > \epsilon$

Resuelva $\hat{Q}^{+(r)}$ y haga $\delta = v(\hat{Q}^{+(r)})$.

Si $\delta > \epsilon$

Sea $(\hat{x}, \hat{y})$ una solución óptima para $\hat{Q}^{+(r)}$.

Eliminar $\hat{y}$ en $\hat{Q}^{+(r)}$.

Resuelva $P(u)/\hat{y}$.

Si $F(P(u)/\hat{y}) \neq \emptyset$ sea $(x, \hat{y})$ una solución óptima.

Haga $(x^{(r+1)}, y^{(r+1)}) = (x, \hat{y})$ y $r = r + 1$

finsi

finsi

finmientras

Algunas observaciones son necesarias: (i) $P(u)/\hat{y}$ puede resultar infactible, (ii) si $(\hat{x}, \hat{y})$ es una solución óptima para $\hat{Q}^{+(r)}$ entonces podría ocurrir que $\hat{y} \in \{y^{(1)}, \cdots, y^{(r)}\}$ con $\delta > \epsilon$. En consecuencia eliminamos $\hat{y}$ de $F(\hat{Q}^{+(r)})$. Ahora $\hat{Q}^{+(r)}$ puede resultar infactible ($v(\hat{Q}^{+(r)}) = -\infty$) pero en ese caso se encontró una solución multiparamétrica ϵ-optimal y el algoritmo está bien definido. Con las nuevas restricciones ($y \notin \{y^{(1)}, \cdots, y^{(r)}\}$) añadidas a $\hat{Q}^{+(r)}$ el algoritmo genera una solución multiparamétrica ϵ-optimal en un número finito de pasos.

4 El paquete Cplex para resolver problemas de programación matemática

Cplex ([14]), es una herramienta para solucionar problemas de programación matemática en general. Su nombre se debe al algoritmo simplex y el lenguaje

de programación C. Sin embargo, no existen demasiadas limitaciones a la hora de escoger el lenguaje a ser utilizado. De esta manera, se puede escoger entre un lote compatible con el paquete como son: C,C++, java, .NET. MATLAB, Python, etc... De igual forma, Cplex es compatible con los sistemas operativos de preferencia general (Linux y Windows). En el caso de este trabajo se usó la versión 12.2 de Cplex, utilizando el entorno Visual Studio Express 2010 en la versión 7.0 de Windows. Para consultar detalles de instalación e integración de componentes revisar [14].

El uso de Cplex varía de acuerdo a la finalidad que se le quiera dar en un momento dado. En este sentido, se pueden solucionar problemas de PL (como $\bar{Q}^{+(r)}_{(K0,K1,K2)}$ en el algoritmo multiparamétrico de ***Ramificación y Acotación***), PE-0-1 (como $P(u)/\hat{y}$ en el algoritmo multiparamétrico basado en ***Relajar y Fijar***) y PEM-0-1 (como $Q^{+(r)}$ en el ***Nuevo*** algoritmo multiparamétrico y $\hat{Q}^{+(r)}$ en el algoritmo multiparamétrico basado en ***Relajar y Fijar)*** entre otros.

Cabe destacar que la generación de los problemas puntuales a ser resueltos por Cplex y el control del arbol de decisiones en el caso de Ramificación y Acotación son exclusiva responsabilidad de nuestros códigos y no de Cplex.

A continuación se muestra la estructura general que utiliza Cplex para tratar los problemas sin importar el tipo de los mismos.

$$
\begin{array}{ll}
\textit{maximizar (o minimizar)} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\\
\textit{sujeto a :} & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \ * \ b_1 \\
& a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \ * \ b_2 \\
& \cdots \\
& a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \ * \ b_m \\
\\
\textit{con las siguientes cotas} & lb_1 \leq x_1 \leq ub_1 \\
& \cdots \\
& lb_n \leq x_n \leq ub_n
\end{array}
$$

En donde $*$ puede ser $\leq, \geq o =$ y las cotas superiores ub_i y cotas inferiores lb_i pueden ser cualquier número real positivo o negativo incluyendo los valores infinito y menos infinito.

A continuación se dará una explicación detallada de cómo representar en Cplex un problema como se ha descrito. Es importante senãlar que no hay exclusividad en la forma en la cual se suministra la matriz de restricción a. Esto significa que puede representarse por filas (como en los códigos desarrollados para este trabajo) y se agregan una a una o, en su defecto, se construye la matriz completa y se suministra dado un método de Cplex (como en los pro-

totipos en MATLAB desarrollados con anticipación a este trabajo). Agregar las restricciones una a una tiene la ventaja de que cuando el problema a resolverse difiere del anterior tan solo porque se agregará una fila (que ocurre a menudo en nuestros algoritmos) entonces Cplex puede hacer uso de herramientas de reoptimización para resolver el nuevo problema utilizando información proveniente del problema anterior.

Se explicará el primer método (por filas) usado en el presente trabajo. Antes, una breve explicación de los datos necesarios para la resolución de los problemas.

$Coeficientes\ de\ la\ función\ objetivo$	$c_1, c_2, \cdots, c_n$
$Coeficientes\ de\ las\ restricciones$	$a_{11}, a_{12}, \cdots, a_{1n}$ $\cdots$ $a_{m1}, a_{m2}, \cdots, a_{mn}$
$Vector\ de\ lado\ derecho$	$b_1, b_2, \cdots, b_m$
$Cotas\ superiores$	$ub_1, ub_2, \cdots, ub_n$
$Cotas\ inferiores$	$lb_1, lb_2, \cdots, lb_n$

Estructuras principales de Cplex para modelar problemas:

(i) *Tipos de datos* :

Para definir datos es recomendable utilizar los definidos por el paquete. De esta forma, se tienen 3 importantes: **IloInt**, **IloFloat** y **IloBool**. Intuitivamente, hacen analogía a los tipos de datos comunes en el lenguaje C++: **int**, **float** y **bool**.

Si se requiere un arreglo de alguno de los tipos de datos, Cplex provee tipos de dato también para arreglos los cuales son: **IloIntArray**, **IloFloatArray** y **IloBoolArray**. Es importante señalar que hay especificaciones especiales para determinar el tamaño de los arreglos y pueden consultarse en la bibliografía.

(ii) *Variables de decisión*

Las variables de decisión x se definen mediante los tipos de dato: **IloIntVar**, **IloFloatVar** y **IloBoolVar**. La primera para variables enteras, la segunda para variables reales y la tercera para variables binarias. Al igual que los tipos de datos fundamentales para tener un arreglo de variables de decisión se utilizan los tipos de datos que provee Cplex: **IloIntVarArray**, **IloFloatVarArray** y **IloBoolVarArray**.

