TESIS

PROGRAMACION LINEAL ENTERA

EL PROBLEMA KNAPSACK

por

FRANCISCO RUIZ DE FRANCISCO

Ingeniero Industrial por la E.T.S. de I.I.

de la Universidad de Sevilla

presentada en la

ESCUELA SUPERIOR DE INGENIEROS INDUSTRIALES

de la

UNIVERSIDAD DE SEVILLA

para la obtención del

Grado de Doctor Ingeniero Industrial

SEVILLA, SEPTIEMBRE 1979

A mis padres
en memoria

TESIS

PROGRAMACION LINEAL ENTERA

EL PROBLEMA KNAPSACK

por

Francisco Ruiz de Francisco

Ingeniero Industrial por la E.S. de I.I.

de la Universidad de Sevilla

presentada en la

ESCUELA SUPERIOR DE INGENIEROS INDUSTRIALES

de la

UNIVERSIDAD DE SEVILLA

para la obtención del

Grado de Doctor Ingeniero Industrial

Sevilla, Septiembre 1979.

El autor desea expresar su más profundo reconocimiento a agradecimiento al Profesor J. Larrañeta por la labor desarrollada en el planteamiento y desarrollo de este trabajo.

Al Profesor J. Aracil, ponente de esta tesis, por el apoyo que en todo momento hemos recibido.

A los compañeros de los departamentos de Organización Industrial de la E.S.I.I. de Sevilla y Madrid por sus críticas y comentarios. Especialmente al Profesor J.M. Bueno por sus comentarios al primer borrador.

Al Ministerio de Universidades e Investigación que ha financiado, mediante una Beca de Formación de Personal Investigador esta tesis.

INDICE

CAPITULO VIII. CONCLUSIONES Y REFERENCIAS.

ANEXOS.

CAPITULO I

INTRODUCCION Y SUMARIO

1.1. El Tema

Para resolver gran cantidad de problemas de organización se formalizan modelos matemáticos de optimización. La formalización de estos es antigua pero su utilización en gran escala - es reciente. La posibilidad de resolver los modelos planteados limita el desarrollo de las aplicaciones prácticas. Dependiendo de las características del problema este será resoluble o no con los algoritmos existentes.

Dentro de la programación matemática una de las lineas más estudiadas y de mayor interés es la programación lineal -- entera. Sobre esta se centra este trabajo haciendo especial énfasis en la busqueda de metodologías prácticas de resolución de problemas.

De los sistemas modelados como problemas de programación matemática el noventa por ciento son problemas de programación lineal |37|. Estos son problemas en los que se pretende obte

ner el óptimo de una función lineal en un poliedro convexo (con-- junto de restricciones lineales). Durante los últimos treinta y - cinco años su área de aplicación se ha expandido y han ido desa-- rrollándose códigos de ordenador altamente eficientes para resol- ver problemas de varios miles de variables y restricciones. Gran cantidad de modelos pueden encontrarse en cualquiera de las múlti ples obras dedicadas a las aplicaciones de la programación lineal (|22|, |26|, |30|, |33|, |36|, |42|, |44|, |70|, |92|,|118|). Res pecto a los códigos comerciales de ordenador existentes puede en- contrarse una extensa relación en |37|.

Así como la mayoría de los problemas de programación matemática son problemas lineales, el ochenta-noventa por ciento de estos, son problemas de programación lineal entera. Un progra- ma lineal entero es simplemente un programa lineal en el que algu na o todas las variables han de tomar valores enteros. En resumen, es añadir restricciones de integridad a un programa lineal. La ne cesidad de introducir la restricción de integridad puede ser debi da a dos motivos : condiciones físicas o condiciones lógicas. Las condiciones físicas son las debidas a la característica de indivi sibilidad de las variables (p.e. número de hombres) y las condi-- ciones lógicas las debidas a discontinuidades lógicas del sistema (p.e. se construye una fábrica o no se construye).

El ámbito de aplicaciones de estos modelos es amplísimo. Documentación sobre su empleo en la planificación, medicina, educación, industria, etc. pueden encontrarse en |4|, |5|, |29|, |36|, |70|, |103|, |105|, |112|, |118|.

El desarrollo de los códigos de ordenador para la resolución de problemas lineales enteros ha sido más lento que el de las aplicaciones. En 1958 Gomory implementó el primer algoritmo riguroso capaz de resolver modelos sencillos con solo unas decenas de variables. Aun hoy los códigos comerciales existentes emplean tiempos de CPU prohibitivos para la resolución de modelos de cierta envergadura, a pesar del desarrollo de nuevos métodos y de los aumentos de capacidad y velocidad de los computadores.

1.2. Resolución de Problemas Lineales Enteros

Los múltiples algoritmos existentes para resolver problemas lineales enteros se agrupan en métodos, según sus características comunes. La clasificación de los métodos es distinta según los autores. En el presente trabajo distinguimos cuatro grupos: métodos de cortes, métodos de enumeración, programación dinámica y métodos de transformación.

Los métodos de cortes consisten básicamente en añadir al problema, relajada la condición de integridad, un nuevo conjun

to de restricciones (cortes), de forma que el óptimo del problema así obtenido será entero. Naturalmente para que el óptimo así obtenido sea el óptimo del problema original es necesario que todas sus soluciones admisibles satisfagan las restricciones añadidas.

Según la base de partida en la resolución del problema relajado y las características de los cortes se distinguen --- tres grupos de algoritmos: fraccionarios duales, enteros primales y enteros duales.

Los métodos de enumeración están basados en la propiedad de que el conjunto de soluciones admisibles es denumerable y normalmente finito o reducible a un conjunto finito. Naturalmente es factible examinar todas y cada una de las soluciones admisi--- bles y si la exploración es inteligente,evitar analizar muchas de estas. Dentro de los métodos de enumeración se distinguen dos tipos de algoritmos : exploración dirigida y enumeración implicita.

La programación dinámica se emplea en la resolución - de problemas lineales enteros de ciertas características muy específicas, como el problema Knapsack. Es un método inadecuado para problemas generales debido a las necesidades de memoria requerida.

En los métodos de transformación se incluyen un con-- junto de algoritmos que en esencia resuelven un programa derivado

del planteado, más fácil y cuyo óptimo sea el del original o al - menos próximo a él. En este método incluimos la transformación en un problema Knapsack, la transformación en un problema cuadrático sin restricciones, la transformación en un problema de minimiza-- ción en un grupo abeliano y la relajación lagrangiana o empleo de los múltiplicadores de Everett. Cada uno de los problemas trans-- formados es a su vez resuelto por alguno de los métodos ya des--- critos o por otros especificos para el tipo concreto de problema considerado.

1.3. El Problema Knapsack

El problema knapsack es un problema lineal entero de - una sola restricción con todos los coeficientes positivos. A pesar de su simplicidad existen bastantes sistemas que se modelan como un problema knapsack (selección de inversiones, selección - de proyectos, máquinas de cambio de moneda, corte de papel o madera,...), apareciendo también como subrutina en la resolución - de otros problemas de programación matemática. En la literatura se le denomina a veces loading, cargo loading o flyway kit pro-- blem.

En la resolución de problemas knapsack se emplean métodos aproximados además de los descritos anteriormente, salvo los de cortes.

Los métodos aproximados obtienen una solución no necesariamente óptima, aunque próxima a él. Uno de ellos calcula la solución rápida (greedy solution) y otro utiliza la relajación - lagrangiana (multiplicadores de Everett). Describiremos bajo que condiciones ambos métodos aproximados obtienen el óptimo y, caso que estas no se cumplan, acotaciones sobre el error entre el valor obtenido y el óptimo.

Dentro de los métodos de enumeración se exponen las -- aportaciones de Gilmore y Gomory, Cabot e Ingargional y Korsh.

La de Gilmore y Gomory |48| emplea una enumeración lexicográficamente descendente partiendo de la solución admisible lexicográficamente máxima.

Cabot |19| propone la obtención de una secuencia de cotas admisibles de la función objetivo, hasta llegar a un cierto

valor no admisible. Esta enumeración se realiza calculando cotas superiores e inferiores para las variables aplicando la técnica de eliminación de Fourier-Motzquin.

Ingargiola y Korsh |79| realizan una reducción del tamaño del problema al que le aplican posteriormente un algoritmo de exploración parcial (branch-search).

La programación dinámica se emplea eficientemente en la resolución del problema knapsack. Se consideran recursiones propuestas por Bellman, Dantzig, Gilmore y Gomory, y Greenberg.

Como líneas de transformación para la resolución de un problema knapsack se exponen la transformación en el knapsack -- normalizado (Value-Independent knapsack problem) y la transformación en un problema de ruta mínima.

El normalizado es un problema knapsack en donde la expresión de la restricción es idéntica a la función objetivo. Puede abordarse su resolución calculando, mediante programación dinámica, el número de soluciones admisibles de una ecuación dio-- fántica.

Si el problema knapsack se transforma en un problema - de ruta mínima, se le puede aplicar cualquiera de los algoritmos existentes que calculan la ruta mínima a través de un grafo.

1.4. Aportaciones a la Resolución del Problema Knapsack

Se proponen nuevos algoritmos de enumeración para el - problema general y una transformación para aquellos que reunan - ciertas características que se especifican.

Mediante una cierta ordenación de las variables se pro pone una enumeración de todas las soluciones admisibles. A par-- tir de la solución lexicográficamente máxima se enumeran solucio nes lexicográficamente descendentes. De esta manera se garanti-- za el análisis de todas las posibles soluciones al llegar al -- vector nulo.

Como es sabido la efectividad del método de enumera--- ción implicita depende del criterio de eliminación de soluciones empleado. Si este es débil será necesario enumerar gran cantidad de soluciones con la consiguiente pérdida de eficiencia.

El primer criterio se basa en fijar sucesivamente el - valor de la función objetivo desestimando aquellas soluciones -- que no la mejoren. El método propuesto emplea esta idea construyendo una secuencia de problemas knapsack de forma que cada uno de ellos es una relajación del anterior y cuya relación ha sido caracterizada. A partir de una cota inicial admisible y otra no admisible del problema original, se reduce monotonamente el in-- tervalo de acotación del óptimo hasta hacerlo nulo. En ese momento se ha encontrado la solución buscada.

El segundo algoritmo de enumeración es una modifica--- ción y mejora del expuesto por Cabot. Debido a que en el método de Cabot la aproximación al óptimo se efectua mediante una sucesión de cotas admisibles, es necesario calcular una solución admisible para cada valor de la función objetivo. Estas soluciones se obtienen por enumeración de todas las variables. Realizando - la aproximación al óptimo desde cotas no admisibles será posible reducir el número de variables exploradas (excepto en el óptimo) ya que aparecerá la inadmisibilidad antes de realizar la enumeración de todas ellas.

Independientemente de las cotas utilizadas, el número de ellas a considerar será elevado si el óptimo está lejos del - valor inicial, con lo que el algoritmo será ineficiente. Para --

evitar este inconveniente, se propone trabajar simultaneamente - con una cota admisible y otra no admisible, reduciendo a la mi-- tal en cada iteración el intervalo entre ambas. Los dos métodos se basan en el concepto de dominancia.

1.5. Resolución de Problemas Lineales Enteros por Enumeración - Implicita

Como ya se ha señalado antes, uno de los métodos de re solución de problemas lineales enteros es la enumeración implici ta. Esta forma de enumeración ha sido fundamentalmente aplicada a problemas binarios o de especiales características. En este -- trabajo se emplea este punto de vista para desarrollar algorit-- mos que resuelven cualquier problema lineal entero acotado.

En el capítulo V se estudia la resolución de los pro-- blemas lineales enteros en que todos los valores que intervienen son positivos, presentandose dos algoritmos de resolución basa-- dos en la enumeración implicita. Cada uno de ellos emplea un cri terio de eliminación de soluciones.

El primero generaliza resultados de Gilmore y Gomory - para el problema knapsack. Básicamente estos consisten en observar que una vez fijadas un conjunto de variables a ciertos valores y resuelto el problema resultante sin la restricción de integridad, si el óptimo así obtenido es peor que una solución entera ya conocida, cualquier solución admisible con las variables - fijadas está dominada por dicha solución. Así se habrá eliminado un conjunto de soluciones: aquel formado por las variables fijadas a los valores considerados. El segundo es una generalización para este caso del método de enumeración lexicográfica, desarrollado para el problema knapsack, al que nos hemos referido anteriormente.

En el capítulo VI se discute la transformación de cualquier problema lineal entero en positivo. Para obtener la solución lexicográficamente máxima, con la que se inicia el método, se propone un algoritmo basados en resultados de algebra lineal.

1.6. Experiencias Computacionales

Para estudiar de forma comparativa los algoritmos propuestos para el problema knapsack, se ha resuelto un conjunto --

elevado de problemas test. Este conjunto se ha dividido en gru-- pos variando las características de número de variables, aportaciones marginales a la función objetivo y tamaño del knapsack. - El resto de los parámetros de los problemas test se han generado aleatoriamente a partir de una distribución uniforme.

Todos los problemas han sido simultaneamente resueltos con los algoritmos aquí propuestos y los más eficientes de los - existentes.

Analizando minuciosamente los resultados de estos experimentos se deduce que el algoritmo LR78, presentado en este trabajo, es el más eficiente de todos los estudiados. Sus tiempos - de ejecución son los menores, dependiendo casi exclusivamente -- del número de variables que intervienen en el problema knapsack.

Los algoritmos desarrollados para problemas lineales - enteros acotados han sido empleados en resolver problemas generados aleatoriamente y se incluyen los resultados obtenidos. Al carecer de otros códigos de ordenador no ha sido posible realizar un estudio comparativo como el realizado para el problema knap-- sack. Debido a esto no es posible establecer para estos algoritmos conclusiones comparativas.

En el capítulo VII se detallan los pormenores del análisis, presentándose las tablas y curvas obtenidas. Se incluyen los programas y resultados de ordenador completos en los anexos.

CAPITULO II

PROGRAMACION LINEAL ENTERA

2.1. Introducción.

Un problema de programación matemática consiste en encontrar el extremo (optimizar) de una función que toma valores reales, para cualquier vector perteneciente al conjunto S, definido sobre el espacio euclideo de n dimensiones, R^n. Es decir

$$f : S \subset R^n \to R$$

siendo el problema

$$(2.1) \qquad \max\ (\min)\ f(x)$$
$$x \in S$$

Así, en todo programa matemático se distingue la función objetivo a optimizar y el conjunto de restricciones del pro-

blema que describe la región S. Dado que max f(x) es igual a -- min -f(x) se consideran problemas de maximización. En este contexto empleamos la siguiente terminología :

- Una solución admisible es cualquier vector que satisfaga las restricciones del problema.

- Una solución es óptima, si es admisible y maximiza la función objetivo.

- Existen soluciones alternativas cuando la solución óptima no es única.

- Un problema es no acotado si el valor óptimo de la función objetivo es no acotado.

- El problema p_1 es una relajación de p_2 si el conjunto de soluciones admisibles de p_1 incluye a las de p_2, siendo idénticas sus funciones objetivos. Obviamente el valor óptimo de la función objetivo de p_1 proporciona una cota superior de la de p_2. Si este valor se obtiene para un vector admisible del programa p_2, es su solución óptima.

Según las características del problema de optimiza---ción se tienen distintos tipos de programas matemáticos. Cuando - tanto la función objetivo como las restricciones son lineales, -- nos referimos a programas lineales continuos, que toman la forma

$$\begin{aligned} \max \quad & cx \\ & Ax = b \\ & x \geq 0 \end{aligned}$$

donde $c = (c_j)$, $b = (b_i)$, $A = (a_{ij})$, $x = (x_j)$ para $i = 1,\ldots,m$ y $j = 1,\ldots,n$. La condición de no negatividad de los vectores admisibles se introduce por conveniencia y no supone pérdida de generalidad. Si se le añade la restricción de que todas o algunas de las coordenadas de los vectores admisibles sean enteras se distin gue la programación lineal entera (PLE) y la mixta (PLM), respectivamente.

Nótese que si en un programa lineal entero (o mixto) se prescinde de la condición de integridad, se obtiene un programa lineal continuo que es una relajación del anterior. En general le denominaremos problema lineal entero relajado o problema li---neal asociado.

Los métodos de resolución de estos problemas, se clasifican en la literatura del tema |43, 105, 112| en métodos de -- cortes, de enumeración y minimización en un grupo. Como ninguno - de estos incluye los algoritmos de programación dinámica, relajación lagrangiana, transformación en un problema cuadrático sin -- restricciones o en un problema knapsack, los agruparemos en cua-- tro clases : metodos de cortes, de enumeración, de transformación y de programación dinámica. En esta clasificación la minimización en un grupo se incluye dentro de los métodos de transformación.

2.2. Método de Cortes.

La enorme dificultad de la resolución de un PLE res-- pecto a su problema lineal asociado se debe a la pérdida de la -- propiedad de región convexa por su conjunto de soluciones admisibles.

En un programa lineal, el conjunto de las soluciones admisibles forman una región convexa en el espacio R^N y su óptimo es un vértice de ésta (|30|, |84|). No así en un PLE, donde es un conjunto discreto dentro de la región convexa, correspondiente al PLE relajado.

Si todos los vértices de la región fuesen enteros, el óptimo también lo sería. Bastaría para obtener el óptimo del PLE

con resolver un problema lineal. A partir de esta idea se conci-- ben los métodos de los cortes.

Dado el PLE

$$\begin{aligned} &\max \quad cx \\ &\quad A_1 x = b_1 \\ &\quad x \geq 0, \text{ entero} \end{aligned} \tag{2.2}$$

los métodos de cortes consiste básicamente en buscar un nuevo conjunto de restricciones ($A_2 x = b_2$) para añadirselas al PLE relajado, de forma que no se excluya ninguna solución admisible del PLE original y que el óptimo del nuevo problema sea entero. Este toma la forma

$$\begin{aligned} &\max \quad cx \\ &\quad A_1 x = b_1 \\ &\quad A_2 x = b_2 \\ &\quad x \geq 0 \end{aligned}$$

En principio, para resolver un PLE bastaría con añadir el número necesario de restricciones al relajado, de manera -

que todos los nuevos vertices fuesen enteros.

Desafortunadamente, el número de vértices de la envoltura convexa de las soluciones admisibles de (2.2) puede ser un número ilimitado, incluso para simples ejemplos de dos ecuaciones con dos incognitas(|80|),o una ecuación con dos incognitas (|97|). Por esto, únicamente se introducirán cortes en las proximidades del óptimo del PLE relajado, aproximando por esa zona el poliedro convexo de este a la envoltura convexa del PLE.

Según las características del punto de partida y de los cortes introducidos se distinguen tres tipos de algoritmos de cortes: fraccionarios duales, enteros duales y enteros primales.

2.2.1. Cortes Fraccionarios Duales.

En el problema (2.2), para garantizar la convergencia del método es necesario que todos los datos de A, b y c sean racionales.

Inicialmente, se resuelve el PLE relajado. Si éste no tiene solución, el problema (2.2) tampoco la tiene. Caso contrario se obtiene, reordenando las variables, una solución de la forma

$$x_i = a_{io} - \sum_{j \in D} a_{ij} x_j \quad (i = 0,1,\ldots,m)$$

donde, para $i = 1,\ldots,m$, x_i son las variables básicas, a_{io} los valores que estan toman en el óptimo, a_{oo} el valor de la función objetivo y a_{oj}, $(j = 1,\ldots,n)$, son los costos relativos de las variables. D es el conjunto de los índices correspondientes a las variables no básicas. Si todos los términos a_{io} son enteros, la solución obtenida es óptima. En caso contrario, en alguna restricción k del tipo,

$$x_k = a_{ko} - \sum_{j \in D} a_{kj} x_j$$

a_{ko} es fraccionario. Se construye un corte de la forma

$$h = -t + \sum_{j \in D} d_j x_j \qquad h \geq 0 \text{ entero}$$

$$|t| \geq 0 \qquad (2.3)$$

donde t, d_j dependen de a_{ko}, a_{kj}. El corte (2.3) hace que la solución básica deje de ser admisible para el problema primal debido a que la nueva variable básica h tiene valor negativo -t. La admisibilidad para el problema dual permanece ($a_{oj} \geq 0$ $j = 1,..,n-m$). Se busca el óptimo de este problema lineal resultante, mediante el algoritmo simplex-dual. Si el óptimo es entero o el problema no tiene solución se ha encontrado la solución óptima del problema entero o esta no existe, respectivamente. En caso de que la solución sea fraccionaria es necesario repetir el proceso, introduciendo nuevos cortes.

Representamos por $[a_{ij}]$ y $\langle a_{ij} \rangle$ los enteros inmediatamente inferior y superior a a_{ij}, respectivamente.

El corte general más utilizado fue propuesto por Gomory |58| y tiene, con esta notación, la expresión :

$$(2.4) \qquad h = -[\lambda]\, a_{io} + [\lambda\, a_{io}] + \sum_{j \in D} ([\lambda] a_{ij} - [\lambda\, a_{ij}])x_j$$

Empleando valores distintos del parámetro λ se obtienen diferentes cortes. El número de estos, para $\lambda = 1,2,...$ es finito y a partir de un cierto valor del parámetro se repiten ciclicamente. En concreto, para $\lambda = 1$ y siendo $f_{kj} = a_{kj} - [a_{kj}]$, Gomory |56| propone el corte

$$h = -f_{ko} + \sum_{j \in D} f_{kj}\, x_j$$

Desde distintos puntos de vista, se han propuesto -- otros cortes. Así, Glover |52| considera

$$h = -(\langle \lambda\, a_{ko} \rangle - \lambda\, a_{ko}) + (\langle \lambda \rangle - \lambda)\, x_k +$$

$$+ \sum_{j \in D} (\langle \lambda\, a_{kj} \rangle - \lambda\, a_{kj})\, x_j$$

utilizado también para generar cortes enteros duales.

Además de los cortes fraccionarios duales expuestos - hasta ahora, existen otros de este tipo que se caracterizan por-- que los coeficientes de las variables que intervienen en él son - binarios. Estos cortes, llamados unitarios (|43|), aparecen con - el propuesto por Dantizg |28|

$$h = -1 + \sum_{j \in D} x_j$$

Gomory y Hoffman |60| mostraron que solamente bajo determinadas condiciones converge un algoritmo con este corte. Bowman y Nemhauser |13| lo modifican a

$$h = -1 + \sum_{j \in D_k} x_j$$

siendo $D_k = \{j \mid j \in D,\ a_{kj} \text{ no entero}\}$

garantizando la convergencia.

Otras propuestas con coeficientes binarios son la de Charnes y Cooper |22|

$$h = -1 + \sum_{j \in D} \delta_j x_j$$

donde $\delta_j = 0$ si $a_{kj} = 0$ y $\delta_j = 1$ si $a_{kj} \neq 0$; y la de Ben-Israel y Charnes |12|

$$h = -1 + \sum_{j \in D} \delta_j x_j$$

donde $\delta_j = \langle a_{kj} \rangle - [a_{kj}]$. Expresado de otra forma $\delta_j = 0$ si a_{kj} es entero y $\delta_j = 1$ si a_{kj} no es entero.

Intuitivamente, un corte es más fuerte que otro si la región que excluye es mayor. Es decir, el corte

$$\sum_{j \in D} p_j x_j \geq t$$

es más fuerte (o incluye) al corte

$$\sum_{j \in D} \overline{p}_j x_j \geq \overline{t}$$

si se cumple, para cada j, que $p_j \leq \overline{p}_j$ y $t \geq \overline{t}$ con alguna de las desigualdades estricta.

Para obtener cortes aun más profundos Bowman y Nemhauser |14|, sugieren que, dado uno cualquiera

$$\sum_{j \in D} d_j x_j \geq t$$

es posible obtener otro

$$\sum_{j \in D} d_j x_j \geq t^*$$

donde $t^* > t$, y por tanto más profundo. La obtención de t^* se realiza calculando

$$t^* = \min \sum_{j \in D} d_j x_j$$

$$x \in Q.$$

siendo Q el conjunto de soluciones admisibles del problema. El interés práctico de este resultado se ve reducido por el esfuerzo computacional necesario para calcular t^*.

2.2.2. Cortes Enteros Duales.

A partir de una tabla dual admisible del problema (2.2) relajado, en la que todos los elementos que en ella intervienen son numeros enteros, cada fila i corresponde a una ecuación del tipo

$$x_i = a_{io} - \sum_{j \in D} a_{ij} x_j \qquad (2.5)$$

Si todos los a_{io} son no negativos la solución es óptima. Caso contrario, en alguna restricción k de (2.5), $a_{ko} < 0$

$$x_k = a_{ko} - \sum_{j \in D} a_{kj} x_j$$

A partir de esta restricción se general un corte

$$(2.6) \qquad h = -t + \sum_{j \in D} d_j x_j \qquad |t| > 0$$

$$h \geq 0 \text{ entero}$$

con todos los coeficientes enteros (t entero, d_j ($j \in D$) entero). Este corte es dual admisible y primal no admisible.

El corte (2.6) ha de ser de tal forma que sea -1 el elemento sobre el que se pivotea al realizar una iteración del algoritmo simplex-dual.

Debido a que la tabla es toda entera y el elemento sobre el que pivotea es -1, la iteración se reduce a suma y resta de filas. La tabla permanecerá toda entera. Si después de esta --

iteración la tabla es primal admisible, se ha llegado al óptimo. Si no es así, es necesario repetir el proceso introduciendo un -- nuevo corte.

Gomory |57| propone el corte (2.4) particularizado para $0 < \lambda < 1$. Así $[\lambda] = 0$ tomando la forma

$$h = [\lambda\, a_{ko}] - \sum_{j \in D} [\lambda\, a_{kj}] x_j$$

Se escoge λ de manera que $[\lambda\, a_{kr}] = -1$ y $(a_{jo} + \; - \; + [\lambda\, a_{kj}]\, a_{or}) \geq 0$, siendo $[\lambda\, a_{kr}]$ el pivote. La columna r sobre la que se pivotea se elegirá de tal forma que

$$a_{or} = \min_{j \in I^-} a_{oj}$$

siendo $I^- = \{j \mid a_{kj} < 0,\ j \in D\}$.

Glover |51| emplea el corte

$$h = - < \frac{a_{ko}}{a_{kr}} > + x_r$$

supuesto todos los elementos de la fila k son no negativos salvo el término independiente y el pivote.

2.2.3. Cortes Enteros Primales.

A partir de una tabla primal admisible del problema - (2.2) relajado, en la que todos los elementos que intervienen son enteros si los costes relativos a_{oj} son no negativos para todo j la solución es óptima. Si algún a_{or} es negativo, se busca la fila k

$$x_k = a_{ko} + \sum_{j \in D} a_{kj} x_j$$

de forma que satisfaga

$$\frac{a_{ko}}{a_{kr}} = \min \left\{ \frac{a_{io}}{a_{ir}} \,\middle|\, a_{ir} > 0,\ 1 \leq i \leq m \right\}$$

Cuando a_{kr} es la unidad, se pivotea sobre él. Si no es así, se genera el corte

$$h = t + \sum_{j \in D-\{r\}} d_j x_j + x_r \qquad t \geq 0 \text{ entero}$$

$$h \geq 0 \text{ entero}$$

de tal forma que

$$\frac{t}{1} \leq \frac{a_{ko}}{a_{kr}} \leq \frac{a_{io}}{a_{ir}} \quad \text{para todo } i \ (1 \leq i \leq m)$$

tras lo que se itera. En ambos casos con el algoritmo simplex, dado que las tablas son siempre admisibles para el primal.

Ben-Israel y Charnes [12] proponen el corte (2.4) particularizado para $\lambda = 1/a_{kr}$, tomando la forma

$$(2.7) \qquad h = \left[\frac{a_{ko}}{a_{kr}} \right] + \sum_{j \in D-\{r\}} \left[\frac{a_{kj}}{a_{kr}} \right] + x_r$$

La condición de que λ sea igual a $1/a_{kr}$ en el corte - de Gomory, es suficiente para que el elemento pivote sea 1. Sin - embargo no es necesario, siendo posible obtener cortes más fuer-- tes con otros valores de λ. La condición necesaria es que el pi-- vote esté en el elemento de valor 1. Esto es,

$$a) \quad \frac{[\lambda\ a_{ko}]}{1} \leq \frac{a_{ko}}{a_{kr}}$$

$$b) \quad 1/a_{kr} \leq \lambda < 2/a_{kr}$$

Como la condición a) es equivalente a

$$\lambda < \frac{< \frac{a_{ko}}{a_{kr}} >}{a_{ko}}$$

pueden ambas resumirse en

$$\frac{1}{a_{kr}} \leq \lambda < \min \{ \frac{2}{a_{kr}} , \frac{< \frac{a_{ko}}{a_{kr}} >}{a_{ko}} \}$$

El algoritmo propuesto por Ben-Israel y Charnes era el primero del tipo primal, con la ventaja respecto a los algoritmos duales, de trabajar siempre con cotas admisibles del problema. Este primer algoritmo es conocido como RPA (Rudimentary primal -- algorithm).

Si el corte nunca es degenerado, es decir $a_{ko} \geq a_{kr}$, la secuencia de valores $\{a_{oo}\}$ es estrictamente creciente y el algoritmo converge. Caso contrario, Mathis |91| expone dos contra-- ejemplos del RPA. En el primero se presenta un caso de ciclos. En el segundo la secuencia de cortes no converge.

Realizando una ordenación lexicográfica de las columnas de la tabla, y utilizando el corte (2.7), Young |122| y Glo-- ver |54| proponen unas reglas de selección de la columna y la fila sobre las que se pivotea, y muestran que el algoritmo resultante, llamado SPA (Simplified primal algorithm), converge en un número finito de iteraciones.

2.3. Método de Enumeración.

En los casos en que el espacio de soluciones de un -- programa entero, lineal o no, es acotado, el conjunto de soluciones admisibles es finito.

Sea el programa entero

$$\begin{aligned} \min \quad & z = f(x) \\ & x \in H \\ & x \text{ entero} \end{aligned}$$

Si el conjunto H es acotado pueden establecerse s_j tales que $x_j \leq s_j$, $(j = 1,\ldots,n)$. El número de soluciones del problema, s, tiene una cota superior

$$s \leq \prod_{j=1}^{n} s_j$$

En principio, evaluando la función objetivo para cada una de las soluciones admisibles, se resuelve el problema. Asociada a la evaluación de menor valor estará la solución óptima del problema. Sin embargo, el número de soluciones s es en general tan grande que no es práctico enumerar explícitamente todas las soluciones admisibles. Por ello es necesario algún tipo de criterio para limitar la enumeración.

Siguiendo a Hu |75| se pueden señalar como características comunes en los algoritmos de enumeración las siguientes :

a) Son fáciles de entender.

b) Son fáciles de programar.

c) El número de pasos a realizar en la resolución de un problema tiene una cota superior.

d) No tienen estructura matemática, si bien existen intentos recientes de establecerla.

Según la forma de eliminación de exploraciones, pueden distinguirse dos tipos bien diferenciados (|23|) : algoritmos de exploración dirigida y de enumeración implicita.

Para un problema general de programación matemática,

$$\min \; f(x) \qquad x \in Q \tag{2.8}$$

la exploración dirigida consta de los siguientes pasos. Primero - se busca una solución admisible del problema (p.e. $x_0 \in Q$) y se - define $L \equiv f(x_0)$. L es, por tanto, una cota superior admisible de la solución del problema.

Se realiza una partición del conjunto de soluciones - Q, en k_1 subconjuntos H_i, de forma que

$$a) \quad \bigcup_{i \in I} H_i = Q \quad , \quad y$$

$$b) \quad H_j \cap H_i = \phi \quad \text{para todo } i \neq j \ (i,j \in I)$$

siendo $I = \{1,2,\ldots,k_1\}$, si bien esta segunda característica no - es esencial.

Se establecen cotas inferiores u_i, $i \in I$ que cumplan

$$u_i \leq \min \{f(x) \mid x \in H_i\} \quad \text{para todo } i \in I$$

Si cualquiera de estas expresiones se satisface con - signo de igualdad ($u_i = f(x^*)$, $x^* \in H_i$), y $L > u_i$, entonces -- $L = u_i$, debido a que se ha encontrado una solución admisible me-- jor de la que se tenía.

Sean

$$I_1 = \{j \mid u_j > L,\ j = 1,\ldots,k_1\}$$

$$I_2 = \{ j \mid u_j \leq L,\ j = 1,\ldots k_1\}$$

Debido a que L es un valor admisible, la solución óptima de (2.8) no se encuentra en H_i, $i \in I_1$. Pueden por tanto --- eliminarse del problema los conjuntos H_i, $i \in I_1$.

El óptimo se encuentra en $\bigcup_{i \in I_2} H_i$. Cada uno de los -- conjuntos H_i, $i \in I_2$ se volverá a dividir en k_2 subconjuntos de - idéntica forma que anteriormente. El proceso continua hasta que - quede un conjunto en el que su cota inferior coincida con la cota superior factible.

El método consta de :

a) Rutina de partición del conjunto de soluciones admisibles en subconjuntos.

b) Rutina de evaluación de cotas inferiores.

En la figura 2.1 se resume el proceso de analisis de cada subconjunto.

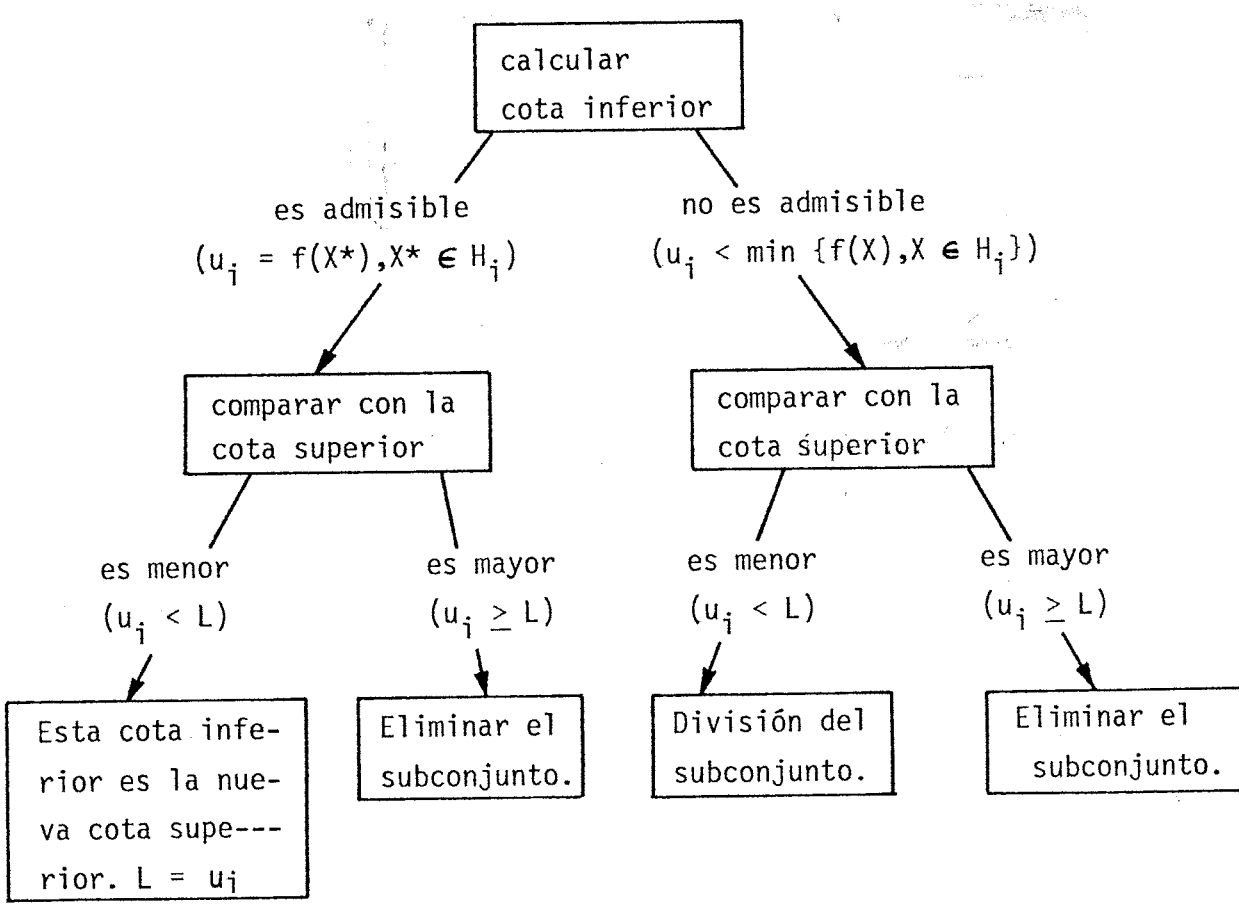


Figura 2.1

El campo de aplicación de esta técnica es muy amplia. Lawler y Wood |86| presentan una exposición general con aplica--- ción a problemas de PLE, programación no lineal, asignación cua-- drática, del viajante, asignación de recursos, secuenciación, -- etc.

El primer algoritmo aplicando el método exploración - dirigida a la programación lineal entera fue presentado por Land y Doig |83|. Puede utilizarse tanto para problemas enteros como - mixtos.

Para su exposición definimos únicamente la inicialización y los criterios de partición y acotación. El desarrollo del algoritmo es idéntico al proceso general desarrollado en el apartado anterior.

1) Inicialización.

Considerese como nodo cero la solución óptima

$$x^* = (x_1^*, x_2^*, \ldots, x_n^*)$$

del problema relajado.

2) Rutina de partición.

Supongase que se desea explorar el nodo activo i del arbol de decisión. Se considera que un nodo es activo si su cota inferior es menor que la cota superior admisible. Para que el nodo i sea activo es necesario que alguna variable no sea entera.

Existen dos tipos de nodos activos, los que no tienen sucesor y los que sí los tienen.

a) Si a partir de ese nodo no se ha realizado ninguna partición, se selecciona aquella variable $x_k = x_k^*$ no entera y tal que su parte fraccionaria sea la mayor. Se divide el conjunto de soluciones Q en dos conjuntos $H_1 = \{x \mid x \in Q,\ x_k = [x_k^*]\}$ --- $H_2 = \{x \mid x \in Q,\ x_k = < x_k^* > = [x_k^*] + 1\}$.

b) Si a partir del nodo ya se ha realizado alguna partición haciendo la variable $x_k = x_k^i, \ldots, x_k^s$ añadir una nueva bifurcación, fijando $x_k = x_k^i - 1$ en el caso de que la cota inferior obtenida haciendo $x_k = x_k^i$ es menor que la obtenida haciendo $x_k = x_k^s$. Caso contrario se hará $x_k = x_k^s + 1$. Si tanto la cota obtenida haciendo $x_k = x_k^i$ como $x_k = x_k^s$ son mayores que la cota superior el nodo deja de ser activo.

3) Rutina de evaluación de cotas inferiores.

La cota inferior en cada nodo será el valor óptimo -- del PLE relajado. Si en un nodo el problema relajado no tiene solución la cota inferior se hará igual a infinito.

El algoritmo de Land-Doig con los anteriores crite--- rios tiene algunos inconvenientes :

a) Debido al criterio de partición es posible que a - partir de un nodo se originen gran número de bifurcaciones. Para evitar esto Dakin |25| propone la siguiente rutina de partición:

Siendo Q el conjunto de soluciones, y supuesto que - x_k^* es fraccionario, particionar Q en dos conjuntos :

$$H_1 = \{x \mid x \in Q,\ x_k \leq [x_k^*]\} \quad , \quad H_2 = \{x \mid x \in Q,\ x_k \geq [x_k^*] + 1\}$$

Con lo cual cada nodo únicamente tendrá dos sucesores inmediatos.

Para evitar calculos innecesarios en la generación de cotas Beale y Small |7|, y más tarde Tomlin |115|, suponen que la base del simplex no cambia en las bifurcaciones. Si es cierto que la base no cambia la cota es exacta. Y si lo hace la solución obtenida es admisible para el dual, luego inferior a la óptima. En ambos casos se tiene una cota inferior.

Para conseguir una buena cota superior inicial, Taha |111|, |112|, propone explorar el hipercubo unidad entero que contiene el óptimo del PLE relajado. La busqueda del mejor vértice del hipercubo (si es admisible) la realiza resolviendo un programa lineal binario.

2.4. Métodos de Transformación.

Un método común en toda la programación matemática es la transformación del problema original, en otro más sencillo, cuya resolución facilite información sobre el óptimo del problema original.

Dentro de los métodos de resolución de PLE mediante transformación del problema se pueden distinguir dos clases. En la primera, el conjunto de soluciones de ambos problemas, original y transformado, son exactamente idénticos pero de diferente estructura y las funciones objetivos son las misma. Dentro de es-

te tipo de transformación se encuentra la reducción de un PLE general a un problema knapsack. En la segunda, la relación es menos intima, coincidiendo los óptimos de ambos problemas únicamente bajo determinadas condiciones. Entre estos se encuentran la modificación del PLE a un problema de minimización sobre un grupo y la utilización de los multiplicadores de Lagrange (o de Everett).

2.4.1. Transformación en un Problema Knapsack.

Consiste básicamente en reducir un sistema de m ecuaciones diofánticas de n variables a un sistema de k ecuaciones -- ($k < m$) y n variables, de forma que el conjunto de soluciones admisibles de ambos sistemas sea idéntico. Esta reducción puede ser ventajosa.

El resultado fundamental se debe a Bradley [16], que mostró que cualquier sistema de ecuaciones diofánticas, lineales o no, con un número finito de soluciones (región acotada) es reducible a una sola ecuación.

Para el sistema de ecuaciones lineales,

$$\sum_{j=1}^{n} a_{1j}\, x_j = b_1$$

$$\sum_{j=1}^{n} a_{2j}\, x_j = b_2$$

$$u_j \geq x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

donde los valores a_{ij} son enteros y libres en el signo, la ecuación resultante es

$$\sum_{j=1}^{n} (a_{1j} + k\, a_{2j}) x_j = b_1 + k\, b_2$$

$$u_j \geq x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

siendo

$$|k| > \max \{ \left| \sum_{j=1}^{n} a_{1j}\, x_j - b_1 \right|,\ u_j \geq x_j \geq 0 \text{ entero} \}$$

Desafortunadamente, aun para muy pocas ecuaciones, los coeficientes de la ecuación agregada toman valores tan grandes que hacen poco práctico la utilización de estas transformaciones en la resolución de un PLE.

Para conseguir menores valores de k basta con elegir esta de forma que (|112|) :

$$k > \max_j \{(b_2 - 1)(a_{1j}/a_{2j}) - b_1\}$$

$$k > \max_j \{b_1 - (b_2 + 1)(a_{1j}/a_{2j})\}$$

A pesar de esta y otras transformaciones (|35|, |55|) los coeficientes siguen siendo elevados incluso para problemas pequeños.

2.4.2. Minimización en un Grupo.

Introducción.

Este método de resolución de PLE (llamado también de minimización sobre conos o algoritmo asintótico), surge al mostrar Gomory |59| que relajando la restricción de no negatividad para algunas variables, el problema se transforma en otro de resolución más fácil. Este nuevo problema, de minimización en un grupo (GMP), resuelve el problema original si la solución obtenida es admisible para él.

A todo PLE, con términos independientes suficientemente grandes, le corresponde un GMP tal que sus óptimos coinciden. Esto es, dado el programa

$$\max \quad cx$$
$$Ax = \lambda\, b$$
$$x \geq 0 \text{ entero}$$

su correspondiente GMP tiene idéntico óptimo para todo λ mayor ó igual que un cierto valor. Debido a que al tender los términos independientes asintoticamente a un cierto valor el GMP resuelve el original a este método se le conoce como algoritmo asintótico.

El algoritmo asintótico puede resumirse en los siguientes pasos (|76|) :

1) Resolver el problema lineal relajando la condición de integridad para obtener la base óptima B.

2) Plantear el problema de minimización en un grupo.

3) Resolverlo obteniendo el valor de las variables no básicas.

4) Comprobar si la sustitución de las variables no básicas en la expresión de las básicas mantiene a estas positivas o cero. Si así es, la solución obtenida es admisible y por tanto óptima. Caso contrario se aplican algoritmos de exploración dirigida (|107|) para aprovechar los cálculos realizados.

En los apartados siguientes se analizan los distintos aspectos de este método de resolución. El primero es la obtención del problema en un grupo. El segundo, el análisis de las condiciones bajo las cuales este resuelve el original y por último un estudio general de los métodos de resolución del problema de minimización en un grupo.

Obtención del problema en un grupo.-

Para el problema lineal

$$\max \quad cx$$
$$Ax = b$$
$$x \geq 0 \text{ entero}$$

con todos los elementos enteros, considerese una base B del pro--

blema lineal asociado. Particionando los vectores y matrices del problema según B, tenemos

$$\begin{aligned} \max \quad & c_B\, x_B + c_D\, x_D \\ & B\, x_B + D\, x_D = b \\ & x_B \geq 0 \text{ entero} \\ & x_D \geq 0 \text{ entero} \end{aligned}$$

donde $c = (c_B, c_D)$, $A = (B,D)$, $x = (x_B, x_D)^T$.

Expresando las variables básicas (x_B) en función de las no básicas, el problema se transforma en

$$\begin{aligned} \max \quad & c_B\, B^{-1}b - (c_B\, B^{-1}D - c_D)x_D \\ & x_B = B^{-1}(b - D\, x_D) \\ & x_B \geq 0 \text{ entero} \\ & x_D \geq 0 \text{ entero} \end{aligned}$$

Si se prescinde de la restricción de no negatividad -- de las variables básicas ($x_B \geq 0$) y teniendo en cuenta que el término $c_B B^{-1} b$ es una constante, el problema así relajado se convierte en

$$\max \; -(c_B \, B^{-1} D - c_D) \, x_D$$

$$B^{-1}(b - D \, x_D) \equiv 0 \pmod 1$$

$$x_D \geq 0 \text{ entero}$$

o lo que es equivalente

$$\min \; (c_B \, B^{-1} D - c_D) \, x_D$$

$$B^{-1} D \, x_D \equiv B^{-1} b \pmod 1$$

$$x_D \geq 0 \text{ entero}$$

Llamando $\bar{c} = (\bar{c}_1, \ldots, \bar{c}_{n-m})$ a los costos relativos de las variables no básicas ($\bar{c} = c_B \, B^{-1} D - c_D$), α_j a la columna j de

$B^{-1}D$ y α_o al vector $B^{-1}b$ se tiene el problema

$$\min \quad \sum_{j=1}^{n-m} \bar{c}_j x_j$$

$$\sum_{j=1}^{n-m} \alpha_j x_j \equiv \alpha_o \pmod 1$$

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n-m)$$

y dado que para cualquier número f, se cumple la relación de congruencia

$$f \equiv f + m \pmod k$$

si m es divisible por k. Representanto por $\bar{\alpha}_j$ el vector formado por las partes fraccionarias de α_j, para todo j

$$\alpha_j \equiv \bar{\alpha}_j \pmod 1$$

con este resulta el problema de minimización en un grupo

$$\min \sum_{j=1}^{n-m} \bar{c}_j x_j$$

$$\sum_{j=1}^{n-m} \bar{\alpha}_j x_j \equiv \bar{\alpha}_o \pmod 1 \tag{2.9}$$

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n-m)$$

Este problema puede interpretarse como una minimización de la variación de la función objetivo original debido a un aumento positivo de las variables no básicas, para así conseguir que las variables básicas, que eran fraccionarias, se conviertan en enteras.