Cada una de las variables de decisión, con límites lb_i y ub_i, se definirán de la siguiente manera:

Tipo de dato	Formato
Variables Binarias	**IloBoolVar**(env , 0 , 1 , ILOBOOL);
Variables Enteras	**IloIntVar**(env , l_i , u_i , ILOINT);
Variables Reales	**IloFloatVar**(env, l_i , u_i , ILOFLOAT);

Observación 4 *Es importante señalar que los arreglos de variables de decisión pueden contener variables de cualquier tipo. De esta forma, se pueden colocar variables binarias, reales y enteras en un mismo arreglo.*

Observación 5 *El primer parámetro en la definición de las variables* **env** *hace referencia a la variable ambiente (environment) de Cplex y su finalidad puede consultarse en la bibliografía.*

(iii) *Función Objetivo*

Para definir la función objetivo fusionamos las variables de decisión con los tipos de datos fundamentales en lo que se conoce, según la documentación, como una **expresión**. Las expresiones tienen un tipo de dato particular definido como **IloExpr**.

Una **IloExpr** se formará haciendo el producto escalar ($c^t x$) entre el vector de coeficientes de la función objetivo c y el vector de variables de decisión x. Por ejemplo, para definir una **IloExpr** dado un arreglo c de coeficientes de tipo **IloIntArray** y un arreglo variables de decisión de tipo **IloIntVarArray** ambos de tamaño n, se haría lo siguiente:

```
IloExpr fo;
IloIntVarArray x(env ,n, l_i , u_i); //l_i ≤ x_i ≤ u_i
IloIntArray c(env,n); //siendo n el tamaño del arreglo

for(IloInt i = 0; i < n; i++)
  fo+=c[i]*x[i];
```

Una vez que conocemos como crear una **IloExpr**, para tener la función objetivo del modelo simplemente se usa el tipo de dato **IloObjetive** de la siguiente manera (asumiendo que fo es nuestra **IloExpr**):

```
IloObjetive Objective = IloMaximize(env,fo);
```

Observación 6 *Todas las estructuras que se generen se deben agregar a un modelo. Los modelos tienen el tipo de dato* **IloModel** *y poseen una cantidad numerosa de métodos que son usados en la codificación, sin embargo, el método*

más importante es el add que nos permite agregar las estructuras que se han ido explicando de la forma modelo.add(estructura);

(iii) *Restricciones*

Al igual que con la función objetivo, para poder definir restricciones del modelo se necesita una expresión **IloExpr**. El tipo de dato a utilizar para las restricciónes se llama **IloRange**. Para las mismas, se utiliza el vector de lado derecho b y su estructura general es la siguiente:

IloRange r(env, l_i , fo , l_s); *//siendo l_i y l_s los límites de la restricción*

En el caso de una restricción que utilizara el vector de lado derecho b y su limite inferior fuese $-\infty$ se escribiría de la siguiente manera:

IloRange r(env, -**IloInfinity** , fo , b_i); *//para toda i*

Es importante señalar que, para efectos de la implementación, cada una de las restricciones estará formada por el producto escalar del vector de variables de decisión y una fila de la matriz de los coeficientes de las restricciones. Esto es, $r_i = a_i^t x$. En notación de Cplex, fo+=a[i][j]*x[i]; //para todo j.

(iv) *El problema*

Una vez que se conocen todas las estructuras asociadas al problema, hay que saber cómo se resuelve el mismo. Las estructuras son añadidas en el **IloModel**, sin embargo, es desde otra estructura **IloCplex** donde se invoca al método para solucionar el problema.

Asumiendo que el modelo está completo, es decir, tiene la función objetivo y sus respectivas restricciones, para poder resolver el problema hay que extraer el modelo y asignárselo al **IloCplex** mediante el método **extract**. La notación es la siguiente:

IloModel Pl; *//asumiendo que está completo el modelo*
IloCplex Plx;
Plx.extract(Pl);

Ya está asignado el modelo al **IloCplex** y ahora, simplemente, se invoca al método **solve** para solucionarlo. La notación es la siguiente:

IloModel Pl; *//asumiendo que está completo el modelo*
IloCplex Plx;
Plx.extract(Pl);
Plx.solve();

Para conocer los valores que toman las variables de decisión x una vez invocado el método **solve** se utiliza el método **getValue**:

IloIntVarArray x(env ,n, l_i , u_i); //$l_i \leq x_i \leq u_i$
IloIntArray xSol(env,n); //*siendo n el tamaño del arreglo*
IloCplex Plx;

for(**IloInt** i = 0; i < n; i++) xSol[i]=Plx.getValue(x[i]);

Finalmente, para conocer el valor del problema de nuevo utilizamos el tipo de dato **IloCplex** asumiendo que ya extraímos el modelo. Un ejemplo de cómo extraer el valor es el siguiente:

IloNum valor;
IloCplex Plx;

valor = Plx.getObjValue();

5 Resultados Computacionales

Los experimentos fueron llevados a cabo sobre dos computadores personales, el primero (Hardware-1) con un procesador Intel-centrino de 2.2 GHz con 4.00 GB RAM y el segundo (Hardware-2) con un procesador Intel-i7 de 2.0 GHz con 12 GB RAM. Los algoritmos se desarrollaron en C++ usando Cplex 12.2 ([14]). Se seleccionaron dos problemas de la literatura para evaluar los algoritmos. No estamos afirmando que los algoritmos propuestos son el mejor enfoque para enfrentar la incertidumbre en tales problemas. Nuestro enfoque es general, no utiliza para nada la estructura particular del problema bajo evaluación, salvo la naturaleza continua o discreta de X, y los experimentos están diseñados para intentar mostrar que trabaja en forma razonable en distintos problemas y que por consiguiente vale la pena continuar los esfuerzos en la misma dirección. Los problemas seleccionados son el problema de Ubicación de Plantas sin restricciones de capacidad en su versión de maximización (Simple Plant Location Problem, SPLP [15]) y el problema de Múltiples Mochilas con Costos Fijos (Fixed-Charge Multiple Knapsack Problem, FCHMKP [16]). Ambos problemas tienen muchísimas aplicaciones prácticas pero su explicación va más allá de los alcances de este trabajo.

El ***nuevo*** algoritmo multiparamétrico, el algoritmo multiparamétrico basado en ***Relajar y Fijar*** y el algoritmo multiparamétrico de ***Ramificación y Acotación*** serán denotados como ***NUEVO***, ***REFIJAR*** y ***RyA*** respectivamente.

Presentaremos la experiencia computacional con la versión multiparamétrica

para el SPLP con ***NUEVO*** y ***RyA*** y para el FCHMKP con ***REFIJAR***.

NUEVO tiene un pésimo rendimiento al resolver FCHMKP y los correspondientes resultados no se presentan ante la imposibilidad de resolver, usando ese algoritmo, problemas de las dimensiones presentadas . Este comportamiento es compatible con el hecho de que Cplex 12.2 es pésimo resolviendo problemas no paramétricos tipo FCHMKP tal como se reporta en la literatura. Usando ***REFIJAR*** evitamos la estructura intratable del problema. Nótese que ***REFIJAR*** no tiene sentido para el SPLP porque las variables tipo x toman valores 0-1 aún sin exigirlo.

Se dispone de prototipos de los algoritmos usando Cplex 12.2, pero desde MATLAB, desarrollados por A.Crema ([17]) sobre Hardware-1 sin aprovechar las posibles reoptimizaciones que se logran al usar C++ para implantar algoritmos sobre Cplex 12.2, porque en la versión sobre MATLAB cada problema durante la ejecución de los algorimos es independiente de todos los demás. Cuando sea posible se compararán los resultados usando la versión MATLAB y la versión Cplex 12.2. sobre o no la misma máquina. La razón para mantener experimentos sobre distintas máquinas es que Cplex es altamente dependiente de la máquina utilizada, lo cual no es siempre una desventaja, y resultaba atractivo verificar el comporetamiento de nuestros algoritmos usando distintos lenguajes (MATLAB vs C++) y distintas máquinas (Hardware-1 vs Hardware-2). Las sucesiones de soluciones generadas no tienen porqué coincidir puesto que debido a la diferencia de precisión entre las máquinas las variables de ramificación seleccionadas pueden variar de una máquina a otra produciendo cambios dramáticos en el arbol de decisión. Claro está que de parar los algoritmos al alcanzar la tolerancia relativa ($\epsilon = \lambda v(P(l))$ solicitada se garantiza una solución multipatamérica ϵ-optimal. Si alguno de los algoritmos se detiene por alcanzar el tiempo máximo permitido o el número máximo de soluciones a ser generadas entonces los errores relativos pueden diferir.

Si se usa un algoritmo de Ramificación y Acotación para resolver cualquier problema entonces una *tolerancia relativa* es usada para podar el arbol de búsqueda (el valor por defecto es 10^{-4} al usar Cplex 12.2). Sea α la *tolerancia relativa*. Nótese que los resultados teóricos son válidos si $\alpha = 0$. Sin embargo, si se usa $\alpha = 0$ el esfuerzo computacional para alcanzar una solución multiparamétrica ϵ-optimal podría no tener sentido (incluso bajo la premisa inicial de este trabajo según la cual se dispone del tiempo sufuciente antes de que las decisiones deban ser tomadas) . Con la finalidad de asegurar, en principio, que se genera una solución con el error relativo exigido usando $\alpha > 0$ se sustituye ϵ por $\epsilon/(1+\alpha)$ para detener los algoritmos. Además, cuando usamos ***REFIJAR***, ϵ es usado como *tolerancia absoluta* para resolver $P(u)/\hat{y}$.

En algunas gráficas, cuando se comparan algoritmos, los resultados asociados a un conjunto de problemas independientes se presentan con lineas continuas en lugar de puntos solo para facilitarle al lector la comparación. Las lineas horizon-

tales que aparecen en las gráficas corresponden a los promedios de los resultados.

Sea λ el *error relativo* a ser utilizado. Sea $z(l)$ el valor suboptimal obtenido al resolver $P(l)$ con la *tolerancia relativa* α. En todos los problemas se cumple $z(l) > 0$ y entonces se define $\epsilon = \lambda z(l)$. Sean $tmax$ y $rmax$ cotas para el tiempo de CPU y el número de soluciones generadas respectivamente. Se usan las siguientes reglas para parar los algoritmos aún si $v(Q^{+(r)}) > \epsilon$: o bien el tiempo de CPU supera $tmax$ después de resolver $Q^{+(r)}$ (o una relajación de $Q^{+(r)}$) o $r \geq rmax$. Cuando los algoritmos paran con $v(Q^{+(r)}) > \epsilon$ entonces calculamos el *error relativo* correspondiente ($\lambda = (1+\alpha)(v^{+(r)})/z(l)$) con $v^{+(r)}$ el valor obtenido resolviendo $Q^{+(r)}$ (o una relajación de $Q^{+(r)}$) con la *tolerancia relativa* α. Sea $(x^{(1)}, y^{(1)})$ la solución suboptimal para $P(l)$ usando α.

Sean $I = \{1, \cdots, n\}$ y $J = \{1 \cdots, m\}$.

En el problema SPLP en (x, y) se está en presencia de n posibles Plantas para abastecer a m Centros de Demanda. Para poder usar la i-ésima planta se debe pagar un costo fijo f_i^0 (independiente de las decisiones sobre los centros de demanda). Si el j-ésimo centro de demanda es asignado a la i-ésima planta se tiene una ganancia igual a c_{ij}. Cada Centro de Demanda debe ser asignado a una única planta. Las variables de decisión tipo y identifican si la i-ésima planta puede utilizarse ($y_i = 1$) o no ($y_i = 0$). Las variables de decisión tipo x identifican si el j-ésimo centro de demanda es asignado a la i-ésima planta ($x_{ij} = 1$) o no ($x_{ij} = 0$). El problema consiste en maximizar la ganancia neta (suma de las ganancias asociadas a la asignación de los centros de demanda a las plantas menos la suma de los costos fijos asociados a las plantas que se utilicen). Las restricciones $x_{ij} - y_i \leq 0$ para todos los i, j impiden que un centro de demanda sea asignado a una planta si no se paga el costo fijo de la misma.