Todas las componentes de los vectores $\bar{\alpha}_j$ son de la forma I_{ij}/M donde I_{ij} es un entero positivo menor que M siendo M el determinante de la base B($M = |B|$). Se demuestra fácilmente que dichos vectores tienen una estructura de grupo abeliano $G(\bar{\alpha})$ con la operación de congruencia modulo 1.

El problema (2.9) se conoce como de minimización sobre un grupo abeliano.

Debido a la estructura del grupo, si el problema tiene alguna solución admisible basta con que algún $\bar{c}_j$ sea negativo

para que la solución óptima sea no acotada (al menos en la variable x_j). Por lo cual para que la solución del problema de minimización tenga interés es necesario que todos los $\bar{c}_j$, es decir todos los costos relativos de las variables no básicas, sean positivos. Por ello se emplea como base B la correspondiente a la solución óptima del problema lineal continuo asociado.

Condición suficiente de optimalidad del PLE.

Dada una base B de un PLE se define como cono K_B generado por las columnas de B al conjunto de puntos que puedan obtenerse por sus combinaciones lineales no negativas. El cono $K_B(d)$ es el obtenido eliminando del cono K_B todos los puntos a una distancia menor que d de sus limites. En dos dimensiones se pueden representar ambos conos K_B y $K_B(d)$ según la figura 2.2 supuesto que a_1 y a_2 son respectivamente la 1^a y 2^a columnas de B.

Obviamente $K_B = K_B(0)$

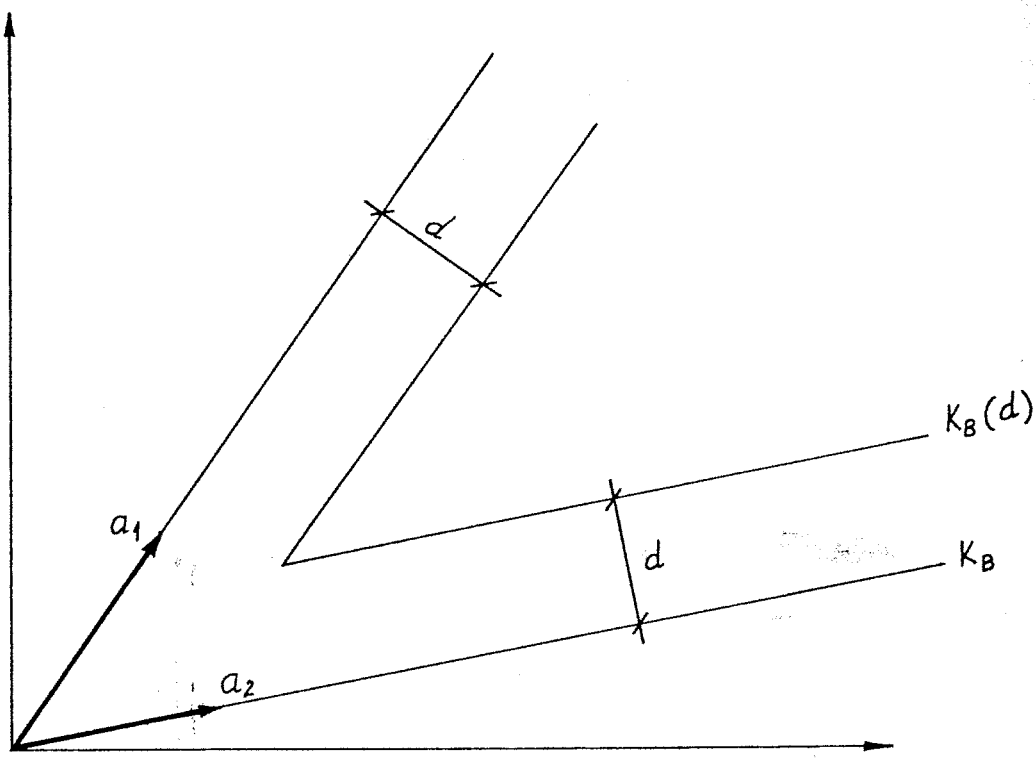


Fig. 2.2

En cualquier programa lineal si $Bx_B = b$ con $x_B \geq 0$, b pertenece al cono K_B. De idéntica forma, para cualquier vector u del cono existirá un $x_B \geq 0$ que satisfaga $Bx_B = u$, debido a que B es regular. Por tanto en programación lineal continua, elegida --

una base cualquiera B, la condición de no negatividad de las va-- riables básicas (las no básicas son cero) es que el vector b de - términos independientes pertenezca al cono K_B.

En programación lineal entera la expresión de las variables básicas es

$$Bx_B = b - Dx_D$$

con las variables no básicas x_D distintas de cero. El vector Dx_D^*, perteneciente al espacio R^m, modifica el vector original b y lo - transforma en $\overline{b} = b - Dx_D^*$. Si este nuevo vector $\overline{b}$ pertenece al cono K_B el valor de las variables básicas x_B será no negativo --- ($Bx_B = \overline{b}$). Por tanto la condición necesaria y suficiente de que - $x_B \geq 0$ es que $\overline{b} = b - Dx_D^*$ esté en el cono.

Para que el vector $\overline{b}$ pertenezca a K_B es condición suficiente, pero no necesaria, que la longitud del vector Dx_D^* sea - menor o igual a 1 y b pertenezca al cono $K_B(1)$. Es condición suficiente porque en el caso más desfavorable, que es cuando el vec--

tor Dx_D^* es perpendicular a alguno de los vectores columnas de B, si la longitud de Dx_D^* es menor que 1 como b pertenece a $K_B(1)$ resulta que $\overline{b} = b - Dx_D^*$ sigue perteneciendo a K_B. No es condición necesaria puesto que Dx_D^* puede ser mayor que 1 y sin embargo debido a su dirección, no salir $\overline{b}$ fuera del cono K_B.

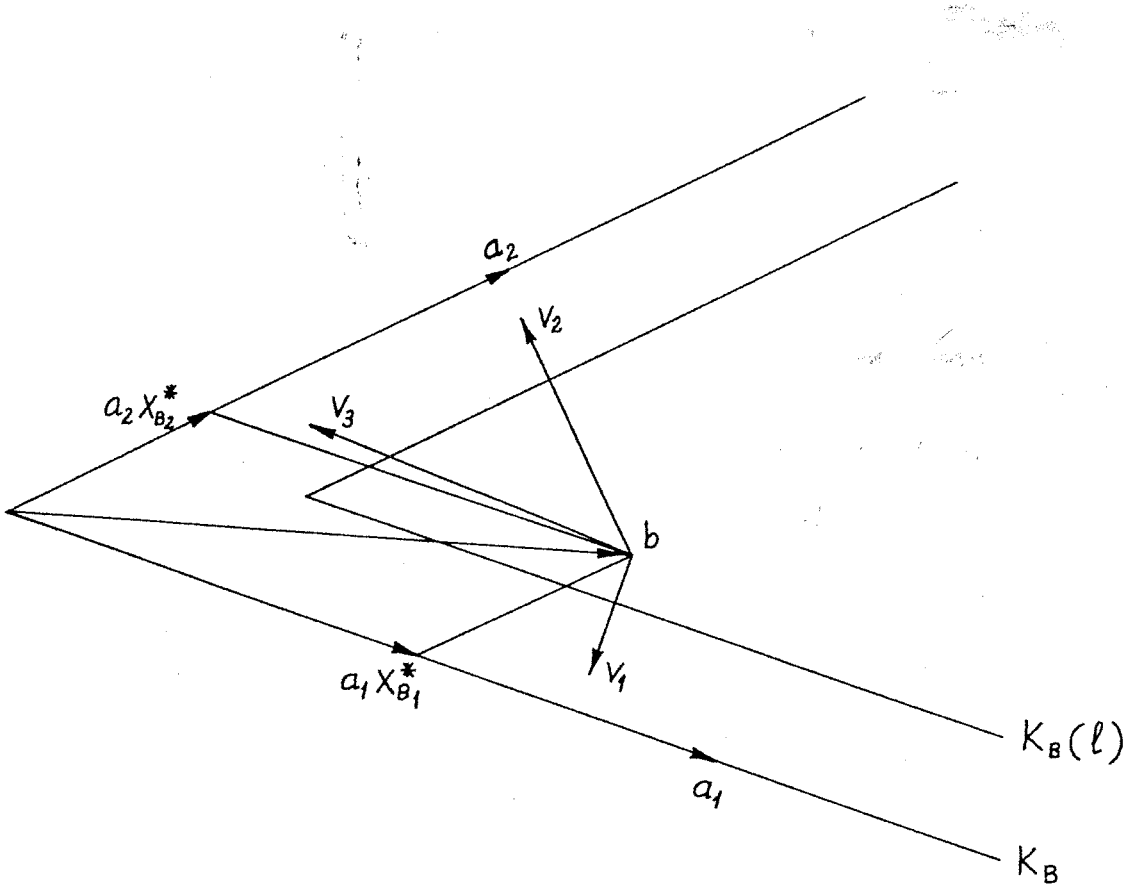


Figura 2.3

En la figura 2.3 se representan distintas posibilidades. Si la dirección de Dx_D^* es V_1 ó V_2 el vector b ha de pertenecer a $K_B(1)$ supuesto que $||Dx_D^*|| = 1$. Sin embargo, si la dirección de Dx_D^* es cualquier otro vector entre V_1 y V_2, por ejemplo - V_3, no es necesario que b pertenezca a $K_B(1)$ para que $\overline{b}$ pertenezca a K_B.

El valor de 1 puede acotarse ($|59|$, $|43|$, $|75|$) en -- función del determinante de la base B y de la norma de los vectores que la constituyen. Las cotas así obtenidas suelen ser débiles.

Optimización del problema sobre el grupo.-

Para resolver el problema de minimización (2.9) consi deramos el grupo abeliano generado por los vectores $\overline{\alpha}_j$, cuyos -- elementos denominaremos g_j, y planteamos el problema

$$\min \sum_{j \in NB} c(g_i)\, x_j$$

$$\sum_{j \in NB} g_i\, x_j = g_h \qquad (2.10)$$

$$x_j \geq 0 \text{ entero} \quad j \in NB$$

Si la asociación elegida entre los elementos g_j y los vectores $\overline{\alpha}_j$ es la adecuada, ambos problemas son equivalentes.

Hu |75| propone definir un grafo orientarlo $\Gamma(V,E)$ de forma que cada elemento del grupo sea un nodo. Al arco (i,j) - (uniendo los nodos g_i y g_j) se le asocia la distancia $c(g_i)$. Sobre este grafo orientado se plantea el problema de la ruta mínima entre el nodo g_o (elemento neutro del grupo) y g_h. La solución así obtenida es la óptima del problema de minimización sobre el grupo.

El problema (2.10) también se puede resolver por programación dinámica (|65|). Definiendo

$$f_k(y) \equiv \min \sum_{j=1}^{k} c(g_j)\, x_j$$

$$\sum_{j=1}^{k} g_j\, x_j = y$$

$$x_j \geq 0 \text{ entero} \quad (j = 1,\ldots,n)$$

si el elemento g_k es usado al menos una vez ($x_k \geq 0$) en $f_k(y)$ se tendrá que

$$f_k(y) = c(g_k) + f_k(y - g_k)$$

y si no

$$f_k(y) = f_{k-1}(y).$$

Por tanto se plantea la relación

$$f_k(y) = \begin{cases} f_{k-1}(y) & \text{si } x_k = 0 \\ c(g_k) + f_k(y-g_k) & \text{si } x_k > 0 \end{cases}$$

donde $f_0(y) = \infty$ para todo y, y $f_i(0) = 0$ para cada i. Así las -- ecuaciones de optimalidad son

$$f_k(y) = \min \{f_{k-1}(y),\ c(g_k) + f_k(y-g_k)\}$$

obteniéndose el óptimo del problema de minimización en el grupo - para $f_n(g_h)$.

Si al óptimo del problema de minimización le corres-- ponde una solución inadmisible del problema original,será necesario seguir examinando soluciones que van alejando consecutivamente la función objetivo del problema en el grupo,de su óptimo.

Para calcular la enesima mejor solución del GMP que - es la primera que hace las variables básicas positivas, Shapiro |107| con posteriores modificaciones (|61|, |62|), propone un algoritmo de enumeración.

2.4.3. Relajación Lagrangiana.

La utilización de los multiplicadores de Lagrange fue extendida por Everett |38| a problemas de optimización con funciones no diferenciables. Everett prueba el siguiente teorema :

Sea el problema de optimización

$$\begin{aligned} &\max \quad H(\ddot{x}) \\ &\text{s.a} \quad c_i(\ddot{x}) \leq c_i \qquad (i = 1,\ldots,n) \\ &\qquad \ddot{x} \in S \end{aligned} \tag{2.11}$$

Sean λ_i para $i = 1,\ldots,n$ números reales no negativos (multiplicadores). Si $\ddot{x}^* \in S$ maximiza la función

$$H(\ddot{x}) - \sum_{i=1}^{n} \lambda_i \, c_i(\ddot{x}) \qquad \ddot{x} \in S$$

y para todo $\ddot{x} \in S$ se cumple $c_i(\ddot{x}) \leq c_i(\ddot{x}^*)$ $(i = 1,\ldots,n)$ entonces $\ddot{x}^*$ es la solución óptima del problema (2.11).

Para el problema

$$\begin{aligned} \max \quad & cx \\ & Ax \leq b \\ & x \geq 0 \text{ entero} \end{aligned}$$

se ha de encontrar un vector no negativo λ tal que el óptimo $\ddot{x}^*$ de

$$\begin{aligned} \max \quad & cx - \lambda Ax \\ & x \geq 0 \text{ entero} \end{aligned}$$

satisfaga $A\ddot{x}^* = b$ (ó $A\ddot{x}^*$ esté muy próximo e inferior a b).

El punto crítico de este método de resolución es encontrar un vector λ tal que el óptimo $\ddot{x}^*$ del problema relajado haga que $A\ddot{x}^*$ sea lo suficientemente próximo a b para que sea aceptable como solución óptima del original.

Brook y Geoffrion |18| y Geofrion |45| proponen métodos iterativos de cálculo de los multiplicadores.

2.5. Métodos Basados en la Programación Dinámica

Para completar la revisión de los métodos de resolución de problemas lineales enteros nos referimos ahora a los basados en programación dinámica, si bien su interés práctico es muy limitado. Las técnicas de optimización de la programación dinámica (|8|, |10|, |95|) se utilizan principalmente en la resolución del problema Knapsack (|47|, |49|, |50|, |64|) y, más generalmente, en la resolución de un PLE utilizando la reducción a una minimización sobre un grupo como ya se ha visto.

Sea el problema entero escrito en la forma

$$\min \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{s.a} \quad \sum_{j=1}^{n} \alpha_j x_j = \alpha_o$$

$$x_j \geq 0 \text{ entero} \quad (j = 1,\ldots,n)$$

donde α_j para j = 0,1,...,n son vectores columna de m dimensiones.

Definiendo la función

$$F(\alpha) = \min \left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} \alpha_j x_j = \alpha,\ x_j \geq 0 \text{ entero} \right\}$$

$$F(0) = 0 \qquad (2.12)$$

donde α es un vector columna admisible y variable.

Nótese que $F(\alpha_o)$ es el óptimo del problema planteado.

Greenberg |66| muestra que el funcional (2.12) conduce a la recursión

$$F(\alpha) = \min_j \{c_j + F(\alpha - \alpha_j)\}$$

$$F(0) = 0$$

siendo α un vector admisible.

Esta recurrencia es la extensión para un PLE general de la recursión utilizada por Dantzig |27| y por Bellman |8| para la resolución del problema knapsack. Excepto para pequeños problemas, no tiene excesivo interés la utilización de este algoritmo - en la resolución de problemas, debido a las elevadas necesidades de memoria de computador.

CAPITULO III

EL PROBLEMA KNAPSACK

3.1. Aplicaciones del Problema Knapsack.

El problema knapsack es el problema de PLE en su -- forma más simple, existiendo varios tipos de ellos conocidos por este nombre.

El problema knapsack general, o problema multi-item - knapsack para distinguirlo del problema 0-1 knapsack, es

$$\max \{ \sum_{j=1}^{n} c_j x_j \mid \sum_{j=1}^{n} a_j x_j \leq b \ , \ x_j \geq 0 \text{ entero} \quad (j = 1,..,n)\}$$

A veces tiene interés el caso en que la restricción - se cumple con signo de igualdad, aunque entonces la admisibilidad no está asegurada.

El interés del modelo en sí, o su resolución, tiene - dos vertientes. En primer lugar, gran cantidad de situaciones reales se modelan como problemas knapsack y en segundo, existen algoritmos de resolución en programación matemática en donde se emple a como rutina la resolución de un problema de este tipo.

Pueden citarse como situaciones modelables como pro-- blemas knapsack, el problema de selección de inversiones (|88|, - |119|, |120|), selección de proyectos, cambio de moneda (|21|). - En el problema del corte de papel, (cutting-stock problem) (|47|, |48|, |49|, |50|), para la selección de los cortes principales se emplea como rutina el problema knapsack.

También se emplea como rutina en la resolución de PLE, para los problemas de minimización de un grupo (GMP). Además, la posibilidad teórica de reducción de cualquier PLE acotado a un -- problema knapsack, implica que basta resolver este problema para - resolver el original.

3.2. Propiedades del Problema Knapsack.

A continuación se exponen dos importantes propiedades del problema knapsack.

Propiedades Periódicas.- Dado el problema knapsack genérico

$$f(y) = \max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_j x_j \leq y \qquad (3.1)$$

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

donde sin pérdida de generalidad se asume que $\rho_j \geq \rho_{j+1}$ para todo j $(j = 1,\ldots,n)$, siendo $\rho_j = c_j/a_j$, Gilmore y Gomory |50| mostraron el siguiente teorema, de gran importancia en la resolución del problema.

Teorema 3.1. Existe un valor y_o tal que para todo $y > y_o$ se cumple que $f(y) = f(y-a_1) + c_1$. En otras palabras, la función $f(y)$ para $y > y_o$ es periódica de periodo c_1.

El cálculo exacto del valor y_o a partir del cual aparece la periodicidad es muy dificil. Sin embargo pueden calcularse fácilmente cotas superiores de y_o.

Hu |75| propone con una simple demostración

$$y_o < \left[\frac{c_1}{\rho_1 - \rho_2} \right]$$

Empleando los resultados de la minimización en un grupo (GMP) en Salkin |105| puede encontrarse la cota

$$y_o < a_1 \ (1 + a_k)$$

siendo $a_k = \max_j a_j$.

Optimalidad de Subestrategias.

Shapiro y Wagner |109| muestran el siguiente teorema

Teorema 3.2. Sea $x^* = (x_1^*,\ldots,x_n^*)$ la solución óptima del problema (3.1) para $y = b$. Entonces, cualquier vector entero $x^o = (x_1^o, \ldots,x_n^o)$, tal que $0 \leq x_i^o \leq x_i^*$ para todo i $(i = 1,\ldots,n)$, es la - solución óptima del problema (3.1) para $y = k$, siendo

$$k = \sum_{j=1}^{n} a_j \ x_j^o$$

Este resultado permite el conocimiento de la solución óptima de problemas knapsack, de tamaños inferior a uno ya calculado, de una forma inmediata.

3.3. Métodos de Resolución.

Existen muchos algoritmos y de muy distintas características, para resolver el problema knapsack. Para clasificarlos se realizará una primera distinción en algoritmos aproximados y algoritmos exactos, según que se obtenga simplemente una buena solución o la solución óptima.

Dentro de los aproximados, distinguimos dos métodos de resolución : el cálculo de la solución rápida y el empleo de los multiplicadores de Everett.

Los algoritmos exactos pueden clasificarse en grandes grupos según la metodología empleada en la busqueda del óptimo. Esta clasificación sin embargo no es excluyente debido a que algunos algoritmos pertenecen simultaneamente a dos grupos. Siguiendo en parte a Salkin y Dekluyer |102|, |104| se considerarán los siguientes grupos.

1.- Programación dinámica.

2.- Enumeración.

3.- Transformación.

3.4. Métodos Aproximados.

Cuando se modelan situaciones reales los datos (costos, cantidades de recurso,...) pueden no ser exactos, existiendo tolerancias en la evaluación de las magnitudes del problema. Al mismo tiempo, los modelos contienen simplificaciones de la realidad. Todo esto hace que el óptimo de un problema de programación matemática sea probablemente una buena solución del sistema, pero no necesariamente la mejor. Cualquier buena solución del modelo será indudablemente una buena solución del problema real.

Por otra parte a veces no se dispone, para la resolución de un modelo, de ningún tipo de computador y es necesario resolverlo sin él. Por tanto son necesarios algoritmos rápidos y simples que den una buena solución, aunque no sea óptima.

Se analizarán a continuación las dos líneas existentes de resolución aproximada del problema knapsack. Por una parte, la solución rápida (greedy solution), y por otra, la utilización de la teoría de los multiplicadores generalizados de Lagrange o multiplicadores de Everett.

3.4.1. Solución Rápida (Greedy Solution).

Una solución admisible para el problema knapsack de - cálculo muy simple y, en general, muy próxima al óptimo, es la solución rápida.

Dado un problema knapsack

$$\max \sum_{j=1}^{n} c_j x_j$$

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

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

supuesto, sin pérdida de generalidad, que las variables están ordenadas de forma que $c_j/a_j \geq c_{j+1}/a_{j+1}$ para todo j $(j = 1,..,n-1)$ y que el óptimo es acotado ($a_j > 0$ para todo j), a la solución admisible $x^G = (x_1^G,\ldots,x_n^G)$, donde

$$x_1^G = \left[\frac{b}{a_1} \right]$$

$$x_i^G = [(b - \sum_{j=1}^{i-1} a_j \, x_j^G)/a_i]$$

para todo i (i = 2,...,n), se le llama solución rápida.

Bajo determinadas condiciones la solución rápida es - la solución óptima. A continuación se analizarán estas condicio-- nes (|89|).

Si no se cumplen las condiciones y por tanto la solución rápida no es óptima, puede establecerse una cota del error - absoluto entre la solución óptima y la solución rápida (|89|), o una cota del error relativo (|114|) entre ambas soluciones.

Considerese el problema knapsack anterior, con signo de igualdad en la restricción, en el que se asumirá, para todo j (j = 1,...,n).

1) a_j y c_j son enteros

2) $\rho_j \geq \rho_{j+1}$

3) $a_j > 0$

4) $a_1 = 1$

Los supuestos 1), 2) y 3) pueden hacerse sin pérdida de generalidad cuando el problema es acotado. El supuesto 4) se realiza para garantizar que el problema descrito tiene solución admisible cualquiera que sea el valor de b.

Dado el problema knapsack se define la función knapsack $f_k(y)$ como

$$f_k(y) = \min \sum_{j=1}^{k} c_j x_j$$

$$\sum_{j=1}^{k} a_j x_j = y$$

$$x_j \geq 0 \text{ entero } (j = 1,..,k)$$

Más adelante se analizará como mediante programación dinámica pueden calcularse recursivamente estas funciones hasta llegar al óptimo del problema original, $f_n(b)$. Para la función knapsack se construye la solución rápida $(x_1^G,\ldots,x_k^G)$, siendo

$$x_k^G = \left[\frac{y}{a_k} \right]$$

$$x_j^G = \left[(y - \sum_{i=j+1}^{k} a_i x_i^G)/a_j \right]$$

donde x_j^G se calcula ordenadamente para j = k-1, k-2,....,1.

Se denomina $g_k(y)$ al valor de la función objetivo de la función knapsack para la solución rápida. Es decir

$$g_k(y) = \sum_{j=1}^{k} c_j \, x_j^G$$

Si la solución rápida es óptima se cumplirá que $f_k(y) = g_k(y)$. Simplificando la notación, si $f_k(y) \leq g_k(y)$ para cualquier valor entero de y, escribimos que $f_k \leq g_k$.

Se cumple siempre que $g_1 = f_1$. Con las anteriores definiciones puede plantearse el siguiente teorema |89|.

Teorema 3.3. Sea $f_k = g_k$ y $a_{k+1} > a_k$ para $k < n$, entonces las siguientes cuatro expresiones son equivalentes.

1.- $g_{k+1} \leq g_k$

2.- $f_{k+1} = g_{k+1}$

3.- $f_{k+1}(m\ a_k) = g_{k+1}\ (m\ a_k)$

4.- $c_{k+1} + g_k(\gamma) \leq m\ c_k$

siendo m y γ los valores enteros definidos por las relaciones

$$0 \leq \gamma < a_k$$

$$a_{k+1} = m\, a_k - \gamma$$

Observese que el punto 3 implica que basta con comprobar el comportamiento de la solución rápida en el tamaño $m\, a_k$, para deducir si esta solución es óptima.

Los siguientes resultados acotan el error cuando la solución rápida no es óptima.

Sea la función error absoluto

$$\Delta_j(y) = g_j(y) - f_j(y)$$

y su valor máximo, para cualquier j,

$$\Delta_j^* = \max_{y=1,2,\ldots} \Delta_j(y)$$

El siguiento resultado permite calcular cotas superiores de este error de forma recurrente.

Sean

a) $k=\max \{j \mid f_j = g_j,\ j = 1,\ldots,n\}$

b) l_j, para cada j $(j = k+2,\ldots,n)$, el mínimo común múltiplo de a_{j-1} y a_j.

c) Ω_j una cota superior de Δ_j^*, para todo j $(j = k+1,\ldots,n)$

Teorema 3.4. Si $a_k < a_{k+1} < \ldots < a_n$, el cálculo de Ω_j puede realizarse recursivamente según la expresión

$$\Omega_j = \Omega_{j-1} + \max_{a_{j-1} \leq y \leq l_j} \{g_j(y) - g_{j-1}(y)\}$$

Con el resultado de este teorema, pueden obtenerse los valores de una cota superior del error absoluto, al tomar como óptimo del problema knapsack la solución rápida.

Analogamente se puede acotar la función error relativo

$$e_j(y) = \frac{g_j(y) - f_j(y)}{f_j(y)}$$

y su máximo valor, para todo j,

$$e_j^* = \max_y e_j(y)$$

Teorema 3.5. Si $f_k = g_k$, entonces

$$e_{k+1}^* \leq \frac{1 + \gamma - m}{m}$$

3.4.2. Multiplicadores de Everett.

Basado en los trabajos de Everett |38| y Brook y Geoffrión |18|, así como en el estudio para problemas de optimización con una restricción puede implementarse un método aproximado para resolver el problema knapsack. Debido a sus características, este método tiene sentido cuando las variables del problema son acotadas.

El problema 0-1 knapsack, caso especial de acotación de variables, ha sido resuelto por Gulley, Swanson y Wolsey apli-

cando los multiplicadores con resultados altamente satisfactorios.

El teorema fundamental de Everett, se aplica al pro-- blema knapsack general con variables acotadas

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_j x_j \leq b \qquad (3.2)$$

$$u_j \geq x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

Siendo λ un número no negativo, llamado multiplicador, el teorema muestra que el óptimo del knapsack es idéntico al óptimo del problema

$$\max \sum_{j=1}^{n} (c_j - \lambda a_j)x_j$$

$$u_j \geq x_j \geq 0 \text{ entero } (j = 1,\ldots,n) \qquad (3.3)$$

cuando $\sum_{j=1}^{n} a_j x_j^* = b$, donde x^* es la solución óptima de (3.3). Si $\sum_{j=1}^{n} a_j x_j^* = p \neq b$, una medida de la proximidad de x^* al óptimo -- de (3.2) es la diferencia (b-p). Cuando ésta es pequeña, la apro-

ximación es buena y el error cometido al considerar que x* es el óptimo del problema original, es reducido.

En el óptimo del problema transformado la variable -- x_j valdrá, dependiendo del valor λ^o escogido,

$$x_j = \begin{cases} u_j & \text{si } (c_j - \lambda^o a_j) > 0 \\ k & \text{si } (c_j - \lambda^o a_j) = 0, \text{ siendo k cualquier entero -} \\ & \text{tal que } u_j \geq k \geq 0 \\ 0 & \text{si } (c_j - \lambda^o a_j) < 0 \end{cases}$$

El valor de una variable x_j es una función escalón de λ, siendo el punto crítico $\lambda = c_j/a_j$.

Considerando las variables ordenadas según eficien--- cias crecientes ($c_j/a_j \leq c_{j+1}/a_{j+1}$ para todo j), se ensayan sucesivos valores para $\lambda(\lambda_1, \lambda_2, \ldots\ldots, \lambda_t)$. Así

$$\lambda_1 = \frac{c_n}{a_n}$$

$$\frac{c_{n-1}}{a_{n-1}} < \lambda_2 < \frac{c_n}{a_n}$$

$$\lambda_3 = \frac{c_{n-1}}{a_{n-1}}$$

$$\vdots$$

$$\lambda_t = \frac{c_1}{a_1}$$

hasta que para cierto valor λ_i, resulte que el óptimo x^* de (3.3) cumple

$$\sum_{j=1}^{n} a_j x_j^* \geq b$$

Como óptimo aproximado del problema (3.2) se toma el óptimo de (3.3) para λ_i o para λ_{i-1} según para cual de los dos, - la diferencia $b - \sum_{j=1}^{n} a_j x_j^*$ sea menor.

Es inmediato ver que si en el problema original las - variables x_j no son acotadas explicitamente, la cota implicita -- $[b/a_j]$ llevará mediante este método a una aproximación burda del óptimo.

3.5. Resolución por Programación Dinámica.

Al margen de la total enumeración, la programación dinámica es históricamente el primer método utilizado. Existen múltiples variantes para la resolución del problema knapsack en el contexto de la programación dinámica. Se desarrollarán a continuación los más significativos por ser los primeros o los más eficientes. Podemos señalar dentro de estos las recurrencias de Bellman |8|, Dantzig |27|, Gilmore y Gomory |47|, |49|, |50| y Greenberg |64|. Otros algoritmos de programación dinámica para el problema knapsack pueden encontrarse en Gilmore y Gomory |48| y Gomory |59|.

También dentro de este contexto, pueden considerarse los trabajos de Shapiro y Wagner |109| y Shapiro |108|, aunque por la originalidad de tratar el problema knapsack como un problema de maximizar la ruta a través de un grafo se analizará en otro apartado.

Bellman |8| propone una función recurrente para la resolución de un problema, en general, no lineal con una restricción lineal.

$$\max f(x_1,.....,x_n) = \sum_{i=1}^{n} g_i(x_i)$$

$$(3.4) \qquad \sum_{i=1}^{n} x_i = b$$

$$x_i \geq 0 \ (i = 1,...,n)$$

Definiendo la función

$$F_k(w) = \max f_k(x_1,...,x_k)$$

$$\sum_{i=1}^{k} x_i = w$$

$$x_i \geq 0 \ (i = 1,...,k)$$

puede plantearse la recurrencia

$$F_k(w) = \max_{0 \leq r \leq w} \{g_k(r) + F_{k-1}(w-r)\}$$

para $k = 2,...,n$ con

$$F_1(w) = g_1(w)$$

con lo cual, puede obtenerse el valor $F_n(b)$, óptimo de (3.4).

Basandose en la anterior función recurrente, Dantzig |27| propone su programa dinámico para la resolución del problema knapsack binario.

Sea

$$F_k(w) = \max \sum_{j=1}^{k} c_j x_j$$

$$\sum_{j=1}^{k} a_j x_j \leq w$$

$$x_j = 0 \text{ ó } 1 \quad (j = 1,\ldots,k)$$

el valor mayor que puede tomar la función objetivo supuesto que intervienen únicamente en su cálculo las k primeras variables y que la restricción de peso es w.

El valor de $F_{k+1}(w)$, con la misma limitación de peso, y además interviniendo la variable x_{k+1}, será igual a $F_k(w)$ si x_{k+1} es cero, o igual a $c_{k+1} + F_{k+1}(w - a_{k+1})$ si x_{k+1} es uno. De donde

$$F_{k+1}(w) = \begin{cases} \max\{F_k(w),\ c_{k+1} + F_{k+1}(w-a_{k+1})\} \text{ si } w \geq a_{k+1} \\ F_k(w) \quad \text{si } w < a_{k+1} \end{cases}$$

El procedimiento es iterativo para cada w(w = 0,...,b) y k (k = 1,...,n) partiendo de

$$F_0(w) = 0 \quad \text{para todo } w.$$

Gilmore y Gomory |49| extienden la recursión al problema knapsack general, en el que las variables pueden tomar valores superiores a la unidad, planteando

$$F_k(w) = \max \{F_{k-1}(w),\ c_k + F_k(w-a_k) \mid w \geq a_k\}$$

siendo $F_j(0) = 0$, para $j = 0,1,\ldots,n$, y $F_0(w) = 0$, para $w = 0,1,\ldots,b$.

Nótese que si en la obtención de $F_k(w)$ no interviene la variable k, entonces lógicamente $F_k(w)$ será igual a $F_{k-1}(w)$. - Si en la obtención de $F_k(w)$ interviene la variable k, entonces como mínimo valdrá 1 y podrá plantearse que $F_k(w) = c_k + F_k(w-a_k)$.

En la computación de $F_k(w)$ solo son necesarios los valores de $F_{k-1}(w)$, $w = 0,1,\ldots,b$. Una vez computados los valores de $F_{k-1}(w)$ pueden, en caso de que se estén realizando los cálculos con un ordenador, descargarse de la memoria los valores de $F_{k-2}(w)$, con el consiguiente ahorro de esta.

La siguiente recurrencia difiere de las anteriores en que procede del análisis de problemas derivados de knapsack donde intervienen todas las variables.

Sea

$$g(w) = \max \sum_{j=1}^{n} c_j x_j$$

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

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

Observese que

$$g(w) = F_n(w)$$

Gilmore y Gomory |50| muestran que

$$g(w) = \max_{i} \{0,\ g(w-a_i) + c_i \mid i \geq d(w-a_i),$$

$$w \geq a_i\}$$

siendo

$$d(w) = \max \{1,\ i \mid g(w) = c_i + g(w-a_i)\}$$

con lo cual se tiene una función de cálculo recurrente de $g(w)$ - $w = 0,1,\ldots\ldots b$.

El algoritmo resultante de la anterior recurrencia es el más eficiente de los existentes de programación dinámica.

Las recurrencias analizadas hasta ahora, calculan la función knapsack para todos los argumentos w desde 0 basta b. Si la restricción es con signo de igualdad, Greenberg |64| propone - la recurrencia

$$g(w) = \max \{c_j + g\ (w - a_j) \mid a_j \leq w\}$$

$$g(0) = 0$$

con la que se evita la total enumeración cuando $a_r = \min_j a_j$ es mayor que uno.

Esta recurrencia es útil para valores grandes de a_j, donde la total enumeración de w ($w = 0,1,\ldots,b$) supone calculos -

innecesarios, debido a que para muchos valores de w la función -- knapsack g(w) no tiene ninguna solución admisible.

3.6. Métodos de Enumeración.

Estas técnicas han sido ampliamente empleadas en el - estudio de la resolución del problema knapsack en sus diferentes formas |19, 20, 48, 67, 78, 79, 82, 94|.

El primer método de resolución fue propuesto por Gil- more y Gomory |48|, basándose en los trabajos sobre exploración - dirigida de Lang y Doig |83|. Aprovechando la particularización - de la técnica de eliminación de Fourier-Motzkin para este proble- ma, se obtiene el algoritmo de Cabot |19|. Por otro lado, Ingar-- giola y Korsh |79| basan el suyo, en la reducción del tamaño del problema a considerar. Estos son los algoritmos existentes más po tentes, desarrollados para el problema knapsack general

En la descripción de los algoritmos escogidos emplea- remos los conceptos de ordenación lexicográfica y extensión de -- vectores. Un vector x es lexicográficamente mayor que otro vector y ($x \prec y$) si en el vector diferencia, x-y, su primera coordenada no nula es positiva. Un vector y es una extensión de otro vector x si tiene más coordenadas que éste siendo las comunes las mis--- mas.

La enumeración de Gilmore y Gomory se realiza de forma lexicográficamente ordenada, eliminándose mediante un test algunas soluciones admisibles sin examinarlas explicitamente. El -- test se basa en que el óptimo de la solución continua del problema es una cota superior del óptimo con las restricciones de integridad.

En el problema knapsack,

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_j x_j \leq b \qquad (3.5)$$

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

se ordenan las variables de tal forma que $\rho_j \geq \rho_{j+1}$ $(j = 1,..n-1)$, y se calcula la solución admisible lexicográficamente máxima. Esta solución rápida es una cota inferior z^A admisible del óptimo.

A continuación se van enumerando las soluciones admisibles de forma lexicográficamente descendentes. Si alguna de las soluciones admisibles mejora la cota inferior se convierte en nueva cota inferior z^A. La enumeración se trunca mediante un test --

muy simple. En efecto, sea $\bar{x}^A = (x_1^A,\ldots,x_m^A)$ $(m < n)$ un vector tal que existan extensiones de $\bar{x}^A$ admisibles para (3.5). Si la solución óptima $\bar{z}$ del problema

$$z = \max \sum_{j=1}^{n} c_j x_j$$

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

$$x_j = x_j^A \quad (j = 1,\ldots,m)$$

$$x_j \geq 0 \quad (j = m+1,\ldots,n)$$

es menor que la cota inferior z^A, cualquier extensión de $(x_1^A,\ldots,x_m^A)$ no es óptima.

Aplicando la técnica de Fourier-Moztkin |30| a la eliminación de variables en el problema knapsack.

$$\max \quad z$$

$$\sum_{j=1}^{n} c_j x_j \geq z$$

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

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

se repiten problemas con solo dos restricciones. En el caso de la variable x_1 se obtiene

a) Si $c_1 > 0$

$$\sum_{j=2}^{n} (c_j a_1 - a_j c_1)x_j \geq za_1 - bc_1$$

$$\sum_{j=2}^{n} a_j x_j \leq b$$

$$x_j \geq 0 \text{ entero } (j = 2,\ldots,n)$$

b) Si $c_1 \leq 0$

$$\sum_{j=2}^{n} c_j x_j \geq z$$

$$\sum_{j=2}^{n} a_j x_j \leq b$$

$$x_j \geq 0 \text{ entero } (j = 2,\ldots,n)$$

Cabot |19| emplea sucesivamente esta técnica llegando finalmente a un sistema del tipo

$$c_n^n \; x_n \geq p_n \; z + q_n$$

$$a_n \; x_n \leq b$$

$$x_n \geq 0 \text{ entero}$$

Fijando z a un valor z^A, se obtiene una acotación de x_n ($L_n \leq x_n \leq U_n$). La elección de x_n entre estos valores implica a su vez una acotación sobre x_{n-1} ($L_{n-1} \leq x_{n-1} \leq U_{n-1}$). Repitiendo este proceso, se llega a una de las siguientes situaciones :

a) Para algún i, el intervalo $[L_i, U_i]$ no contiene -- ningún valor entero.

b) El intervalo $[L_1, U_1]$ contiene valores enteros.

En el primer caso no existe solución admisible con -- los valores previamente fijados. En el segundo se ha encontrado - una solución admisible y su correspondiente valor z^A, cota infe-- rior de la solución óptima.

Cabot propone partir de una "buena" cota admisible -- del problema knapsack z^A, e ir incrementando el valor hasta que - no sea posible encontrar una solución admisible.

La cota admisible inicial puede ser, por ejemplo, la obtenida por la solución rápida, y el incremento de z^A ser cada - vez el máximo común divisor de $c_1, c_2, \ldots, c_n$. Cuando para un valor z^S no sea posible encontrar ninguna solución admisible, el valor $z^* = z^S$ - m.c.d. $(c_1, \ldots, c_n)$ será el óptimo buscado.

Siendo

$$d_j = \left[\frac{b}{a_j} - \frac{z^A}{c_j} \right]$$

ordena las variables con d_j decreciente en j para conseguir la máxima eficiencia de su algoritmo.

Basandose en un trabajo sobre el knapsack binario -- (|78|), Ingargiola y Korsh |79| extienden los resultados obteni-- dos al problema general. Proponen dos algoritmos bien diferenciados para resolverlo. El primero transforma el original en otro de menor complejidad. A continuación el segundo algoritmo, de exploraciones parciales (branch-search), resuelve el transformado. Esta opción es la más eficiente según Ingargiola y Korsh. Sin embargo, se pueden utilizar ambos de forma independiente. Puede aplicarse el algoritmo de reducción al problema original y, posteriormente, resolver el reducido por cualquier otro algoritmo de resolución. Otra opción es aplicar directamente el algoritmo de exploración - parcial a un problema knapsack cualquiera.

En el algoritmo de reducción, el problema

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_j x_j \leq b \tag{3.6}$$

$$0 \leq x_j \leq d_j \text{ entero } (j = 1,\ldots,n)$$

donde d_j puede ser una cota superior explicitamente dada o bien implicita $d_j = [b/a_j]$ $(j = 1,\ldots,n)$, se transforma en

$$\max \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_j x_j \leq b \tag{3.7}$$

$$u_j \geq x_j \geq l_j \text{ entero } (j = 1,\ldots,n)$$

siendo u_j, l_j $(j = 1,\ldots,n)$ valores enteros. En general, para algunas variables $u_j = 0$ lo cual implica que $x_j = 0$; para otras variables $u_j = l_j$ entonces $x_j = u_j$. Y también $l_j > 0$ con lo que puede hacerse un cambio de variables $\bar{x}_j = x_j - l_j$.

El problema (3.7) puede reescribirse de nuevo eliminando algunas variables. Distinguiendo según las cotas de las variables,

$$U = \{j \mid u_j = l_j\} \quad , \quad V = \{j \mid u_j = 0\} \quad ,$$

$$B = \{j \mid u_j > l_j,\ l_j > 0\} \ , \ D = \{j \mid u_j > l_j,\ l_j = 0\}$$

se tiene el problema transformado

$$\sum_{j\in U} c_j u_j + \max \sum_{j\in D} c_j x_j + \sum_{j\in B} c_j x_j$$

$$\sum_{j\in D} a_j x_j + \sum_{j\in B} a_j x_j \leq b - \sum_{j\in U} a_j u_j = \overline{b}$$

$$u_j \geq x_j \geq l_j, \text{ entero}$$

Haciendo el cambio de variables $\overline{x}_j = x_j - l_j$, $u_j = u_j - l_j$ para $j \in B$ resulta

$$\sum_{j\in U} c_j u_j + \sum_{j\in B} c_j l_j + \max \left(\sum_{j\in D} c_j x_j + \sum_{j\in B} c_j \bar{x}_j \right)$$

$$\sum_{j\in D} a_j x_j + \sum_{j\in B} a_j \bar{x}_j \leq \bar{b} - \sum_{j\in B} a_j l_j$$

$$u_j \geq x_j \geq 0 \text{ entero } (j \in D)$$

$$u_j \geq \bar{x}_j \geq 0 \text{ entero } (j \in B)$$

que es un problema knapsack semejante al problema original pero - de menor número de variables. Este problema puede ser resuelto -- por cualquiera de los algoritmos existentes.

Tanto el algoritmo de reducción como el de explora--- ción parcial están basados en un teorema. Para la exposición del algoritmo es preciso realizar previamente algunas definiciones sobre el problema (3.6) :

- Una asignación es un vector $y = (y_1,\ldots,y_n)$ que satisface $0 \leq y_j \leq d_j$ para todo j $(j = 1,..,n)$.

- Para una asignación y se define su función VALOR (y) como

$$\text{VALOR } (y) = \sum_{j=1}^{n} c_j y_j$$

- Una <u>hipotesis</u> es un vector $x = (x_1,\ldots,x_n)$ tal que $x_j \geq 0$ entero ó $x_j = -1$ para todo j $(j = 1,\ldots,n)$.

- Una <u>asignación y es compatible con una hipotesis x</u> si es admisible y entera, y cumple para $x_i \geq 0$ que $y_i = x_i$ para todo i $(i = 1,\ldots,n)$.

- Se denomina A(x) al conjunto de asignaciones compatibles con una hipotesis x.

- Una <u>hipotesis x es admisible</u> si el conjunto A(x) no es vacio.

- Se define $\underline{\tilde{x}}$ como el elemento de A(x) tal que VALOR $(\tilde{x})$ $\geq$ VALOR (x) para todo $x \in A(x)$.

- Se llama <u>E</u> a la hipotesis (-1, -1,., -1).

- Para la hipotesis x donde $x_i < 0$, se denota por $x^{i,j}$ a la hipotesis donde todas las componentes son iguales a las de x, excepto la componente i que toma el valor j.

- Se denomina <u>pivot de una hipotesis x</u>, p(x), a :

1. Si x es admisible no trivial al menor entero r tal que

$$\sum_{i \in I} a_i d_i \geq b - \sum_{j \in J} a_j x_j$$

siendo $I = \{i | x_i < 0, i = 1,\ldots,r\}$, $J = \{j | x_j \geq 0, j = 1,\ldots,n\}$.