La formulación utilizada para el SPLP en (x, y) es:

$$max \ \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} - \sum_{i=1}^{n} f_i^0 y_i \ \ s.a.$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ \ \forall j \in J$$

$$x_{ij} - y_i \leq 0, \ \ x_{ij} \in \{0,1\} \ \ \forall i \in I \ \ \forall j \in J, \ \ y \in \{0,1\}^n$$

Los datos fueron generados (utilizando un generador diseñado por Crema, A.) como sigue: n puntos se generaron al azar en el cuadrado unitario, sea d_{ij} la distancia en norma 1 desde el punto i hasta el punto j, $D_j \sim U(Dl, Du)$, $c_{ij} = 3D_j/(1+d_{ij})$, $f_i^0 = Fmin + s_i(Fmax - Fmin)$ con $s_i = (\sum_{j=1}^{n} c_{ij} - fmin)/(fmax - fmin)$, $fmin = min\{\sum_{j=1}^{n} c_{ij} : \ \ i = 1, \cdots, n\}$ y $fmax = max\{\sum_{j=1}^{n} c_{ij} : i = 1, \cdots, n\}$. A continuación se generan los intervalos alrededor de f_i^0 como

sigue: $l_i = -(1+\beta)f_i^0$ y $u_i = -(1-\beta)f_i^0$. Finalmente todos los datos son redondeados al entero mas cercano.

Nótese que se puede eliminar la condición 0-1 de las variables tipo-x en un SPLP y lo mismo vale para el correspondiente $Q^{+(r)}$. Se usó ***NUEVO*** y ***RyA*** para resolver la versión multiparamétrica de SPLP usando la estrategia de ***Búsqueda en profundidad*** para seleccionar el próximo problema candidato (escogiendo primero la rama correspondiente al valor 1 de la variable de ramificación) y con la siguiente regla para seleccionar la variable de ramificación: sea $(\bar{x}, \bar{y})$ la solución óptima para $\bar{Q}^{+(r)}_{(K0,K1,K2)}$, si $(u_j - l_j)min\{\bar{y}_j, 1-\bar{y}_j\} = max\{(u_k - l_k)min\{\bar{y}_k, 1-\bar{y}_k\} : 0 < \bar{y}_k < 1\}$ entonces usamos j para definir los descendientes de $Q^{+(r)}_{(K0,K1,K2)}$. Salvo por los términos $(u_k - l_k)$ el criterio de ramificación es standard. Se incluyeron los términos mencionados para privilegiar a las variables con mayor incertidumbre. Otros criterios similares pueden usarse sustituyendo $(u_k - l_k)$ por $(u_k - l_k)/u_k$, $(u_k - l_k)/l_k$, etc...

El primer conjunto de problemas (Conjunto I) fue generado con $n \in \{100, 150\}$, $Dl = 1, Du = 100, Fmin \in \{100, 150, 200\}$, $Fmax \in \{400, 600, 800\}$. El porcentaje de perturbación se fijó en 5 ($\beta = 0.05$). Se generó un problema para cada combinación para un total de dieciocho problemas. Se usó ***NUEVO*** con $\alpha = 0.0001,\ \lambda = 0.005,\ tmax = 1800$ segundos y $rmax = 100$. Además, se usó ***RyA*** con $\lambda = 0.005$.

Se compararon tres versiones de ***NUEVO*** como sigue: sobre Hardware-2 según se describió en la sección 4 (***NUEVO-c++-8***), con la misma versión pero sobre Hardware-1 (***NUEVO-c++-2***) y con el prototipo del algoritmo desarrollada usando MATLAB sobre Hardware-1 (***NUEVO-MATLAB***).

Los promedios de los errores relativos utilizando ***NUEVO-c++-8***, ***NUEVO-c++-2*** y ***NUEVO-MATLAB*** resultaron 0.0050, 0.0051 y 0.0052 respectivamente. Los peores valores fueron 0.0055, 0.0060 y 0.0065, todos en el mismo problema (problema 4). La diferencia notable en el problema 4 se explica porque ***NUEVO-c++-8*** paró al alcanzar 100 soluciones, ***NUEVO-c++-2*** se detuvo al alcanzar el tiempo máximo habiendo generado 70 soluciones y ***NUEVO-MATLAB*** paró al alcanzar el tiempo máximo habiendo generado 57 soluciones. Los promedios de tiempo de CPU resultaron 480,890 y 1000 segundos respectivamente. La limitante del tiempo máximo permitido tiende a igualar los promedios de ***NUEVO-c++-2*** y ***NUEVO-MATLAB*** aun cuando la tendencia clara es que el primero es más rápido que el segundo. El promedio de soluciones generadas fue 41.88, 35.66 y 33.05 respectivamente (ver figuras 1,2 y 3).

El segundo conjunto de problemas (Conjunto II) fue generado con $n \in \{50, 75, 100, 125, 120, 200\}$, $Dl = 100, Du = 10000, Fmin \in \{25000, 50000, 75000\}$ y $Fmax = 100000$. El porcentaje de perturbación se fijó en 5 ($\beta = 0.05$). Se generó un problema para cada combinación para obtener 18 problemas. Se

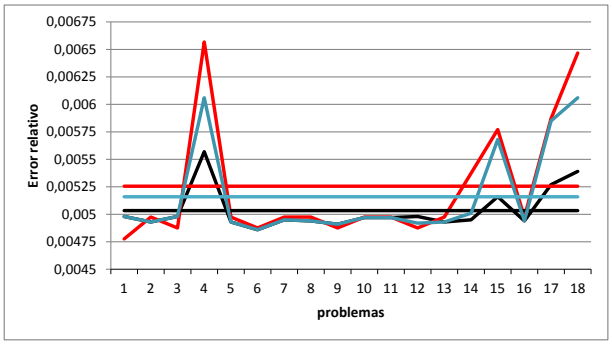

Figura 1: SPLP, Conjunto I, errores relativos usando ***NUEVO-c++-8*** (en negro), ***NUEVO-c++-2*** (en azul) y ***NUEVO-MATLAB*** (en rojo)

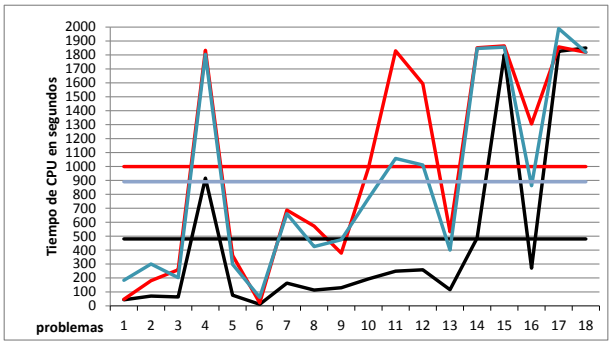

Figura 2: SPLP, Conjunto I, tiempo de CPU en segundos usando ***NUEVO-c++-8*** (en negro), ***NUEVO-c++-2*** (en azul) y ***NUEVO-MATLAB*** (en rojo)

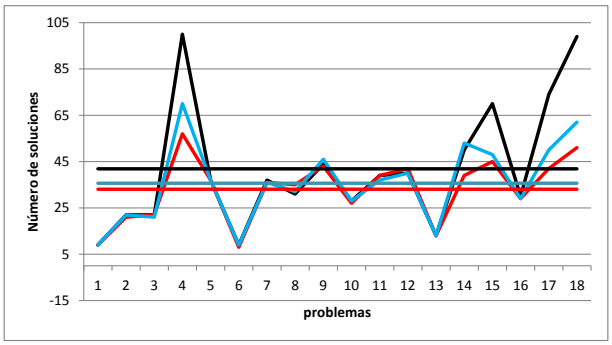

Figura 3: SPLP, Conjunto I, soluciones generadas usando ***NUEVO-c++-8*** (en negro), ***NUEVO-c++-2*** (en azul) y ***NUEVO-MATLAB*** (en rojo)

usó ***NUEVO-c++-8*** con $\alpha = 0.0001$, $\lambda = 0.005$, $tmax = 1800$ segundos y $rmax = 100$. Las figuras 4, 5 y 6 corresponden a los errores relativos, tiempo de CPU en segundos y número de soluciones generadas respectivamente.

El tercer conjunto de problemas (Conjunto III), idéntico al Conjunto I, se utilizó para comparar ***NUEVO-c++-8*** con ***RyA*** sobre la misma máquina. Se usó ahora ***RyA*** con ($\alpha = 0.0001$, $\lambda = 0.005$, $tmax = 1800$ segundos y $rmax = 200$). Los promedios de tiempo de CPU resultaron 480 y 263 segundos respectivamente. Se colocó rmax en 200 para permitirle a ***RyA*** un buen rendimiento habida cuenta de que es capaz de generar más soluciones que ***NUEVO-c++-8*** en el mismo tiempo. ***RyA*** fue superior en 15 de los 18 problemas (ver figura 7) y logró garantizar el error relativo exigido ($\lambda = 0.005$) en todos los casos. ***NUEVO-c++-8*** falló en cuatro casos, tres de los cuales se interrumpieron al llegar al tiempo máximo (problemas 15,17 y 18) alcanzando los errores relativos 0.0051, 0.0052 y 0.0054 respectivamente. Es claro que ***NUEVO-c++-8*** podría haber alcanzado el error relativo exigido en el problema 4 si se le permite generar hasta 200 soluciones, pero para el mismo problema ***RyA*** alcanzó el error exigido en un tiempo claramente inferior. La relación promedio entre el tiempo de CPU usando ***RyA*** y ***NUEVO-c++-8*** fue 0.66 (ver figura 8). El número de soluciones que necesita ***RyA*** para garantizar el error relativo exigido es en general, como era de esperarse, superior al que necesita ***NUEVO-c++-8*** (ver figura 9).

El cuarto conjunto de problemas (Conjunto IV) fue generado con $n = 200$, $Dl = 1, Du = 100, Fmin \in \{100, 150, 200\}$ y $Fmax = 400, 600, 800$. El porcentaje de perturbación se fijó en 5 ($\beta = 0.05$). Se generó un problema para cada combinación para obtener 9 problemas. Se usó ***NUEVO-c++-8*** y ***NUEVO-***

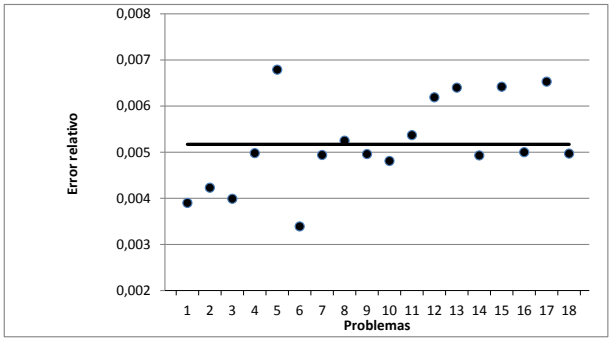

Figura 4: SPLP, Conjunto II, errores relativos usando ***NUEVO-c++-8***

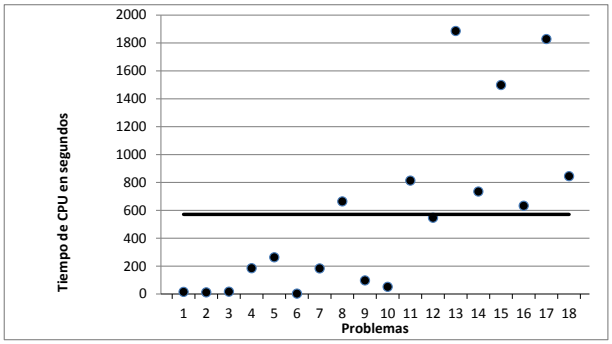

Figura 5: SPLP, Conjunto II, tiempo de CPU en segundos usando ***NUEVO-c++-8***

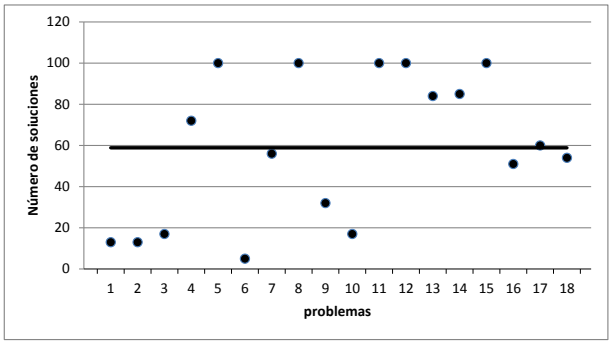

Figura 6: SPLP, Conjunto II, número de soluciones generadas usando ***NUEVO-c++-8***

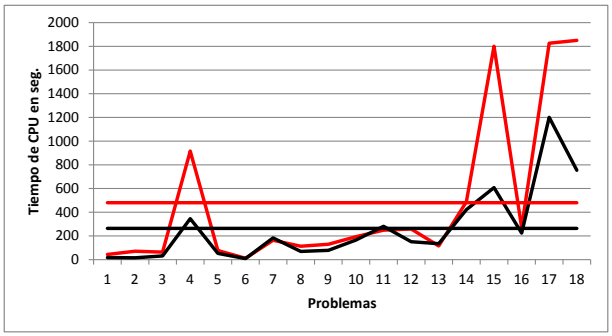

Figura 7: SPLP, Conjunto III, tiempos de CPU en seg usando ***RyA*** (en negro) y ***NUEVO-c++-8*** (en rojo)