2. Si x no es admisible $p(x) = 0$.

3. Si x es trivial $p(x) = n + 1$

- Asignación continua LP(x) compatible con una hipo-- tesis es una asignación tal que

$$LP(x)_i = \begin{cases} x_i \quad si \quad x_i \geq 0 \\ d_i \quad si \quad x_i < 0, \; i < p(x) \\ 0 \quad si \quad x_i < 0, \; i > p(x) \\ (b - \sum_{j \in J} a_j x_j - \sum_{k \in K} a_k d_k)/a_i \quad si \\ \qquad\qquad\qquad\qquad x_i < 0, \; i = p(x) \end{cases}$$

siendo $J = \{j | x_j \geq 0,\ j = 1,..,n\}$,

$$K = \{j | x_j < 0,\ j = 1,..,p(x)\}.$$

- Se define m(x) para una hipotesis x, a la parte entera del valor de la componente pivot de la asignación continua de x. Es decir

$$m(x) = \left[LP(x)_{p(x)}\right]$$

- Solución cota inferior LB(x) compatible con una hipotesis x, es una asignación tal que $LB(x)_i = LP(x)_i$ si $i \neq p(x)$ y $LB(x)_{p(x)} = m(x)$.

A partir de las anteriores definiciones Ingargiola y Korsh muestran el siguiente resultado :

Teorema 3.6. Sea y una asignación perteneciente a A(x), la hipotesis $x^{i,j}$ es admisible y además VALOR (y)>VALOR ($LP(x^{i,j})$), entonces :

a) Si $y_i > j$ entonces $\mathcal{X} \geq j+1$

b) Si $y_i < j$ entonces $\mathcal{X} \leq j-1$

Basandose en este resultado puede implementarse ambos algoritmos, el de reducción y el de exploración parcial.

3.7. Transformación.

De igual forma que en programación lineal entera, un método de abordar el problema knapsack es transformarlo en otro - de idéntico óptimo, pero más agradable de resolver.

Se expone en primer lugar la transformación de un problema knapsack acotado en otro normalizado (value-independent -- knapsack problem), y un algoritmo de cálculo del número de soluciones admisibles de una ecuación diofántica. A continuación se - describe un algoritmo de programación dinámica basado en él, para resolver el problema knapsack normalizado.

Otra linea de ataque del problema, que más que una -- transformación es un nuevo punto de vista, es la correspondencia entre él y uno de ruta mínima a través de un grafo, propuesta por Shapiro |108|.

Bradley |16| mostró, que para cualquier problema knapsack, existe un vector entero $H = (h_1,\ldots,h_n)$ y valores enteros - γ, Π_1, Π_2 tales que el óptimo del problema

$$\gamma + \max \sum_{j=1}^{n} h_j x_j$$

$$(3.8) \qquad \Pi_1 \leq \sum_{j=1}^{n} h_j x_j \leq \Pi_2$$

$$x_j \geq 0 \text{ entero } (j = 1,\ldots,n)$$

llamado knapsack normalizado, es el mismo que el óptimo del original. Además, el conjunto de soluciones admisibles y los correspondientes valores de las funciones objetivas son idénticos.

Faaland |39|, |40| propone el siguiente algoritmo de resolución de este problema.

Sea $f_k(i)$ el número de soluciones de la ecuación diofántica

$$\sum_{j=1}^{k} h_j x_j = i$$

para $0 \leq x_j \leq d_j$. Obviamente, el valor óptimo de (3.8) es

$$z^* = \max \{i \mid f_n(i) \neq 0,\ \pi_1 \leq i \leq \pi_2\}$$

Para obtener este valor, z^*,se calculan los de $f_k(i)$ sucesivamente, para todo i ($i = 0,1,\ldots\pi_2$) y para todo k ($k = 1,2,\ldots,n$), según la función recurrente

$$f_k(i) = f_{k-1}(i) \quad \text{para} \quad 1 \leq i \leq h_k - 1$$

$$f_k(i + h_k) = \begin{cases} f_{k-1}(i + h_k) + f_k(i) \text{ para } 0 \leq i \leq h_k\, d_k \\ f_{k-1}(i + h_k) + f_k(i) - f_{k-1}(i - h_k\, d_k) \text{ para} \\ \qquad\qquad h_k\, d_k \leq i \leq \pi_2 - h_k \end{cases}$$

que se resuelve por programación dinámica.

Por otra parte, Shapiro |108| mostró la relación entre el knapsack y ciertos problemas definido sobre grafos unimodulares: existe una correspondencia biunivoca entre las soluciones admisibles del problema knapsack y el conjunto de rutas del nodo de salida al de entrada de un grafo orientado $G \equiv \{N,A\}$. El conjunto de nodos es $N = \{0,1,\ldots,b\}$, donde 0 es el nodo de salida y b el de entrada, y el de arcos, $A = \{(i,j) \mid j - i = a_k$ para todo k ($k = 1,\ldots,n$)$\}$. A cada arco (i,j) tal que $a_k = j - i$ se le asocia la distancia c_k. Con estos elementos, la ruta máxima de 0

a b, z*, en el grafo G, es el valor óptimo del knapsack. Los ar-- cos que intervienen en esta ruta describen el valor óptimo de las variables.

Con estos resultados, se puede resolver el problema - knapsack calculando la ruta máxima del grafo asociado al proble-- ma. La resolución del problema de la ruta máxima se realiza por - cualquiera de los algoritmos conocidos para ese problema (|32|).

CAPITULO IV

NUEVOS METODOS DE RESOLUCION DEL PROBLEMA KNAPSACK

4.1. Introducción

En este capítulo se estudían dos métodos de resolución del problema knapsack basados en la enumeración lexicográfica de soluciones admisibles y un método de transformación para proble-- mas de ciertas características muy específicas.

El primer método conjuga por una parte aspectos de la reducción de variables de Fourier-Motzkin, ya empleado por Cabot para el knapsack, con criterios de eliminación de soluciones, sugeridos por los trabajos de Gilmore y Gomory. El segundo método - modifica el propuesto por Cabot. En ambos se emplean simultaneamente cotas superiores e inferiores del óptimo de la función objetivo y se enumeran soluciones admisibles de acuerdo con una ordenación lexicográfica de los vectores. De acuerdo con los resultados se presentan dos algoritmos de resolución del problema knap-- sack, cuyas experiencias computacionales se exponen en el capítulo VII.

En todo el capítulo empleamos el concepto de aporta--- ción marginal al knapsack, $\rho_j = c_j/a_j$, siendo c_j y a_j los coefi-- cientes de la variable j-ésima del problema. Consideramos las variables ordenadas de forma que las aportaciones marginales sean - crecientes. Debido a ello utilizamos la ordenación lexicográfica en el sentido decreciente de los índices: al ordenar lexicográficamente dos vectores se comparan de las últimas coordenadas a las primeras.

4.2. Enumeración Lexicográfica con Acotaciones Sucesivas.

El problema knapsack

$$\max \sum_{j=1}^{n} c_j x_j$$

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

$$x_j \geq 0 \text{ entero}$$

con c_j, a_j y b enteros y positivos se puede reescribir como

$$\max \quad z$$

$$\sum_{j=1}^{n} c_j x_j \geq z$$

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

$$x_j \geq 0 \text{ entero}$$

Los pares admisibles son vectores (x,z) de n+1 coordenadas en que las n primeras x_j representan las n actividades del problema y la última, z, el valor de la función objetivo. Nos referiremos a par entero si todas las coordenadas son enteras positivas.

Aplicando secuencialmente la técnica de eliminación de Fourier-Motzkin a las restricciones del problema se obtiene el sistema equivalente

$$\sum_{j=k}^{n} c_j^k x_j \geq z^k$$

(4.1)

$$\sum_{j=k}^{n} a_j x_j \leq b$$

para $k = 1,\ldots,n$. Nótese que el sistema anterior consta de 2n desigualdades, siendo las originales las que corresponden a $k = 1$ y las restantes, relajaciones de éstas.

En función de los datos originales

$$c_j^k = (c_j\ a_{k-1} - a_j\ c_{k-1})\ a_1\ a_2 \cdots a_{k-2}$$

$$z^k = (z\ a_{k-1} - b\ .\ c_{k-1})\ a_1\ a_2 \cdots a_{k-2}$$

Por tanto el sistema (4.1) es equivalente a

$$\sum_{j=k}^{n} (\rho_j - \rho_{k-1})\ a_j\ x_j + b\ \rho_{k-1} \geq z$$

$$\sum_{j=k}^{n} a_j\ x_j \leq b$$

para $k = 1,\ldots,n$.

El problema knapsack lo resolveremos mediante el análisis de una secuencia de n problemas p(k) definidos como

$$\max \quad z$$

$$\sum_{j=k}^{n} (\rho_j - \rho_{k-1}) a_j x_j + b\, \rho_{k-1} \geq z$$

$$\sum_{j=k}^{n} a_j x_j \leq b$$

$$x_j \geq 0 \text{ entero } (j = k,\ldots,n)$$

Por conveniencia definimos, para un par dado (x,z), -- las funciones

$$H_k(x,z) \equiv \sum_{j=k}^{n} (\rho_j - \rho_{k-1})\, a_j\, x_j + b\, \rho_{k-1} - z$$

$$R_k(x) \equiv b - \sum_{j=k}^{n} a_j\, x_j$$

que representan las holguras en las restricciones.

La admisibilidad del par entero (x^o, z^o) para el problema P(k) es equivalente, por definición, a que $H_k(x^o, z^o) \geq 0$ y $R_k(x^o) \geq 0$.

Teorema 4.1.- Sea (x^o, z^o) admisible para el problema P(k). Entonces,

1 - Para $1 \leq k \leq p \leq n$

$$H_k(x^o,z^o) \leq H_p(x^o, z^o) \quad y$$

$$R_k(x^o) \leq R_p(x^o)$$

2 - Si $R_k(x^o) = 0$, entonces $(\overline{x},\overline{z})$ es admisible para P(1) siendo $\overline{x} = (0,\ldots,0, x_k^o,\ldots,x_n^o)$ y $\overline{z} = z^o + [H_k(x^o,z^o)]$.

Demostración.-

1 - De acuerdo con la definición de $H_k(x,z)$, para todo k

$$(4.2) \qquad H_{k+1}(x^o,z^o) - H_k(x^o,z^o) = (\rho_k - \rho_{k-1})\, R_k(x^o) \geq 0$$

La relación de $R_k(x^o)$ es trivial.

2 - Hemos de mostrar que $R_1(\overline{x}) \geq 0$ y $H_1(\overline{x},\overline{z}) \geq 0$.

Si $R_k(x^o)$ es cero

$$R_1(\overline{x}) = ... = R_{k-1}(\overline{x}) = R_k(\overline{x}) = 0$$

Además

$$H_1(\overline{x},\overline{z}) = H_k(\overline{x},\overline{z}) \qquad |R_k(\overline{x}) = 0|$$

$$= H_k(x^o,\overline{z}) \qquad |\overline{x} = (0,..,0,x^o_k,...x^o_n)|$$

$$= H_k(x^o,z^o) + (z^o-\overline{z}) \qquad |\text{definición } H_k(x,z)|$$

$$\geq 0 \qquad |\text{definición de } \overline{z}|.//$$

Corolario 4.1 - Sea (x^o,z^o) admisible para p(k). El par entero -- (x^o,z^o) es admisible para p(k-1) si y solo si satisface

$$a_{k-1}\, x^{o}_{k-1} + H_k(x^o,z^o)/(\rho_{k-1} - \rho_{k-2}) \geq R_k(x^o) \geq a_{k-1}\, x^{o}_{k-1}$$

Demostración.-

La segunda desigualdad impone que $R_{k-1}(x^o) \geq 0$ y la -- primera equivale a $H_{k-1}(x^o,z^o) \geq 0$.//

Corolario 4.2. Sea (x^o,z^o) admisible para p(k). Si $R_k(x^o) \equiv 0 \pmod{a_{k-1}}$ entonces $(\overline{x},\overline{z})$ es admisible para p(1), siendo

$$\overline{x} = (0,\ldots,0,\ R_k(x^o)/a_{k-1},\ x^o_k,\ldots,x^o_n)$$

$$\overline{z} = z^o + [H_k(x^o,z^o)]$$

Demostración.-

Haciendo $\overline{x}_{k-1} = R_k(x^o)/a_{k-1}$, por la relación de con--- gruencia $\overline{x}_{k-1}$ es entero y $R_{k-1}(\overline{x}) = 0$. El resultado se sigue del teorema 4.1.//

Proposición 4.1. - Sea $x^o = (x_1^o,\ldots,x_p^o, 0,\ldots,0, x_k^o,\ldots,x_n^o)$ y -- (x^o,z^o) admisible para p(k). El par (x^o,z^o) es admisible para -- P(p+1) si y solo si

$$\rho_p \geq \rho_{k-1} - H_k(x^o,z^o)/R_k(x^o)$$

Demostración.-

Obviamente $R_{p+1}(x^o) = R_k(x^o) \geq 0$.

Hemos de mostrar que $H_{p+1}(x^o,z^o) \geq 0$.

$$H_k(x^o,z^o) - H_{p+1}(x^o,z^o) = \sum_{j=p+1}^{k-1} (\rho_j-\rho_{j-1}).R_j(x^o) \quad |\text{según 4.2}|$$

$$= R_k(x^o) \cdot \sum_{j=p+1}^{k-1} (\rho_j-\rho_{j-1}) \quad |R_j(x^o) = R_k(x^o) \text{ para } j = p+1,\ldots,k|$$

$$= R_k(x^o).(\rho_{k-1} - \rho_p) \quad |\text{suma telescopica}|$$

Luego

$$H_{p+1}(x^o,z^o) = H_k(x^o,z^o) - R_k(x^o)(\rho_{k-1} - \rho_p)$$

de donde se deduce el resultado.//

Comentarios.-

Nótese que los problemas P(k) son sucesivas relajaciones del problema original: cada uno es una relajación del anterior. De esta forma se acota progresivamente supuestos valores de la función objetivo z empleando la función H_k. Esta indica, en el problema P(k), el exceso sobre la cota fijada z, que ha de mantenerse positivo hasta p(1) para que z sea admisible. Si es así, z se convierte en una cota inferior admisible del problema original. Para mantener la restricción del tamaño del knapsack se emplea la función R_k, que es la parte aún no asignada, manteniéndola siempre positiva. Ambos indicadores están intimamente relacionados, siendo el incremento de H_k a H_{k+1} directamente proporcional a R_k:

$$H_{k+1}(x,z) - H_k(x,z) = (\rho_k - \rho_{k-1})\, R_k(x)$$

El teorema formaliza estos resultados, que son empleados para reducir la cantidad de enumeraciones aumentando rápidamente el valor de la cota inferior. En los dos corolarios se explicitan expresiones análiticas que disciernen si un vector admisible para P(k) lo es para P(k-1). Bajo determinadas condiciones no es necesario aplicar el test a toda la secuencia de problemas, pasandose directamente de admisibilidad para P(k) a admisibilidad para P(1). En el caso frecuente en que la holgura del knapsack -- R_k, es más pequeña que el tamaño a_j $(p < j < k)$ para algunas variables, la proposición presenta un sencillo test que comprueba si P(p) puede alcanzar la cota.

En conjunto, estos resultados permiten examinar eficientemente la existencia de solución admisible, correspondiente a cierta cota z, para el problema P(1).

4.3. Algoritmo de Enumeración.

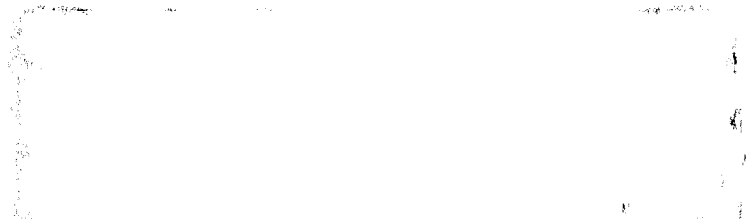
Usando los anteriores resultados se ha desarrollado un algoritmo. Partiendo de una cota superior no admisible z^S y una inferior admisible z^I, el algoritmo disminuye, en sucesivas iteraciones, el valor de la cota superior y aumenta el de la inferior, hasta llegar al óptimo cuando $z^S = z^I + 1$.

Se utiliza como cota superior inicial z^S, el óptimo de la solución continua redondeada superiormente, como inferior z^I - la solución rápida. Para el valor medio z^M ($z^M = (z^I + z^S)/2$) se busca, aplicando los resultados obtenidos, admisibilidad para el problema P(1). Se inicia la enumeración a partir de la solución - lexicográfica máxima x^L, descendiendo lexicográficamente hasta es tablecer si z^M es admisible o no para P(1). Si lo es, para el vec tor admisible x^o, se actualizan z^I y z^M haciendo z^I igual a -- $z^M + H_1(x^o, z^M)$ y $z^M = (z^I + z^S)/2$, y se vuelve a buscar admisi-- bilidad para z^M, descendiendo lexicográficamente a partir de x^o. Si z^M no es admisible se actualizan z^S igual a z^M y $z^M = (z^I + z^S)/2$ repitiendose el proceso a partir de x^L.

En general, cuando la cota resulta admisible, se incre menta en $H_1(x,z)$ convirtiendose en nueva cota inferior admisible. Caso contrario será la nueva cota superior no admisible. Siempre, tras calcular una nueva cota, el descenso lexicográfico se comien za partiendo del último vector admisible obtenido.

En la figura 4.1 se esquematiza el desarrollo general del algoritmo.

A continuación se expone la lista de operaciones del - algoritmo y su diagrama de flujo detallado.



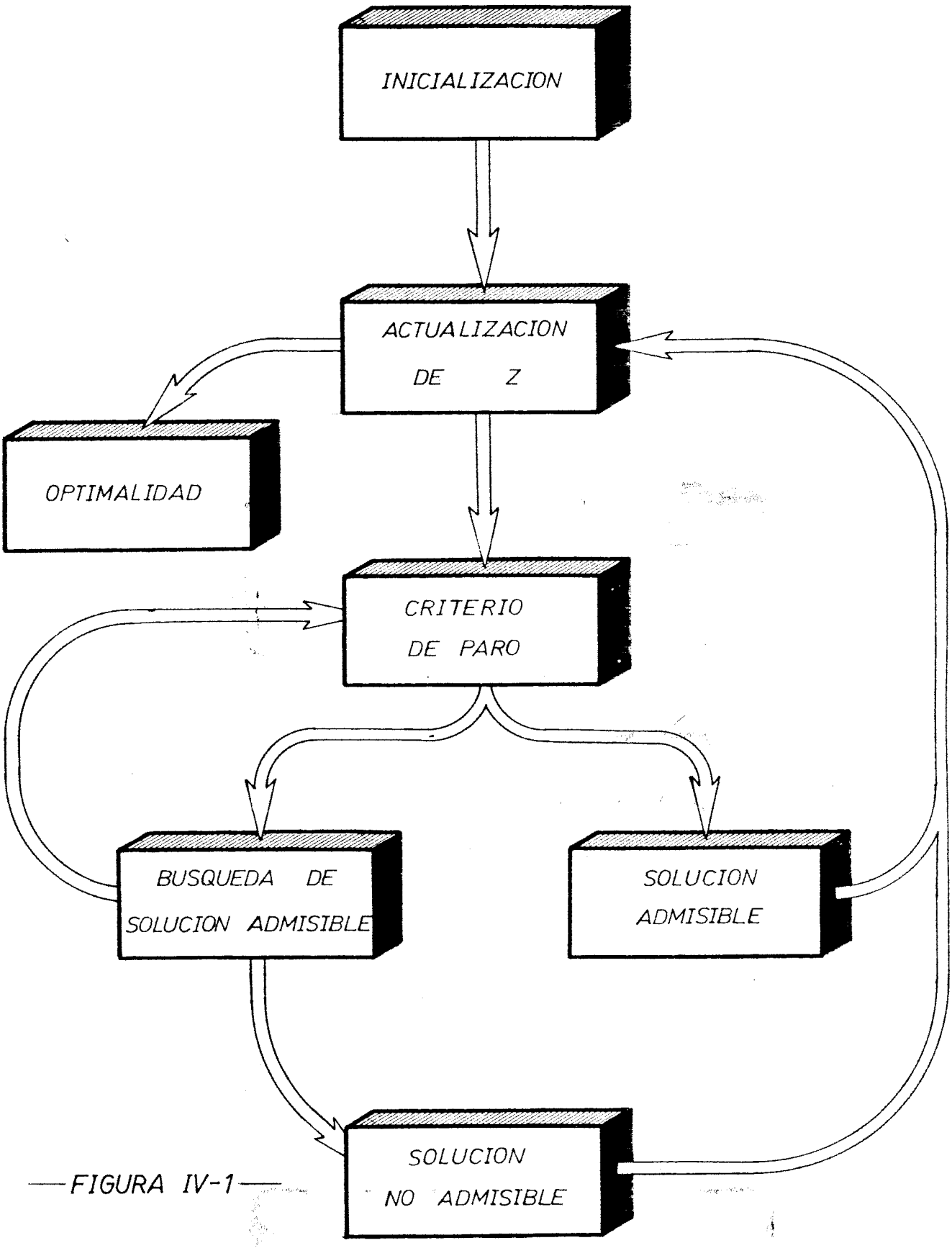

INICIALIZACION
ACTUALIZACION DE Z
OPTIMALIDAD
CRITERIO DE PARO
BUSQUEDA DE SOLUCION ADMISIBLE
SOLUCION ADMISIBLE
SOLUCION NO ADMISIBLE

FIGURA IV-1

Desarrollo del algoritmo.

1. Ordenar las variables tales que $\rho_1 \leq \ldots \leq \rho_n$. Calcular la solución rápida zo, xo(I). Hacer Ro(I) = C(I)/A(I), DiRo(I) = Ro(I) - Ro(I-1), DiRo(1) = Ro(1). Caltular HV(I) = DiRo(I) * A(I) * xo(I) = B * Ro(I-1) - zo para todo I. Hacer RV = B - Σ x o(I) * A(I).

2. Hacer zI = Ro(N) * B . Ir a 3.

3. Si zI = zo ir a 4. Caso contrario ir a 5.

4. Fin del algoritmo.

5. Hacer z = |(zI + zo)/2 + 0.5|. Si z = zI ir a 6. Caso contrario ir a 7.

6. Hacer z = zo ir a 4.

7. Hacer H(I) = HV(I) + zo - z, x(I) = xo(I) para todo I. Hacer R = RV, I = 0 ir a 8.

8. Hacer I = I + 1. Si x(I) $\leq$ 0 ir a 9. Caso contrario ir a 10.

9. Si I < N ir a 8. Caso contrario ir a 12.

10. Hacer x(I) = x(I) - 1, R = R + A(I), H = H(I) - DiRo(I) * -- A(I). Si H $\geq$ 0 ir a 14. Caso contrario ir a 11.

11. Hacer R = R + A(I) * x(I), x(I) = 0. Ir a 9.

12. Si z = zI ir a 4. Caso contrario ir a 13.

13. Hacer zI = z. Ir a 3.

14. Hacer ROM = Ro(I-1) - H(I)/R, IND = I. Ir a 15.

15. Si I = 2 ir a 25. Si I < 2 ir a 21. Si I > 2 ir a 16.

16. Hacer I = I - 1. Si R < A(I) ir a 17. Caso contrario ir a 18.

17. Si Ro(I) < ROM ir a 8. Caso contrario ir a 15.

18. Hacer x(I) = R/A(I), H(I) = H(IND) - R * (Ro(IND-1) - Ro(I-1)) + A(I) * x(I) * DiRo(I). Si H(I) $\geq$ 0 ir a 19. Caso contrario ir a 22.