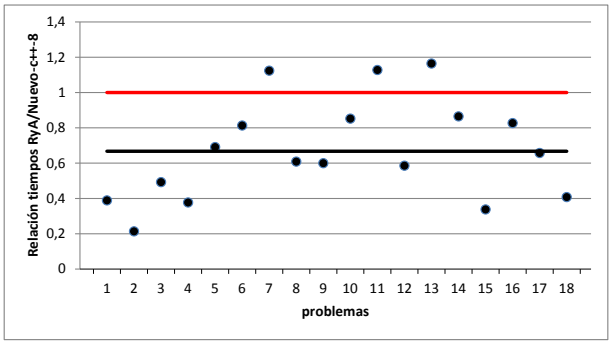

Figura 8: SPLP, Conjunto III relación tiempos de CPU en seg usando ***RyA*** y ***NUEVO-c++-8***

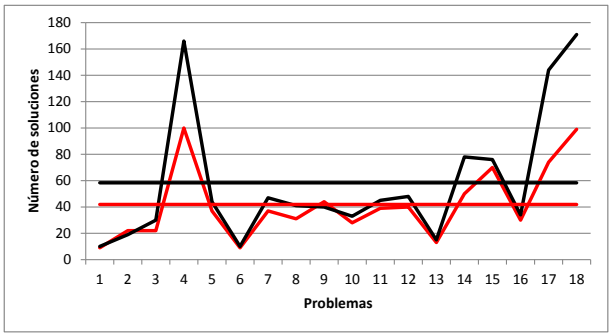

Figura 9: SPLP, Conjunto III, número de soluciones generadas usando ***RyA*** (en negro) y ***NUEVO-c++-8*** (en rojo)

MATLAB con $\alpha = 0.0001,\ \lambda = 0.005,\ tmax = 1800$ segundos y $rmax = 100$. Ambos algoritmos, y en todos los casos, se detuvieron al llegar al tiempo máximo permitido. La figura 10 corresponde a los errores relativos obtenidos. Los promedios alcanzados resultaron 0.0064 y 0.0071 respectivamente. La diferencia a favor de ***NUEVO-c++-8*** se explica porque es capaz de generar más soluciones en el mismo tiempo (ver figura 11).

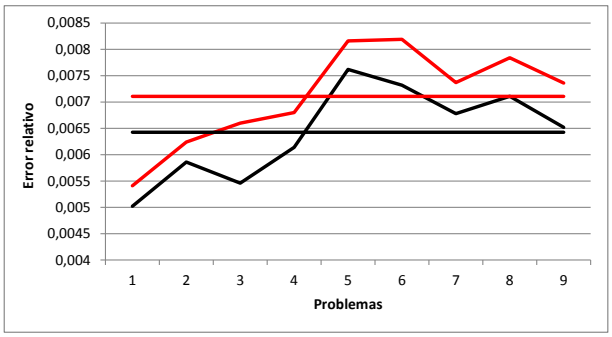

Figura 10: SPLP, Conjunto IV, errores relativos usando ***NUEVO-c++-8*** (en negro) y ***NUEVO-MATLAB*** (en rojo)

En el caso del SPLP ***RyA*** resultó superior a ***NUEVO***. La diferencia a favor de la primera estrategia se acentúa en los problemas con mayor dificultad. En algunos problemas de poca dificultad la segunda estrategia supera a la primera por lo cual no se debe descartar a ninguna de las dos. Como era de esperarse la implantación de ***NUEVO*** usando c++ vía Cplex 12.2 es superior a la versión con MATLAB vía Cplex 12.2 con independencia del hardware utlizado. De nuevo la diferencia es mayor en la medida que se incrementa la dificultad del problema. Este último comportamiento parece ser explicado por el hecho de que la versión MATLAB no utiliza herramientas de reoptimización que sí usa la versión en c++ pero tales herramientas resultan una carga innecesaria si el problema es de poca dificultad y su utilidad se manifiesta en la medida que la dificultad se incrementa.

En el problema FCHMKP en (x, y) se está en presencia de n mochilas en las cuales pueden o no incluirse m artículos. Cada artículo puede incluirse en a lo más una de las mochilas. Para utilizar la i-ésima mochila debe pagarse un costo fijo igual a f_i^0 (independiente de los artículos que se incluyan en ella). Si el j-ésimo artículo se incluye en alguna mochila se tiene una ganancia igual a c_j y se utilizan w_j unidades de peso. La i-ésima mochila resiste un peso

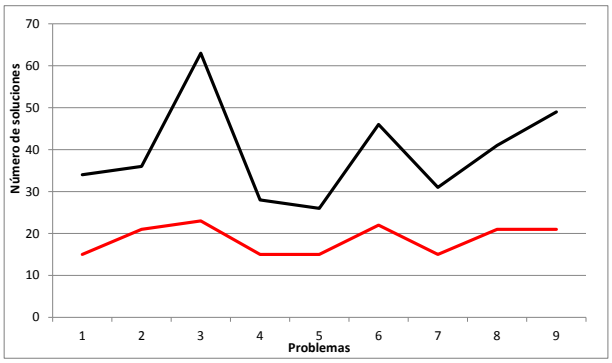

Figura 11: SPLP, conjunto IV, número de soluciones generadas usando ***NUEVO-c++-8*** (en negro) y ***NUEVO-MATLAB*** (en rojo)

máximo igual a W_i. Las variables de decisión tipo y identifican si la i-ésima mochila puede utilizarse ($y_i = 1$) o no ($y_i = 0$). Las variables de decisión tipo x identifican si el j-ésimo artículo se incluye en la i-ésima mochila ($x_{ij} = 1$) o no $x_{ij} = 0$). El problema consiste en maximizar la ganancia neta (suma de las ganancias de los artículos incluidos en las mochilas menos la suma de los costos fijos asociados a las mochilas que se utilicen) respetando las capacidades de las mochilas. Las restricciones $\sum_{j=1}^{m} w_j x_{ij} \leq W_i y_i$ para todos los i impiden que un artículo se incluya en una mochila por la cual no se ha pagado y controlan el peso máximo que puede incluirse en la mochila. Las restricciones $\sum_{i=1}^{n} x_{ij} \leq 1$ para todos los j controlan que cada artículo sea asignado a lo más a una mochila.

La formulación usada para el FCHMKP en (x, y) es la siguiente ([16]):

$$max \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_j x_{ij} - \sum_{i=1}^{n} f_i^0 y_i \quad s.a.$$

$$\sum_{j=1}^{m} w_j x_{ij} \leq W_i y_i \quad \forall i \in I$$

$$\sum_{i=1}^{n} x_{ij} \leq 1 \quad \forall j \in J, \quad x_{ij} \in \{0,1\} \quad \forall i \in I \quad \forall j \in J, \quad y \in \{0,1\}^n$$

Los datos fueron generados como sigue respetando un procedimiento que puede verse en [16]: $w_j \sim U(1,1000)$, $c_j \sim U(1,1000)$ y $f_j^0 = \rho_j W_j$ con $W_j = 500n\delta(\gamma_j / \sum_{i=1}^{n} \gamma_j)$, $\gamma_i \sim U(0,1)$ y $\rho_j \sim U(0.5,1.5)$. A continuación

se generan los intervalos alrededor de f_j^0 como sigue: $l_j = -(1+\beta_j)f_j^0$ y $u_j = -(1-\beta_j)f_j^0$ con $\beta_j \sim U(0,\beta)$. Finalmente todos los datos son redondeados al entero mas cercano.

El quinto conjunto de problemas (Conjunto V) se generó con $n \in \{25, 35\}$, $m \in \{500, 750, 1000\}$ y $\delta \in \{0.25, 0.35, 0.50\}$. El porcentaje de perturbación se fijó en 7.5 ($\beta = 0.075$). Se generaron cuatro problemas para cada combinación para un total de 72 problemas. Se usó ***REFIJAR*** con $\alpha = 0.005$, $\lambda = 0.005$, $tmax = 2400$ segundos y $rmax = 50$.

Se compararon dos versiones de ***REFIJAR*** como sigue: sobre Hardware-2 según se describió en la sección 4 (***REFIJAR-c++-8***) y con el prototipo del algoritmo desarrollada usando MATLAB sobre Hardware-1 (***REFIJAR-MATLAB***). En todos los casos se alcanzó el error relativo exigido. El promedio del tiempo de CPU fue de 59.72 y 168.61 segundos respectivamente. La cantidad de problemas que requirieron más de 250 segundos fue 5 y 11 respectivamente. Con más de 500 segundos fue 1 y 5 respectivamente. Con más de 750 segundos fue 0 y 4 respectivamente. Los peores casos fueron de 724.32 (problema 1) y 2366.22 (problema 65) segundos respectivamente. En 67 problemas el tiempo de ***REFIJAR-c++-8*** fue mejor que el de ***REFIJAR-MATLAB***. El promedio de la relación entre los tiempos fue 0.467 (ver figuras 12 y 13).

En el caso del FCHKPM ***REFIJAR*** permitió resolver problemas imposibles de resolver con ***NUEVO***. ***REFIJAR-c++-8*** es superior a ***REFIJAR-MATLAB***. La diferencia a favor de la primera implantación se acentúa, de nuevo, en los problemas con mayor dificultad. En algunos problemas de poca dificultad la segunda implantación supera a la primera, incluso con la diferencia de hardware, por lo cual no se debe descartar a ninguna de las dos.

Finalmente se generaron 48 problemas (Conjunto VI) como en el Conjunto V (tres para cada combinación de datos) pero con el porcentaje de perturbación en 5 ($\beta = 0.05$). Se usó ***REFIJAR-c++-8*** y ***REFIJAR-MATLAB*** con $\alpha = 0.005$, $\lambda = 0.005$, $tmax = 2400$ segundos y $rmax = 50$. La dificultad de los problemas disminuye y con ello la diferencia de los resultados con las dos implantaciones (ver figura 14). Ambas versiones garantizaron el error relativo exigido. Los promedios de tiempo fueron 32,74 y 38,69 segundos respectivamente. ***REFIJAR-c++-8*** resultó superior a ***REFIJAR-MATLAB*** en 33 de los 48 problemas. Los problemas 36 y 45 presentan tiempos atípicos respecto al resto para la primera y segunda implantación y son un ejemplo de cómo el hardware puede influir en el comportamiento de algoritmos que usen Cplex 12.2 como herramiento de cálculo porque los árboles de decisión al ejecutarse un algoritmo de Ramificación y Acotación pueden variar dramáticamente producto de distintas decisiones de ramificación.

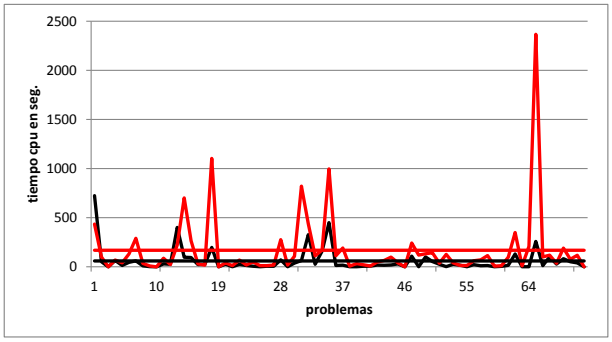

Figura 12: FCHKPM, Conjunto V, tiempo de CPU en segundos usando ***REFIJAR-c++-8*** (en negro) y ***REFIJAR-MATLAB*** (en rojo)

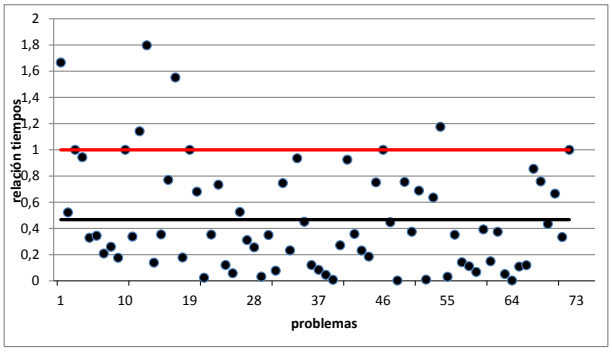

Figura 13: FCHKPM, Conjunto V, relación entre tiempo de CPU en segundos usando ***REFIJAR-c++-8*** y ***REFIJAR-MATLAB***

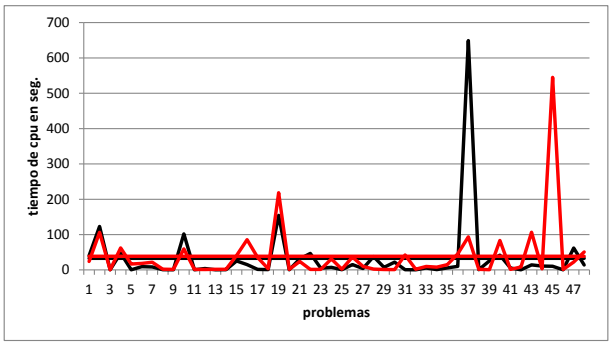

Figura 14: FCHKPM, Conjunto VI, tiempo de CPU en segundos usando ***REFIJAR-c++-8*** (en negro) y ***REFIJAR-MATLAB*** (en rojo)