19. Hacer R = R - A(I) * x(I), IND = I. Si R $\leq$ 0 ir a 21. Caso -- contrario ir a 20.

20. Hacer ROM = Ro(I + 1) - H(I)/R. Ir a 15.

21. Hacer xo(I) = x(I) para todo I. Hacer zo = z + H(IND). Hacer H(I) = H(I) - H(IND) para todo I. Ir a 23.

22. Hacer x(I) = 0. Ir a 8.

23. Si zI $\leq$ zo ir a 4. Caso contrario ir a 24.

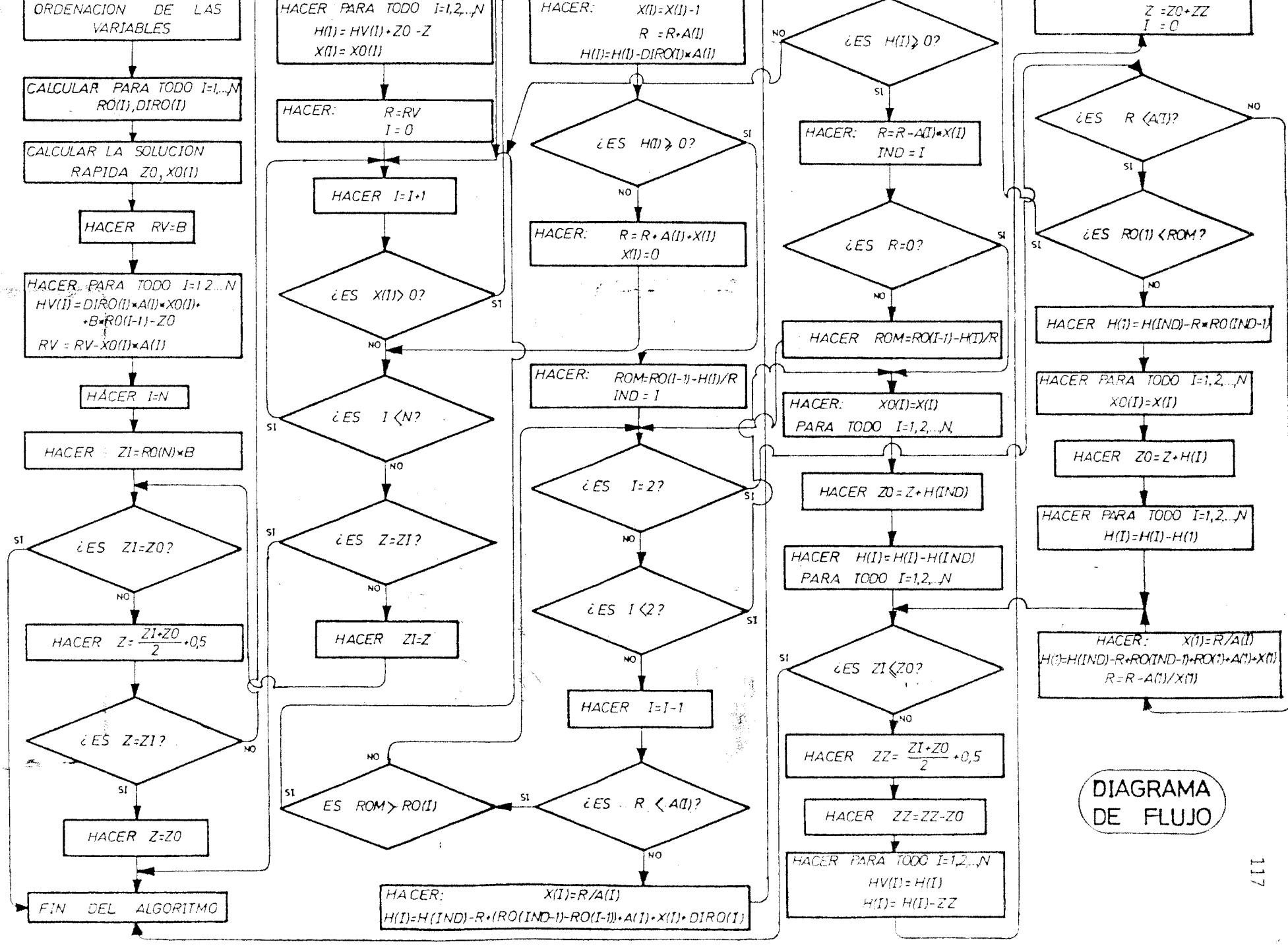
24. Hacer zz = |(zI + zo)/2 + 0.5|. Hacer zz = zz = zo. Hacer HV(I) = H(I) para todo I. Hacer H(I) = H(I) - zz para todo I. Hacer RV = R, z = zo + zz, I = 0. Ir a 8.

25. Si R < A(I) ir a 26. Caso contrario ir a 28.

26. Si Ro(1) < ROM ir a 8. Caso contrario ir a 27.

27. Hacer H(1) = H(IND) - R * Ro(IND - 1). Hacer xo(I) = x(I) para todo I. Hacer zo = z + H(1). Hacer H(I) = H(I) - H(1) para todo I. Ir a 23.

28. Hacer x(1) = R/A(1), H(1) = H(IND) - R * Ro(IND - 1) + Ro(1) * A(1) * x(1), R = R - A(1) * x(1). Ir a 23.


ORDENACION DE LAS VARIABLES
CALCULAR PARA TODO I=1,..,N RO(I), DIRO(I)
CALCULAR LA SOLUCION RAPIDA Z0, X0(I)
HACER RV=B
HACER PARA TODO I=1 2 ..N HV(I)=DIRO(I)*A(I)*X0(I)+ +B*RO(I-1)-Z0 RV = RV-X0(I)*A(I)
HACER I=N
HACER Z1=RO(N)*B
¿ES Z1=Z0?
SI
NO
HACER Z=(Z1+Z0)/2+0,5
¿ES Z=Z1?
SI
NO
HACER Z=Z0
FIN DEL ALGORITMO
HACER PARA TODO I=1,2,..,N H(I)=HV(I)+Z0-Z X(I)=X0(I)
HACER: R=RV I=0
HACER I=I+1
¿ES X(I)>0?
SI
NO
¿ES I<N?
SI
NO
¿ES Z=Z1?
SI
NO
HACER Z1=Z
¿ES ROM>RO(I)
SI
NO
HACER: X(I)=X(I)-1 R=R+A(I) H(I)=H(I)-DIRO(I)*A(I)
¿ES H(I)>0?
SI
NO
HACER: R=R+A(I)*X(I) X(I)=0
HACER: ROM=RO(I-1)-H(I)/R IND=I
¿ES I=2?
SI
NO
¿ES I<2?
SI
NO
HACER I=I-1
¿ES R<A(I)?
SI
NO
HACER: X(I)=R/A(I) H(I)=H(IND)-R*(RO(IND-1)-RO(I-1))*A(1)*X(I)*DIRO(I)
¿ES H(I)>0?
NO
SI
HACER: R=R-A(I)*X(I) IND=I
¿ES R=0?
SI
NO
HACER ROM=RO(I-1)-H(I)/R
HACER: X0(I)=X(I) PARA TODO I=1,2,..,N
HACER Z0=Z+H(IND)
HACER H(I)=H(I)-H(IND) PARA TODO I=1,2,..,N
¿ES Z1<Z0?
SI
NO
HACER ZZ=(Z1+Z0)/2+0,5
HACER ZZ=ZZ-Z0
HACER PARA TODO I=1,2,..,N HV(I)=H(I) H(I)=H(I)-ZZ
Z=Z0+ZZ
I=0
¿ES R<A(I)?
NO
SI
¿ES RO(1)<ROM?
SI
NO
HACER H(1)=H(IND)-R*RO(IND-1)
HACER PARA TODO I=1,2,..,N X0(I)=X(I)
HACER Z0=Z+H(I)
HACER PARA TODO I=1,2,..,N H(I)=H(I)-H(1)
HACER: X(1)=R/A(1) H(1)=H(IND)-R*RO(IND-1)+RO(1)*A(1)*X(1) R=R-A(1)*X(1)
DIAGRAMA DE FLUJO

4.4. Modificaciones de la Enumeración de Cabot. Algoritmo.

En el método de enumeración que propone Cabot se parte de una cota inferior admisible de la función objetivo z. Se incrementa en un cierto valor Δz y se busca hasta encontrar una solución admisible. Este proceso continua hasta que no existe solución admisible para la cota propuesta (figura 4.2).

Utilizando el mismo método de enumeración se propone partir de una cota superior z de la función objetivo. Si resulta admisible, es la óptima. Caso contrario se decrementa una cierta cantidad repitiendose el proceso (figura 4.3).

Se pueden señalar dos ventajas de esta modificación respecto al método original.

a) Debido a que las cotas de las variables se calculan para valores mayores de z, el valor de las inferiores l_j serán más grandes disminuyendo el intervalo (u_j, l_j). Al ser la cantidad de evaluaciones función directa del tamaño de éste disminuiran las operaciones con lo que el algoritmo gana en eficiencia.

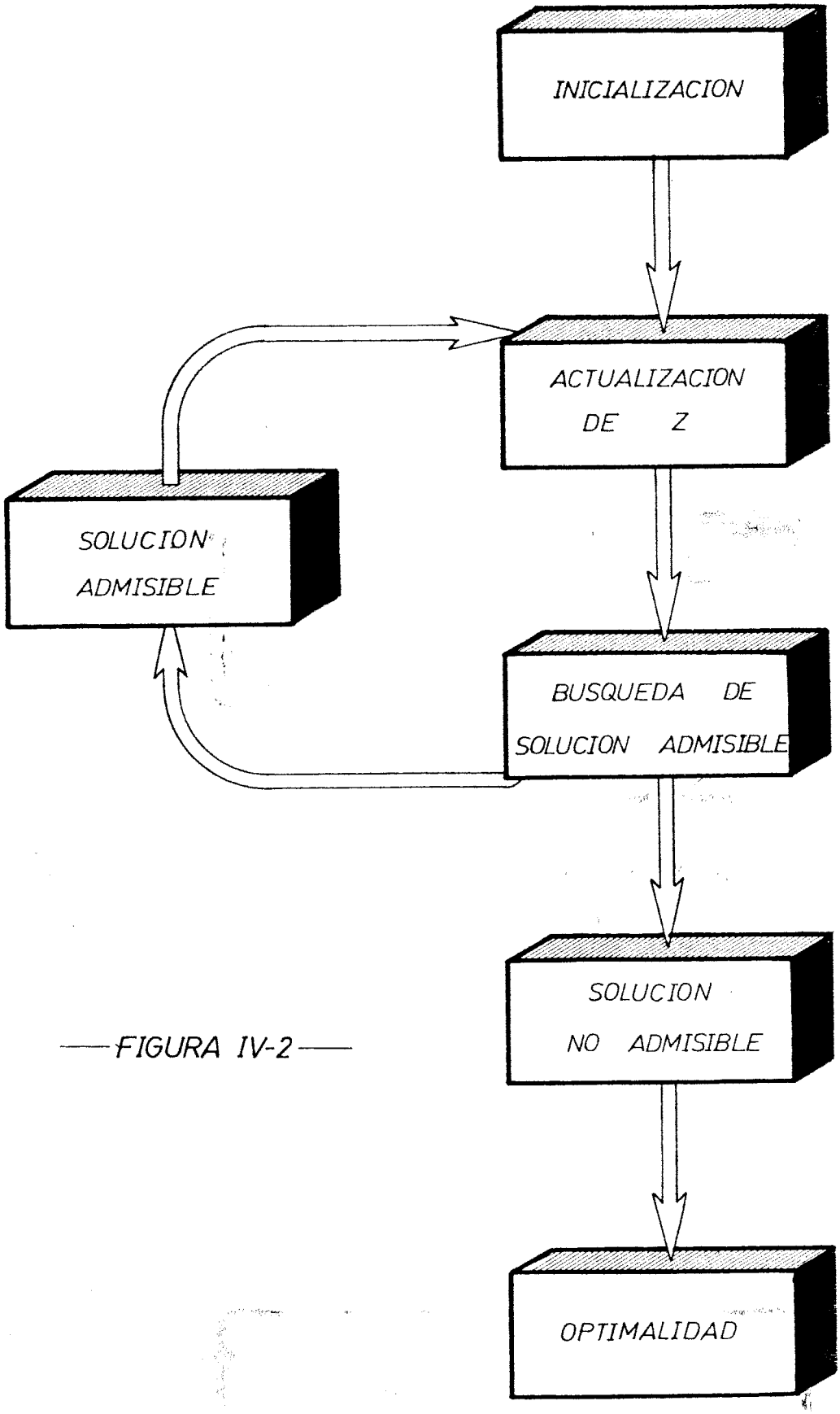
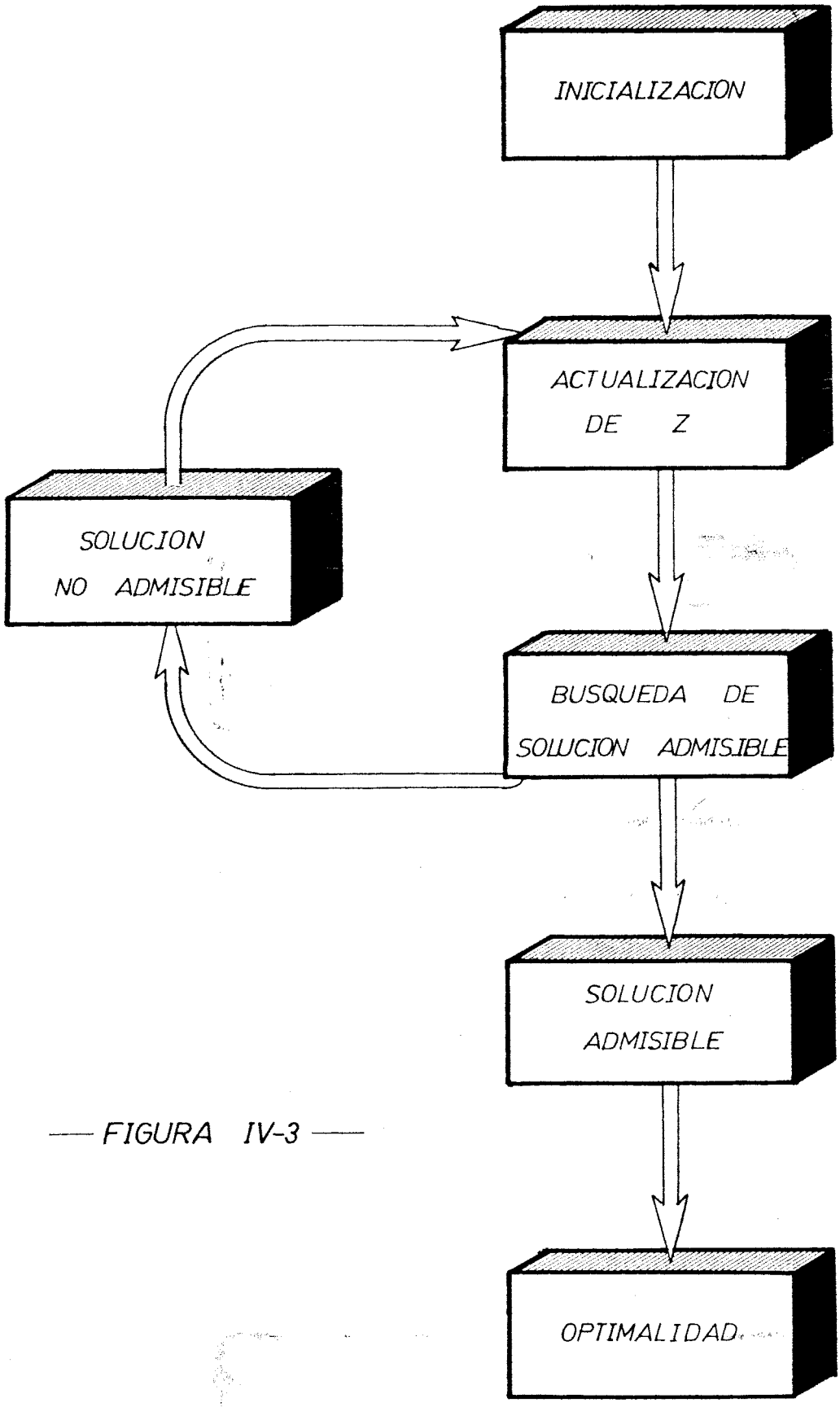

INICIALIZACION
ACTUALIZACION DE Z
SOLUCION ADMISIBLE
BUSQUEDA DE SOLUCION ADMISIBLE
SOLUCION NO ADMISIBLE
OPTIMALIDAD

FIGURA IV-2


INICIALIZACION
ACTUALIZACION DE Z
SOLUCION NO ADMISIBLE
BUSQUEDA DE SOLUCION ADMISIBLE
SOLUCION ADMISIBLE
OPTIMALIDAD

— FIGURA IV-3 —

b) Debido a la no admisibilidad del valor de z (excepto en el óptimo) es necesario calcular los extremos u_j, l_j, hasta que el intervalo no contenga ningún valor entero. Unicamente en el caso que z sea el óptimo será necesario calcular las cotas para las primeras variables.

Para la obtención de la cota superior de z se propone utilizar el sistema resultante después de la eliminación por Fourier-Motzkin de las n-1 primeras variables. El sistema es

$$c_n^n \, x_n \geq P^n \, z - Q^n$$

$$a_n \, x_n \leq b$$

$$x_n \geq 0 \text{ entero}$$

De donde

$$0 \leq x_n \leq [b/a_n]$$

Si $c_n^n \geq 0$ resulta

$$z \leq (c_n^n \, . \, [b/a_n] + Q^n)/P^n$$

Una cota superior será

$$z = [(c_n^n [b/a_n] + Q^n)/P^n]$$

Si $c_n^n < 0$, empleando el valor inferior de x_n, resulta

$$z \leq Q^n/P^n$$

ya que P^n es siempre positivo. Así, una cota superior de z es

$$z = [Q^n/P^n]$$

También puede utilizarse como cota superior el mayor - entero menor que el óptimo del problema knapsack continuo.

Para problemas en los que la solución rápida sea óptima o esté próxima al óptimo, el algoritmo original propuesto por Cabot es más eficiente por tener que realizar la enumeración para muy pocos valores de z. Por otra parte si el óptimo está alejado

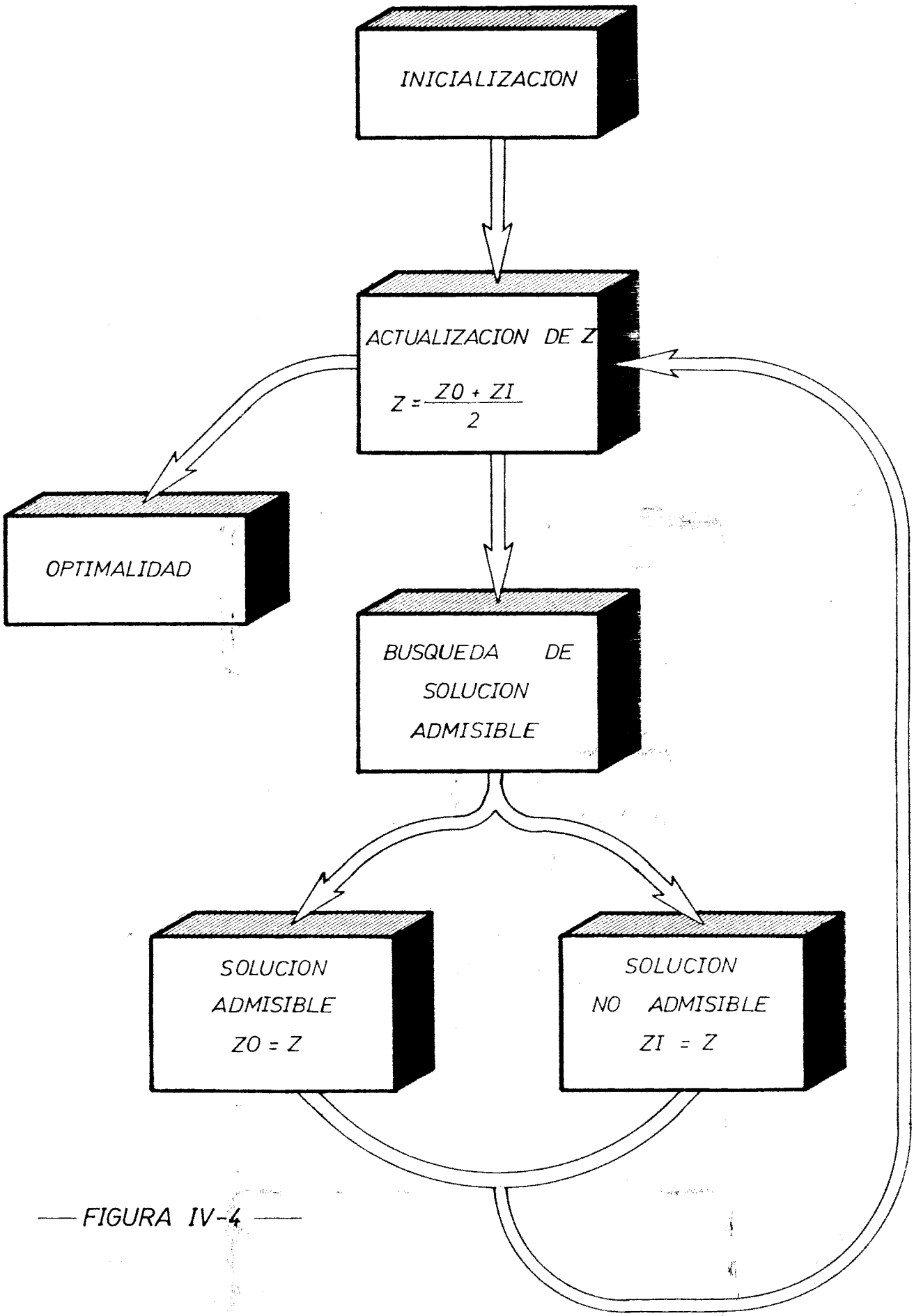
tanto de la cota inferior obtenida por la solución rápida como de la cota superior obtenida por los métodos ya expuestos, la efi--- ciencia de ambos algoritmos, original y modificado, es pequeña.

Para paliar este defecto de pérdida de eficiencia de - ambos algoritmos si la cota, inferior en uno, superior en otro, - está lejos del óptimo, proponemos un criterio que mejora el algoritmo convirtiéndolo en uno de los más eficientes para el problema knapsack (diagrama 4.4). Consiste básicamente en

a) Obtener inicialmente una cota superior no admisible z^S y una cota inferior admisible z^I. Haciendo $z^M = \left[\frac{z^I + z^S}{2} \right]$ se analiza si el sistema tiene solución admisible para z^M.

b) Si tiene solución admisible se hace $z^I = z^M$. Caso - contrario $z^S = z^M$. Si $z^S = z^I + 1$ se ha llegado al óptimo. Y si - no se calcula un nuevo z^M según a) y repitiendo el apartado b).

Estas modificaciones se han incluido en la implementación del algoritmo según se detalla.


INICIALIZACION
ACTUALIZACION DE Z
$Z = \frac{Z0 + ZI}{2}$
OPTIMALIDAD
BUSQUEDA DE SOLUCION ADMISIBLE
SOLUCION ADMISIBLE
Z0 = Z
SOLUCION NO ADMISIBLE
ZI = Z

— FIGURA IV-4 —

Desarrollo del algoritmo

Paso 1. Calcular la solución rápida y el valor de la función objetivo correspondiente z^0. Ir a 2.