6 Resumen y posibles extensiones

Hemos diseñado algoritmos para aproximar la solución del problema multiparamétrico de PEM-0-1 relativo a la función objetivo. Se consideró la incertidumbre para los parámetros que definen el vector de la función objetivo correspondiente a un subconjunto de variables 0-1 asumiendo que cada parámetro pertenece a un intervalo conocido. Mediante el uso del concepto del *escenario mas favorable para y* se rediseñó el *viejo* algoritmo que puede verse en [7]. Con el *nuevo* algoritmo se pueden resolver problemas con dimensiones mayores a las consideradas previamente. Sin embargo, el problema a ser resuelto en cada paso del *nuevo* algoritmo puede ser todavía dificil de resolver y entonces vale la pena considerar algoritmos basados en relajaciones. Se presentaron dos alternativas que usan relajaciones basados en la heurística de *relajar y fijar* y en el enfoque de *Ramificación y Acotación*.

Se presentó una experiencia computacional con los tres algoritmos ***NUEVO***, ***REFIJAR*** y ***RyA*** para aproximar la solución de la versión multiparamétrica de dos conocidos problemas de la literatura: SPLP y FCHKPM. Esperamos que nuestra experiencia computacional haya mostrado que nuestro enfoque es promisorio y queremos destacar que enfoques diseñados para problemas no paramétricos fueron usados sin mayores cambios en su filosofía para aproximar la solución de problemas multiparamétricos.

Se pueden diseñar algoritmos especializados con la finalidad de resolver problemas difíciles con dimensiones grandes usando un enfoque para resolver $Q^{+(r)}$ (o una relajación de $Q^{+(r)}$) que utilice expresamente la estructura de $P(f)$.

REFIJAR va claramente en esa dirección porque el problema restringido al fijar las variables tipo y puede ser resuelto con algún algoritmo especializado (en el caso del FCHMKP habría que resolver un problema de mochilas múltiples sin costos fijos para el cual existen algoritmos especializados). Nuestro enfoque constituye, desde este punto de vista, una metodología general.

En el caso de ***RyA*** el próximo paso, como siempre en estos casos, es considerar distintos criterios para seleccionar el próximo problema a ser evaluado y para seleccionar la variable de ramificación. Como siempre la bondad de cualquier estrategia dependerá del problema a ser resuelto.

Por último un ***comentario de A.Crema***: El uso de enfoques bien conocidos no está restringido a *Relajar y Fijar* y *Ramificación y Acotación*. Los detalles escapan a los objetivos de este trabajo pero puede señalarse que enfoques basados en:

-la *Descomposición de Benders* (con los cortes generados todavía válidos cuando pasamos de $Q^{+(r)}$ a $Q^{+(r+1)}$),

-la *Búsqueda Local* (resolviendo $Q^{+(r)}$ en una vecindad de un punto de referencia, retornando a la búsqueda global resolviendo $Q^{+(r)}$ cuando la búsqueda local falle para generar un nuevo punto de referencia y así sucesivamente) y

-*Relajaciones Lagrangeanas* (dualizando las restricciones relativas a las soluciones generadas para definir un problema lagrangeano no-paramétrico con la misma estructura del problema original $P(f)$)

pueden ser diseñados facilmente.

7 Referencias

[1] Fatma Kılınç-Karzan, Alejandro Toriello, Shabbir Ahmed, George Nemhauser, Martin Savelsbergh: 'Approximating the stability region for binary mixed.integer program', Operations Research Letters 37 (2009) 250-254.

[2] Alexander Mitsos: 'Parametric mixed integer 0-1 linear programming: the general case of a single parameter', European Journal of Operational Research 94 (3) (2009) 663-686.

[3] : Viveck Dua and Efstratius N. Pistikopoulos: 'An algorithm for the solution of Multiparametric Mixed Integer Programming Problem', Annals of Operations Researh 99 (1-4) (2000) 123-139.

[4] Li,Z and M.G.Ierapetritou: 'A new methodology for the general multiparametric mixed integer linear programming', Ind. Eng. Chem. Res. 46 (2007) 5141-5151.

[5] Larry Jenkins: 'Parametric methods in integer linear programming', Annals of Operations Research 27 (1990) 77-96.

[6] Alejandro Crema: 'A contraction algorithm for the multiparametric integer linear programming problem', European Journal of Operational Research' 101 (1) (1997)130-139.

[7] Alejandro Crema: 'An algorithm for the multiparametric 0-1 integer linear programming', European Journal of Operational Research 125 (2000) 18-24.

[8] Crema, A.: 'An algorithm for the multiparametric 0-1-integer linear programming problem relative to the constraint matrix.', Operations Research Letters (27) (1) (2000) 13-19.

[9] Alejandro Crema: 'The multiparametric 0-1-integer linear programming problem: a unified approach', European Journal of Operational Research (139) (2002) 511-520.

[10] José Luis Quintero y Alejandro Crema: 'An algorithm for multiparametric min max 0-1-integer programming problems relative to the objective function', RAIRO Oper. Res. (39) (2005) 243-252.

[11] José Luis Quintero y Alejandro Crema. 'An algorithm for multiparametric 0-1-Integer Programming problems relative to a generalized min max objective function', RAIRO Oper. Res. 43 (2009) 1-12

[12] Edgar Hugo Peraza y Alejandro Crema: Proyecto de Tesis Doctoral de Edgar Hugo Peraza para optar al grado de Doctor en Computación, Escuela de Computación, Facultad de Ciencias, UCV (2009).

[13] Laurence A. Wolsey: 'Integer Programming', Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience publication (1998).

[14] IBM ILOG CPLEX Optimization Studio V12.2.

[15] Francisco Barahona and Fabian A. Chudak: 'Near-optimal solutions to large-scale facility location problems',Discrete Optimization 2 (2005) 35-50

[16] Takeo Yamada and Takahiro Takeoka: 'An exact algorithm for the fixed-charge multiple knapsack problem', European Journal of Operational Research 192 (2009) 700-705.

[17] Alejandro Crema, Edgar Hugo Peraza y Fernando Crema: ' Approximating the solution for the multiparametric 0-1-mixed integer linear programming problem with interval data', Optimization Online (August, 2012).