Paso 2. Aplicando la técnica de eliminación de F-M calcular C_i^j para $j = 1,\ldots,n$, $i = j,\ldots,N$. Ir a 3.

Paso 3. Hallar P^j y Q^j para $j = 1,\ldots n$. Ir a 4.

Paso 4. Si $C_n^n > 0$ hacer :

$$zI = \left|(|b/a_n|\ C_n^n - Q^n)/P^n\right|$$

Si $C_n^n \leq 0$ hacer

$$zI = \left|-Q^n/P^n\right|$$

Ir a 5.

Paso 5. Si $zI \leq z0$ ir a 6. Caso contrario ir a 7.

Paso 6. Solución óptima es z0. Fin del algoritmo.

Paso 7. Hacer

$$z = < \frac{zI + z0}{2} > .$$ Ir a 8.

Paso 8. Si $C_n^n > 0$ hacer

$XS(N) = b/a_n$

$XI(N) = (P^n * z + Q^n)/C_n^n$

Si $XI(N) < 0$ hacer $XI(N) = 0$

Si $C_n^n \leq 0$ hacer

$XS(N) = (P^n * z + Q^n)/C_n^n$

$XI(N) = 0$

Si $XS(N) > |b/a_n|$ hacer $XS(N) = |b/a_n|$.

Ir a 9.

Paso 9. Si $XS(N) < XI(N)$ ir a 12. Caso contrario ir a 10.

Paso 10. Hacer X(N) = XS(N), I = N - 1. Ir a 11.

Paso 11. Hacer

$$S_1 = \sum_{j=I+1}^{N} a_j x_j \ , \ S_2 = \sum_{j=I+1}^{N} c_j^i x_j$$

Ir a 13.

Paso 12. Si z = zI hacer zI = zI + 1 y z = zI e ir a 5.

Si $z \neq zI$ hacer zI = z e ir a 5.

Paso 13. Si $c_i^i \leq 0$ ir a 15. Caso contrario ir a 14.

Paso 14. Hacer $XS(I) = (b-S_1)/a_i$

$XI(I) = (P^i z + Q^i - S_2)/c_i^i$

Si XI(I) < 0 hacer XI(I) = 0

Ir a 16.

Paso 15. Hacer XI(I) = 0

$XS(I) = (P^i z + Q^i - S_2)/c_i^i$

Si $XS(I) > (b - S_1)/a_i$ hacer $XS(I) = (b-S_1)/a_i$

Ir a 16.

Paso 16. Si XS(I) < XI(I) ir a 19. Caso contrario ir a 17.

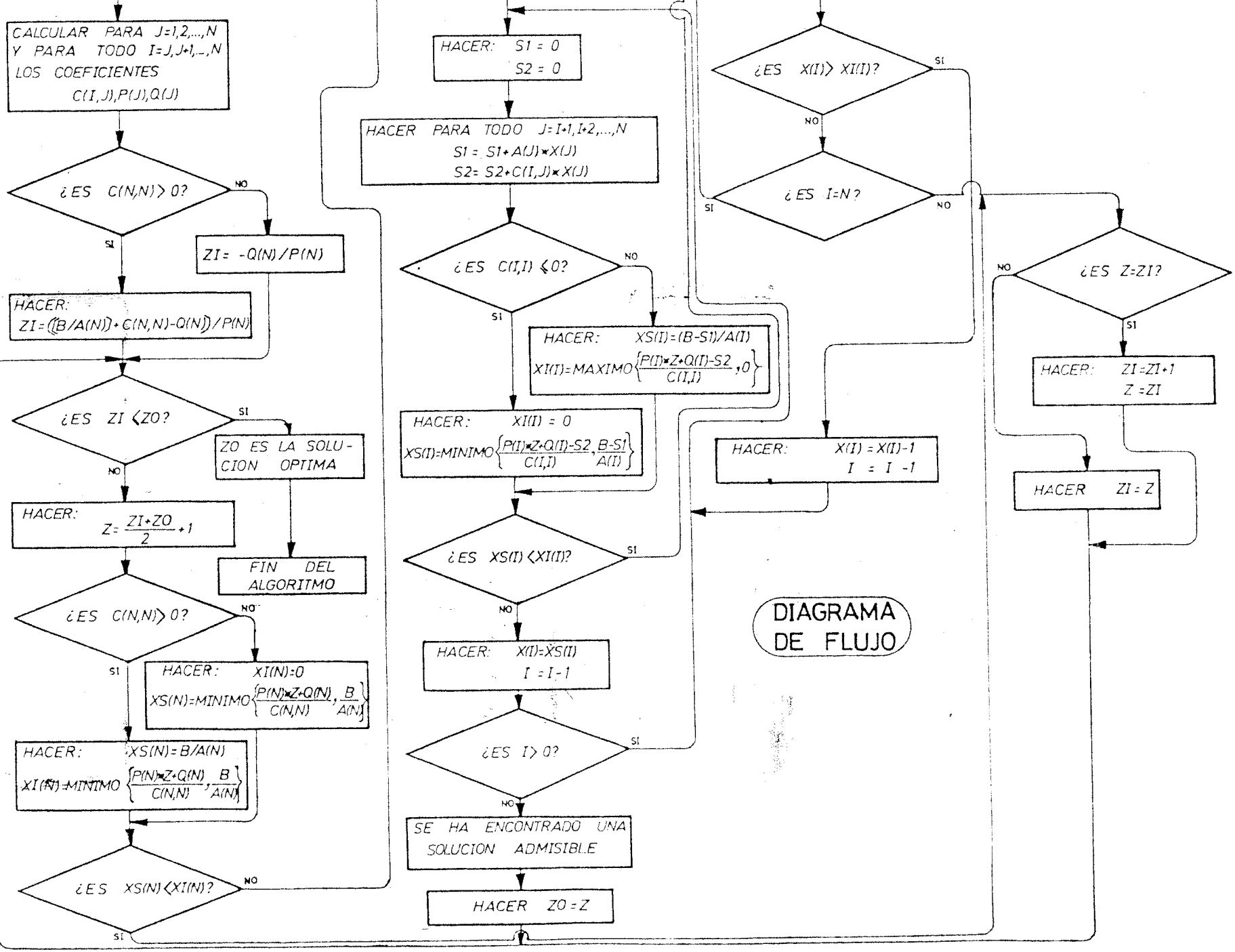
Paso 17. Hacer X(I) = XS(I), I = I-1. Si I > 0 ir a 11. Caso contrario ir a 18.

Paso 18. Se ha encontrado una solución admisible para z. Hacer z0 = z e ir a 5.

Paso 19. Hacer I = I + 1. Si X(I) > XI(I) ir a 21. Caso contrario ir a 20.

Paso 20. Si I = N ir a 12. Caso contrario ir a 19.

Paso 21. Hacer X(I) = X(I)-1, I = I-1. Ir a 11.


CALCULAR PARA J=1,2,...,N
Y PARA TODO I=J,J+1,...,N
LOS COEFICIENTES
C(I,J),P(J),Q(J)
¿ES C(N,N) > 0?
NO
SI
ZI = -Q(N)/P(N)
HACER:
ZI = ((B/A(N)) + C(N,N)-Q(N))/P(N)
¿ES ZI < ZO?
SI
ZO ES LA SOLUCION OPTIMA
NO
HACER:
Z = ZI+ZO/2 + 1
FIN DEL ALGORITMO
¿ES C(N,N) > 0?
NO
SI
HACER: XI(N)=0
XS(N)=MINIMO {P(N)*Z+Q(N)/C(N,N), B/A(N)}
HACER: XS(N)=B/A(N)
XI(N)=MINIMO {P(N)*Z+Q(N)/C(N,N), B/A(N)}
¿ES XS(N) < XI(N)?
NO
SI
HACER: S1 = 0
S2 = 0
HACER PARA TODO J=I+1,I+2,...,N
S1 = S1+A(J)*X(J)
S2 = S2+C(I,J)*X(J)
¿ES C(I,I) < 0?
NO
SI
HACER: XS(I)=(B-S1)/A(I)
XI(I)=MAXIMO {P(I)*Z+Q(I)-S2/C(I,I), 0}
HACER: XI(I) = 0
XS(I)=MINIMO {P(I)*Z+Q(I)-S2/C(I,I), B-S1/A(I)}
¿ES XS(I) < XI(I)?
NO
SI
HACER: X(I)=XS(I)
I = I-1
¿ES I > 0?
NO
SI
SE HA ENCONTRADO UNA SOLUCION ADMISIBLE
HACER ZO = Z
¿ES X(I) > XI(I)?
NO
SI
¿ES I=N?
SI
NO
HACER: X(I) = X(I)-1
I = I -1
¿ES Z=ZI?
NO
SI
HACER: ZI=ZI+1
Z =ZI
HACER ZI = Z
DIAGRAMA DE FLUJO

4.5. Transformación de un Problema Knapsack Particular

Al problema

$$\max \; cx$$
$$ax = b$$
$$x \geq 0 \text{ entero}$$

se le puede dar un enfoque parecido al de la reducción del problema a un problema de minimización sobre un cono pero sin considerar que las variables básicas son las óptimas del problema lineal relajado. Realizamos la partición de c, a, x en $c = (c_B, c_D)$, $a = (a_B, a_D)$, $x^T = (x_B, x_D)$ donde x_B tiene dos coordenadas.

El problema se transforma en

$$\max \; k$$
$$c_B x_B + c_D x_D = k$$
$$a_B x_B + a_D x_D = b$$
$$k, x_B, x_D \geq 0 \text{ enteros}$$

donde B es el conjunto de índices de las variables x_B y D el de - las x_D. Llamando

$$S_B = c_B' a_B + a_B' c_B$$

$$S_D = c_B' a_D + a_B' c_D$$

el problema puede reescribirse, si S_B es regular,

$$\max k$$

$$x_B = S_B^{-1} (b c_B' + k a_B' - S_D x_D)$$

$$x_B, x_D, k \geq 0 \text{ enteros}$$

ó lo que es igual

$$\max k$$

$$S_B^{-1} (b c_B' + k a_B' - S_D x_D) \geq 0$$

$$S_B^{-1} (b c_B' + k a_B' - S_D x_D) \equiv 0 \pmod 1$$

$$k, x_D \geq 0 \text{ entero}$$

Para simplificar la notación, se reordenan las variables de forma que

$$B = \{1,2\} \quad y \quad D = \{3,\ldots,n\}.$$

Entonces

$$S_B = \begin{vmatrix} 2\,a_1\,c_1 & a_1\,c_2 + a_2\,c_1 \\ a_1\,c_2 + a_2\,c_1 & 2\,a_2\,c_2 \end{vmatrix}$$

$$\Delta = |S_B| = -\,(a_1\,c_2 - a_2\,c_1)^2$$

Haciendo $\gamma = a_1\,c_2 - a_2\,c_1$ se tiene

$$S_B^{-1} = \frac{1}{\gamma^2} \begin{vmatrix} -2\,a_2\,c_2 & a_1\,c_2 + a_2\,c_1 \\ a_1\,c_2 + a_2\,c_1 & -2\,a_1\,c_1 \end{vmatrix}$$

Con estos elementos el problema anterior toma la forma

$$\max\ k$$

$$c_2\, b - a_2\, k - \sum_{m \in D} \gamma_m^2\, x_m \geq 0$$

$$-c_1\, b + a_1\, k + \sum_{m \in D} \gamma_m^1\, x_m \geq 0$$

$$c_2\, b - a_2\, k - \sum_{m \in D} \gamma_m^2\, x_m \equiv 0 \pmod{\gamma} \tag{4.3}$$

$$-c_1\, b + a_1\, k + \sum_{m \in D} \gamma_m^1\, x_m \equiv 0 \pmod{\gamma} \tag{4.4}$$

$$k,\ x_m \geq 0 \text{ entero } (m \in D)$$

donde $\gamma_m^1 = c_1\, a_m - a_1\, c_m$ y $\gamma_m^2 = c_2\, a_m - a_2\, c_m$. Nótese que siempre es posible elegir B de forma que S_B^{-1} exista y que γ sea positivo.

Si γ divide a γ_m^1 y a γ_m^2 para todo $m \in D$ las restricciones (4.3) y (4.4) toman la forma

$$\frac{c_2 b}{\gamma} - \frac{a_2 k}{\gamma} - \sum_{m \in D} h_m^2 x_j \equiv 0 \pmod 1$$

$$-\frac{c_1 b}{\gamma} + \frac{a_1 k}{\gamma} + \sum_{m \in D} h_m^1 x_j \equiv 0 \pmod 1$$

Siendo $h_m^2 = \gamma_m^2/\gamma$, $h_m^1 = \gamma_m^1/\gamma$ donde h_m^2 como h_m^1 son enteros. Por tanto las expresiones anteriores son equivalentes a

$$c_2 b - a_2 k \equiv 0 \pmod \gamma \tag{4.5}$$

$$c_1 b + a_1 k \equiv 0 \pmod \gamma \tag{4.6}$$

De este sistema de ecuaciones de congruencia se tiene que $k = t + \dot{\gamma}$ ($\dot{\gamma}$ = multiplo de γ). Llevando esto al problema se obtiene

$$\max \quad k$$

$$c_2 b - a_2 k - \sum_{m \in D} \gamma_m^2 x_m \geq 0$$

$$-c_1 b + a_1 k - \sum_{m \in D} \gamma_m^1 x_m \geq 0 \tag{4.7}$$

$$k = \{t,\ t + \gamma,\ t + 2\gamma, \ldots.\}$$

$$x_m \geq 0 \text{ entero } (m \in D)$$

Mediante un sencillo ejemplo se ilustra la aplicación de este resultado.

Sea el problema

$$\max\ 5x_1 + 4x_2 + 3x_3 + 2x_4 + x_5$$

$$6x_1 + 7x_2 + 8x_3 + 9x_4 + 10x_5 = b$$

$$x_i \geq 0 \text{ entero } (i = 1,\ldots,5)$$

reordenando las variables, se transforma

$$\max\ 2x_1 + 3x_2 + 5x_3 + 4x_4 + x_5$$

$$9x_1 + 8x_2 + 6x_3 + 7x_4 + 10x_5 = b$$

$$x_i \geq 0 \text{ entero } (i = 1,\ldots,5)$$

Para este problema $\gamma = 11$, $\gamma_3^1 = -33$, $\gamma_4^1 = -22$, $\gamma_5^1 = 11$ $\gamma_3^2 = -22$, $\gamma_4^2 = -11$, $\gamma_5^2 = 22$. Se cumple el supuesto de que γ divide a γ_m^1 y γ_m^2 para todo m.

Planteando las ecuaciones (4.5) y (4.6)

$$3b + 8k \equiv 0 \pmod{11}$$

$$-2b + 9k \equiv 0 \pmod{11}$$

Si tomamos por ejemplo b = 19 resulta

$$57 - 8k = \dot{11}$$

$$-38 + 9k = \dot{11}$$

Para que k sea entero ha de cumplirse

$$k = 3 + \dot{11}$$

Por tanto el problema (4.7) tomará la forma

max k

$$\frac{57-8k}{11} + 2x_3 + x_4 - 3x_5 \geq 0$$

$$\frac{-38+9k}{11} - 3x_3 - 2x_4 + x_5 \geq 0$$

$$k = \{3, 14, 25, \ldots.\}$$

$$x_3\ x_4\ x_5 \geq 0 \text{ entero}$$

CAPITULO V

ENUMERACION LEXICOGRAFICA PARA PROBLEMAS LINEALES ENTEROS POSITIVOS

5.1. Introducción

El capítulo está dedicado al estudio de métodos de enumeración implicita para la resolución de problemas lineales enteros en que todos los coeficientes de la función objetivo y las restricciones son positivos. Nos referiremos a problemas lineales enteros positivos. Generalizando el concepto de aportación marginal a la función objetivo en problemas de varias restricciones, para el problema

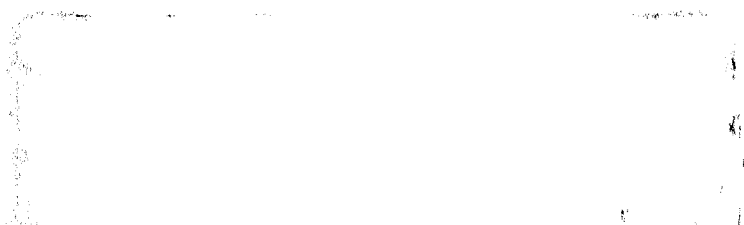
$$
\begin{aligned}
&\max \quad z \\
&\sum_{j=1}^{n} c_j x_j \geq z \\
&\sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i = 1,\ldots,m) \qquad (5.1) \\
&x_j \geq 0 \text{ entero}
\end{aligned}
$$

definimos $\rho_j^i = c_j/a_{ij}$. Supondremos que los coeficientes a_{ij} son tales, que mediante una ordenación adecuada de las variables, se satisface para restricción i, que las aportaciones marginales ρ_j^i son crecientes ($\rho_j^i \leq \rho_{j+1}^i$ para todo j). Como veremos en el capítulo próximo, las hipotesis impuestas al modelo no son tan restrictivas como, a priori, pudiera parecer.

Un primer método generaliza los criterios de terminación, basados en la idea de la extensión fraccionaria de soluciones admisibles para generar cotas superiores, ya empleados por Gilmore y Gomory para el problema knapsack. El segundo extiende el punto de vista de la enumeración lexicográfica de soluciones admisibles con acotaciones sucesivas de la función objetivo, presentado en el capítulo IV para el problema knapsack, al problema de varias restricciones. Siguiendo a los resultados presentados se desarrollan algoritmos que los implementan.

5.2. Terminología y Notación

En el primer método se analiza el problema lineal entero positivo descrito en 5.1. En el método de acotaciones sucesivas se considera que las m restricciones tecnológicas son con signo de desigualdad



$$\max \quad z$$

$$\sum_{j=1}^{n} c_j x_j \geq z$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad (i = 1,\ldots,m) \tag{5.2}$$

$$x_j \geq 0 \text{ entero}$$

Introducimos ahora la notación empleada en este capítulo :

Se dice que el vector (x,z) de n+1 coordenadas es un par admisible para un problema lineal entero positivo si sustituido en el problema correspondiente satisface las restricciones.

Definimos como $x^{o,k}$ el conjunto de vectores enteros $(x_1,\ldots,x_{n-k}, x^o_{n-k+1},\ldots, x^o_n)$ donde las k últimas coordenadas j toman valores fijados x^o_j tales que $\sum_{j=n-k+1}^{n} a_{ij} x^o_j \leq b_i$ para cada i. Para k=n escribiremos x^o ($x^o \equiv x^{o,n}$).

Para el vector x^o definimos :

1. $R_k^i(x^o) \equiv b_i - \sum_{j=n-k+1}^{n} a_{ij}\, x_j^o$

que representa la holgura de la restricción i para el vector $(0,\ldots,0,\ x_{n-k+1}^o,\ldots,x_n^o)$.

2. $E_k^i(x^o) \equiv \sum_{j=n-k+1}^{n} c_j\, x_j^o + \rho_{n-k}^i\, R_k^i(x^o)$

que representa una cota superior de $\sum_{j=1}^{n} c_j\, x_j$ para $x^{o,k}$.

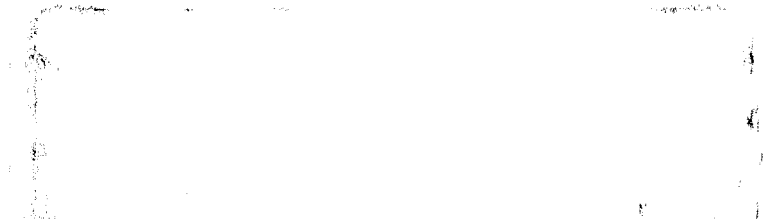
3. $E_k(x^o) = \min_i E_k^i(x^o)$

4. $I_k(x^o) \equiv \max \sum_{j=1}^{n-k} c_j\, x_j + \sum_{j=n-k+1}^{n} c_j\, x_j^o$

$$\sum_{j=1}^{n-k} a_{ij}\, x_j = R_k^i(x^o) \quad (i = 1,\ldots,m)$$

$$x_j \geq 0 \text{ entero} \quad (j = 1,\ldots,n-k).$$

De igual forma que en el capítulo anterior, y dado que las aportaciones marginales son crecientes, utilizamos la ordenación lexicográfica en el sentido decreciente de los índices de las coordenadas.



Finalmente el vector e_k representa el vector de n coordenadas todas nulas excepto la n-k+1 cuyo valor es la unidad.

5.3. Primer Criterio de Eliminación

Si se posee una cota inferior admisible de la función objetivo que mejora a la extensión continua de un cierto vector, la cota es inalcanzable por cualquier extensión entera admisible. Esta idea intuitiva se formaliza en el siguiente resultado.

Proposición 5.1.

Sea z^o una cota inferior admisible del problema lineal (5.1). Si $z^o > E_k(x^o)$ entonces

$$z^o > I_k(x^o)$$

Demostración : Trivial, pues $E_k(x^o) \geq I_k(x^o)$ por definición de ambos.//

Esta relación toma diferentes formas según los vectores de los que se parta para obtener soluciones admisibles. En la enumeración lexicográfica que empleamos, a partir de un cierto vector $x^{o,k}$ (con las k últimas coordenadas fijadas) cuyas extensiones no son óptimas, evitamos explorar todos los vectores comprendidos entre este y el vector $(x^{o,k} - L\ e_k)$, empleando los corolarios que se exponen a continuación.

Corolario 5.1

Dado un vector cualquiera x^o se cumple

$$E_k(x^o) > E_k(x^o - L\ e_k)$$

para $0 < L \leq x^o_k$.

Demostración : Para cada i tenemos

$$E^i_k(x^o - L\ e_k) = E^i_k(x^o) - L\ a_{n-k+1}\ (\rho^i_{n-k+1} - \rho^i_{n-k})$$

Siendo $L\ a_{in-k+1}\ (\rho^i_{n-k+1} - \rho^i_{n-k}) \geq 0$.//

Corolario 5.2

Sea z^o una cota inferior admisible del problema (5.1), tal que $z^o > E_k(x^o)$. Si para algún i

$$z^o > \sum_{n-k}^{n} c_j x_j^o + \rho_{n-k+1}^i R_{k+1}^i(x^o) - a_{n-k+2}(\rho_{n-k+2}^i - \rho_{n-k+1}^i)$$

entonces las extensiones de los vectores x tales que

$$x^{o,k} \leq x \leq (x^{o,k-1} - L\, e_{k-1})$$

no mejoran la cota z^o.

Demostración :

$$z^o > \sum_{n-k}^{n} c_j x_j^o + \rho_{n-k+1}^i(x^o) - a_{n-k+2}(\rho_{n-k+2}^i - \rho_{n-k+1}^i) \text{ |por hipotesis|}$$

$$= E_{k+1}^i(x^o - e_{k+1})$$

$$\geq E_{k+1}(x^o - e_{k+1})$$

$$\geq E_{k+1}(x^o - L\, e_{k+1}) \qquad |x^o_{k+1} \geq L \geq 1|$$

$$\geq I(x^o - L\, e_{k+1}).$$

La enumeración se realiza descendiendo lexicográficamente. En cualquier fase de la enumeración, cuando se esté considerando un vector $x^{o,k}$, todos los vectores lexicográficamente mayores ya han sido analizados. A partir de él se examinan sus extensiones o un vector lexicográficamente menor, según se cumpla la proposición 5.1 o no. Se extiende el vector $x^{o,k}$ a $x^{o,t}$ ---- $(t > k)$ mientras se satisface la proposición: si $t = n$ se ha encontrado una solución admisible que mejora la cota inferior. La extensión se realiza haciendo $x^o_{j+1} = [\min_i \{R^i_j(x^o)/a_{ij}\}]$ secuencialmente para $j = k,\ldots,t-1$. Si la extensión continua, $E_k(x^o)$, no alcanza la cota, se desciende lexicográficamente. En el corolario 5.2 se tiene, mediante una simple comprobación, como descender desde el vector $x^{o,k}$ al vector $(x^{o,k-1} - e_{k+1})$, lexicográficamente menor sin eliminar ninguna solución admisible.

5.4. Primer Algoritmo de Enumeración

El algoritmo se inicia calculando la solución admisible lexicográficamente máxima (solución rápida) y su correspondiente valor de la función objetivo que se emplea como cota inferior. En la enumeración se distinguen dos rutinas bien diferenciadas. Una de extensión del vector considerado que finaliza con una nueva y mejor solución admisible o cuando se comprueba que las extensiones continuas no superan la cota. La otra, de descenso lexicográfico, disminuye el valor de una cierta variable, de acuerdo con los resultados de los corolarios, a partir de la cual se comienza de nuevo la extensión según el proceso descrito. El algoritmo termina cuando en la fase de descenso lexicográfico se alcanza el vector nulo.

Desarrollo del algoritmo.

0. Calcular $RO(I,J) = C(J)/A(I,J)$, $DIRO(I,J) = RO(J+1) - RO(J)$. Calcular la solución admisible lexicográficamente máxima XO, ZO. Hacer X = XO. Ir a 1.

1. Hacer para todo $I = 1,\ldots,M$, $R(I) = \sum_{J=N-M+1}^{N} XO(J) * A(I,J)$, $T(I) = D * XO(N-M+I)$, $Z = ZO - \sum_{J=N-M+1}^{N} C(J) * XO(J)$. Hacer $J = N-M$. Ir a 2.

2. Si $X(J) = 0$ ir a 7. Caso contrario ir a 6.

3. Si para $I = 1,\ldots,M$ algún $ZO - Z \geq A(I,J) * D(I,J) + RO(I,J+1) * R(I)$ ir a 5. Caso contrario ir a 4.

4. Si para $I = 1,\ldots,M$ algún signo $(T(I)) \neq$ signo (D) y $J \geq CI(I)$ ir a 2. Caso contrario ir a 9.

5. Hacer para todo $I = 1,\ldots,M$, $R(I) = R(I) + X(J) * A(I,J)$, $T(I) = T(I) + X(J) * G(I,J)$. Hacer $Z = Z - C(J) * X(J)$, $X(J) = 0$. Ir a 7.

6. Hacer para todo I = 1,...,M, R(I) = R(I) + A(I,J), T(I) = = T(I) + G(i,J). Hacer Z = Z - C(J), X(J) = X(J) - 1. Ir a 3.

7. Si J = 0 ir a 8. Caso contrario hacer J = J-1 e ir a 2.

8. Fin del algoritmo. El óptimo es XO, ZO.

9. Hacer J = J+1. Si J > N-M ir a 14. Caso contrario ir a 10.

10. Si para I = 1,...,M algún ZO $\geq$ Z + R(I) * RO(I,J) ir a 2. Caso contrario ir a 11.

11. Hacer X(J) = mínimo $_{I}$ {R(I)/A(I,J)}. Si para I = 1,...,M algún R(I) = 0 ir a 12. Caso contrario ir a 9.

12. Si para todo I = 1,...,M, R(I) = 0 ir a 13. Caso contrario - ir a 2.

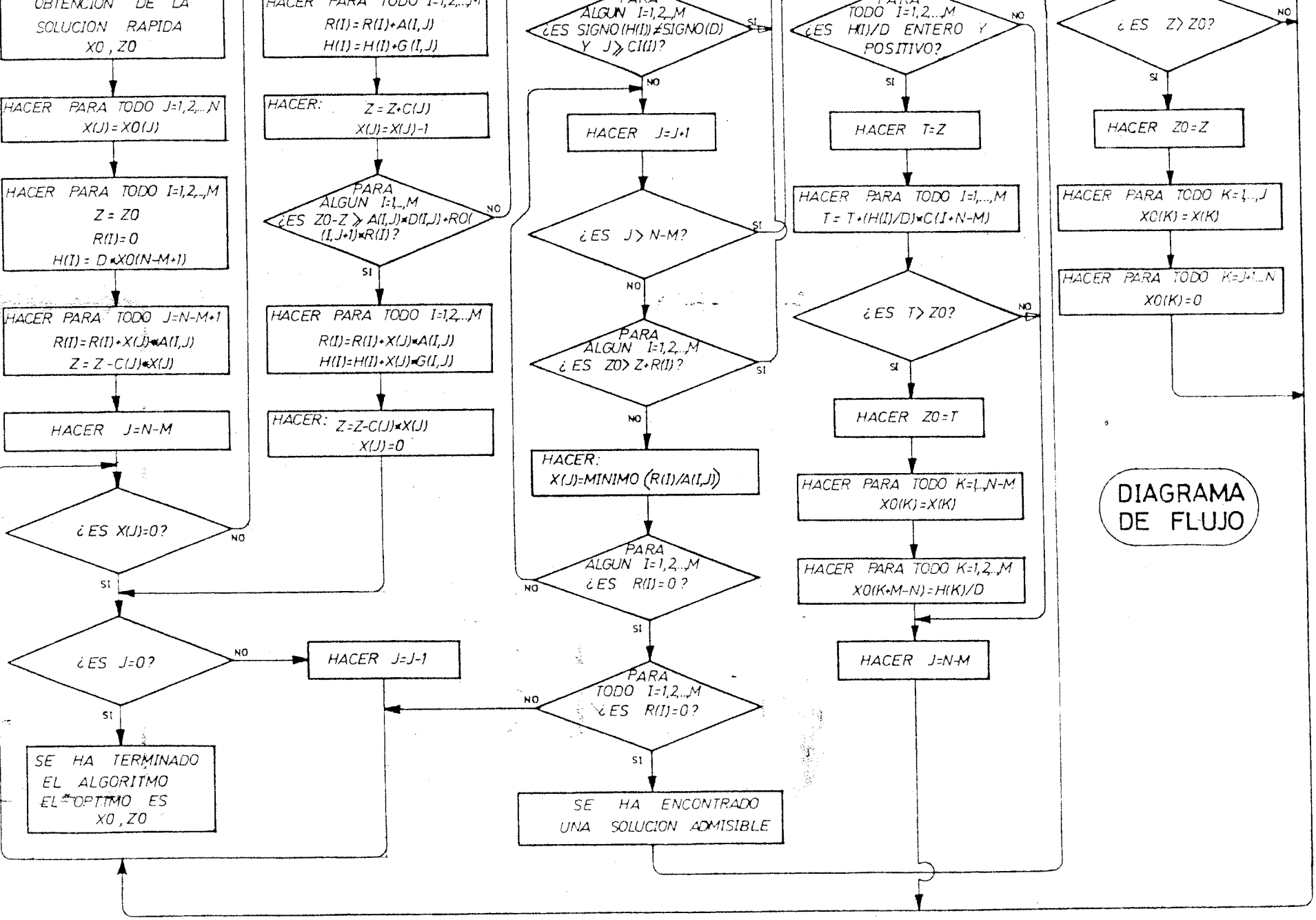
13. Se ha encontrado una solución admisible. Si Z > ZO hacer para todo I = 1,...,J, XO(J) = X(J), para todo I = J+1,...,N, XO(J) = 0, ZO = Z. Ir a 2.

14. Si para todo I = 1,...,M, H(I)/D es entero y positivo ir a - 15. Caso contrario ir a 17.

15. Si $Z + \sum_{I=1}^{M} (H(I)/D) * C(N-M+I) > ZO$ ir a 16. Caso contrario ir a 17.

16. Hacer para todo I = 1,....,M, X(N-M+I) = H(I)/D. Hacer para todo J = 1,....,N, XO(J) = X(J). Hacer ZO = Z + $+ \sum_{K=N-M+1}^{N} X(K) * C(K)$. Ir a 17.

17. Hacer J = N-M. Ir a 2.


OBTENCION DE LA SOLUCION RAPIDA X0 , Z0
HACER PARA TODO J=1,2,...,N X(J) = X0(J)
HACER PARA TODO I=1,2,...,M Z = Z0 R(I) = 0 H(I) = D*X0(N-M+1)
HACER PARA TODO J=N-M+1 R(I) = R(I) + X(J)*A(I,J) Z = Z - C(J)*X(J)
HACER J=N-M
¿ES X(J)=0?
NO
SI
¿ES J=0?
NO
HACER J=J-1
SI
SE HA TERMINADO EL ALGORITMO EL OPTIMO ES X0 , Z0
HACER PARA TODO I=1,2,...,M R(I) = R(I) + A(I,J) H(I) = H(I) + G(I,J)
HACER: Z = Z + C(J) X(J) = X(J) - 1
PARA ALGUN I=1,...,M ¿ES Z0 - Z ≥ A(I,J)*D(I,J) + R0(I,J+1)*R(I)?
NO
SI
HACER PARA TODO I=1,2,...,M R(I) = R(I) + X(J)*A(I,J) H(I) = H(I) + X(J)*G(I,J)
HACER: Z = Z - C(J)*X(J) X(J) = 0
PARA ALGUN I=1,2,...,M ¿ES SIGNO(H(I)) ≠ SIGNO(D) Y J ≥ C(I)?
SI
NO
HACER J=J+1
¿ES J > N-M?
SI
NO
PARA ALGUN I=1,2,...,M ¿ES Z0 > Z + R(I)?
SI
NO
HACER: X(J) = MINIMO (R(I)/A(I,J))
PARA ALGUN I=1,2,...,M ¿ES R(I) = 0?
NO
SI
PARA TODO I=1,2,...,M ¿ES R(I) = 0?
NO
SI
SE HA ENCONTRADO UNA SOLUCION ADMISIBLE
PARA TODO I=1,2,...,M ¿ES H(I)/D ENTERO Y POSITIVO?
NO
SI
HACER T=Z
HACER PARA TODO I=1,...,M T = T + (H(I)/D)*C(I+N-M)
¿ES T > Z0?
NO
SI
HACER Z0=T
HACER PARA TODO K=1,...,N-M X0(K) = X(K)
HACER PARA TODO K=1,2,...,M X0(K+M-N) = H(K)/D
HACER J=N-M
¿ES Z > Z0?
NO
SI
HACER Z0=Z
HACER PARA TODO K=1,...,J X0(K) = X(K)
HACER PARA TODO K=J+1,...,N X0(K) = 0
DIAGRAMA DE FLUJO

5.5. Enumeración Lexicográfica con Acotaciones Sucesivas

Para el problema lineal entero positivo

$$
\begin{aligned}
&\max \quad z \\
&\sum_{j=1}^{n} c_j x_j \geq z \\
&\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad (i = 1,\ldots,m) \qquad (5.3) \\
&x_j \geq 0 \text{ entero } (j = 1,\ldots,n)
\end{aligned}
$$

donde c_j, $a_{ij} > 0$ $(i = 1,\ldots,m)$, $(j = 1,\ldots,n)$, $\rho_k^i \leq \rho_j^i$ $(i = 1,\ldots,m)$ $(1 \leq k \leq j \leq n)$, definimos la secuencia de problemas -- {P(K)}

$$\max \quad z$$

$$\sum_{j=k}^{n} (a_{ik-1}\, c_j - c_{k-1}\, a_{ij}) x_j \geq z\, a_{ik-1} - b_i\, c_{k-1}$$

$$\sum_{j=k}^{n} a_{ij}\, x_j \leq b_i$$

$$x_j \geq 0 \text{ entero} \quad (j = k,\ldots,n)$$

$$x_j \text{ libre} \quad (j = 1,\ldots,k-1)$$

para $i = 1,\ldots,m$. Imponiendo $c_o = 0$ y $a_{io} = 1$ P(1) coincide con el problema original (5.3).

Definimos la función $H_k^i(x,z)$ que representa las holguras de cada una de las m primeras restricciones escritas como

$$H_k^i(x,z) \equiv \sum_{j=k}^{n} (\rho_j^i - \rho_{k-1}^i)\, a_{ij}\, x_j + b_i\, \rho_{k-1}^i - z \geq 0$$

y la función $R_k^i(x)$ las correspondientes al ampleo de los recursos b_i

$$R_k^i(x) \equiv b_i - \sum_{j=k}^{n} a_{ij}\, x_j \geq 0$$

Claramente, la admisibilidad del par (x^o,z^o) para el problema P(k) es equivalente, por las definiciones a que $H_k^i(x^o,z^o) \geq 0$ y $R_k^i(x^o) \geq 0$, para todo i.

El problema lineal entero positivo (5.3) lo resolveremos mediante el análisis de la relación entre los n problemas de la secuencia {P(k)}.

Teorema 5.1.

El problema P(p) es una relajación del problema P(k) si $1 \leq k \leq p \leq n$.

Demostración :

Sea (x^o,z^o) un par admisible para P(k). Con respecto a las m primeras restricciones, se tiene

$$\sum_{j=k+1}^{n} (c_j - \rho_k^i a_{ij})\, x_j^o + b_i\, \rho_k^i$$

$$= \sum_{j=k}^{n} (c_j - \rho_{k-1}^i a_{ij}) x_j^o + b_i\, \rho_{k-1}^i + (\rho_k^i - \rho_{k-1}^i) R_k^i(x^o)$$

$$\geq z^o + (\rho_k^i - \rho_{k-1}^i).R_k^i(x^o) \qquad |(x^o,z^o) \text{ admisible en } P(k)|$$

$$\geq z^o \qquad |(\rho_k^i - \rho_{k-1}^i)R_k^i(x^o) \geq 0|$$

Las otras restricciones se satisfacen por ser (x^o,z^o) admisible para P(k).//

Corolario 5.3.

Si (x^o,z^o) es la solución óptima de P(k) entonces z^o - es una cota superior de P(p) para p = 1,...k-1.

Teorema 5.2.

Sea (x^o,z^o) admisible para el problema P(k). Entonces

1. Para $1 \leq k \leq p \leq n$, $i = 1,\ldots,m$

$$H_k^i(x^o,z^o) \leq H_p^i(x^o,z^o)$$

$$R_k^i(x^o) \leq R_p^i(x^o)$$

2. Si a) $R_k^i(x^o) = 0$ para $i \in I_1$

b) $R_k^i(x^o) > 0$ y $\rho_{k-1}^i \leq H_k^i(\bar{x},z^o)/R_k^i(x^o)$ para $i \in I_2$

entonces, el par $(\bar{x},\bar{z})$ es admisible para P(1) con $\bar{x} = (0,\ldots,0, x_k^o,\ldots,x_n^o)$ y $\bar{z} = z^o + \min_i |H_1^i(\bar{x},z^o)|$ mientras que $(\bar{x},\bar{z}+1)$ no lo es.

Demostración :

1. $H_{k+1}^i(x^o,z^o) - H_k^i(x^o,z^o) = (\rho_k^i - \rho_{k-1}^i)\, R_k^i(x^o)$

$$\geq 0 \quad , \quad y$$

$$R_{k-1}^i(x^o) - R_k^i(x^o) = a_{ik+1} \cdot x_{k+1}^o$$

$$\geq 0.$$

2. Hemos de mostrar que $H_1^i(\bar{x},\bar{z})$ y $R_1^i(\bar{x})$ son no negativos

Para el par $(\bar{x},z^o)$ se cumple, para cualquier

$$R_1^i(\bar{x}) = R_2^i(\bar{x}) = \ldots = R_k^i(\bar{x}) = R_k^i(x^o) \geq 0 \qquad y$$

$$H_1^i(\bar{x},z^o) = H_k^i(\bar{x},z^o) - \rho_{k-1}^i \, R_k^i(\bar{x})$$

$$\geq 0 \qquad \text{| según a) y b)|}$$

para todo i. Además

$$H_1^i(\bar{x},\bar{z}) = H_1^i(\bar{x},z^o) - (z^o - \bar{z}) \quad \text{|definición de } H_1^i(x,z)\text{|}$$

$$= H_1^i(\bar{x},z^o) - \min_i |H_1^i(\bar{x},z^o)| \quad \left|\bar{z} = z^o + \min_i |H_1^i(\bar{x},z^o)|\right|$$

$$\geq 0.$$

siendo $H_1^i(\bar{x},\bar{z}) < 1$ para al menos una restricción i.//

Corolario 5.4.

Sea x^o,z^o admisible para P(k). Si $R_k^i(x^o) = 0$ para todo i, entonces $(\bar{x},\bar{z})$ es admisible para P(1), siendo $\bar{x} = (0,\ldots 0,x_k^o,\ldots,x_n^o)$ y $\bar{z} = z^o + \min_i |H_k^i(x^o,z^o)|$.

Corolario 5.5.

Sea (x^o,z^o) admisible para P(k). El par (x^o,z^o) es admisible para P(k-1) si y solo si satisface

$$\min_i \{R_k^i(x^o)/a_{ik-1}\} \geq x_{k-1}^o \geq \max_i \{R_k^i(x^o)/a_{ik-1} - H_k^i(x^o,z^o)/$$

$$/a_{ik-1}\,(\rho_{k-1}^i - \rho_{k-2}^i)\}$$

Demostración :

Por una parte

$$R_{k-1}^i(x^o) = R_k^i(x^o) - a_{ik-1}\,x_{k-1}^o$$

≥ 0, de donde sigue la primera desigualdad.

Además,

$$H^i_{k-1}(x^o,z^o) = H^i_k(x^o,z^o) - (\rho^i_{k-1} - \rho^i_{k-2})(R^i_{k-1}(x^o) - a_{ik-1}\, x^o_{k-1})$$

$$\geq 0, \quad \text{luego para todo } i$$

$$x^o_{k-1} \geq R^i_k(x^o)/a_{ik-1} - H^i_k(x^o,z^o)/a_{ik-1}(\rho^i_{k-1} - \rho^i_{k-2})$$

con lo que se obtiene la segunda acotación.//

Proposición 5.2.

Sea $x^o = (x^o_1,\ldots,x^o_p, 0,\ldots,0, x^o_k,..,x^o_n)$ y (x^o,z^o) admisible para P(k). El par (x^o,z^o) es admisible para P(p + 1) si y solo si

$$\rho^i_p \geq \rho^i_{k-1} - H^i_k(x^o,z^o)/R^i_k(x^o)$$

para todo i.

Demostración :

Obviamente para cada i

$$R^i_{p+1}(x^o) = R^i_k(x^o) \geq 0$$

Hemos de mostrar que $H^i_{p+1}(x^o,z^o) \geq 0$.

$$H^i_k(x^o,z^o) - H^i_{p+1}(x^o,z^o) = \sum_{j=p+1}^{k-1} (\rho^i_j - \rho^i_{j-1}) R^i_j(x^o) \qquad |\text{según 5.}|$$

$$= R^i_k(x^o) \sum_{j=p+1}^{k-1} (\rho^i_j - \rho^i_{j-1}) \quad |R^i_j(x^o)=R^i_k(x^o) \text{ para } j=p+1,\ldots,k|$$

$$= R^i_k(x^o)\,(\rho^i_{k-1} - \rho^i_p) \qquad |\text{suma telescopica}|$$

Luego, para todo i

$$H^i_{p+1}(x^o,z^o) = H^i_k(x^o,z^o) - R^i_k(x^o)(\rho^i_{k-1} - \rho^i_p)$$

de donde se deduce el resultado.//

Proposición 5.3.

Sea x^L la solución admisible lexicográficamente máxima de P(1). Entonces su óptimo z* satisface

$$z^* \leq \min_i \{(\rho_n^i - \rho_{n-1}^i)a_{in}\, x_n^L + b_i \cdot \rho_{n-1}^i\}$$

Demostración :

Cualquier par (x,z) admisible para P(1) satisface

$H_n^i(x,z) \geq 0$. Luego

$$z \leq (\rho_n^i - \rho_{n-1}^i)a_{in}\, x_n + b_i\, \rho_{n-1}^i$$

$$\leq (\rho_n^i - \rho_{n-1}^i)a_{in}\, x_n^L + b_i\, \rho_{n-1}^i$$

para todo i. De donde resulta la proposición.//

El desarrollo que se ha planteado en esta sección es - una generalización, para el problema con varias restricciones, --

del punto de vista empleado en el analisis del problema knapsack. Los comentarios allí efectuados son válidos para este caso. Si -- bien los resultados son más complejos con varias restricciones, - se pueden reducir las enumeraciones al ser las condiciones de admisibilidad de las extensiones más selectivas.

5.6. Segundo Algoritmo de Enumeración

Describimos ahora un algoritmo basado en los resulta-- dos de la sección anterior. Se parte de una cota inferior admisible obtenida a partir de la solución lexicográficamente máxima x^L y una cota superior calculada según la proposición 5.2. Nótese -- que el intervalo máximo entre ambas cotas nunca será superior a

$$\min_i \rho_{n-1}^i \, R_n^i (x^L)$$

El algoritmo divide el intervalo entre ambas cotas a - la mitad en cada iteración. Por tanto una cota, a priori, sobre - el número de iteraciones que realiza el algoritmo viene dada por el menor número N tal que

$$\min_i \rho_{n-1}^i R_n^i (x^L)/2^N \leq 1$$

A continuación se expone la lista de operaciones del algoritmo y su diagrama de flujo.

Desarrollo del algoritmo.

0. Calcular RO(I,J) = C(J)/A(I,J), DIRO(I,J) = RO(I,J) - RO(I,J--1). Calcular la solución admisible lexicográficamente máxima X), ZO. Hacer ZI = mínimo $_I$ {DIRO(I,1) * A(I,1) * XO(1) + B(I) * RO(I,2) + 1}^{I}. Calcular para XO, ZO los valores de H(I,J), R(I). Hacer X = XO y Z = ZO. Ir a 1.

1. Hacer CZ = (ZI + ZO)/2 - Z, Z = (ZI + ZO)/2. Si Z > Z) ir a - 2. Caso contrario ir a 15.

2. Hacer para todo I,J RV(I) = R(I), HV(I,J) = H(I,J). Ir a 3.

3. Hacer para todo I,J, H(I,J) = H(I,J) - CZ. Hacer J = N+1. Ir a 4.

4. Hacer J = J-1. Ir a 5.

5. Si X(J) > 0 ir a 6. Caso contrario ir a 16.

6. Si para I = 1,...,M algún H(I,J) < 0 ir a 19. Caso contrario hacer X(J) = X(J) - 1, hacer para todo I = 1,....,M, R(I) = = R(I) + A(I,J), H(I,J) = H(I,J) - DIRO(I,J) * A(I,J). Si para I = 1,...,M algún H(I,J) < 0 ir a 19. Caso contrario hacer para todo I = 1,....,M, ROM(I) = RO(I, J+1) - H(I,J)/R(I), - IND = J. Ir a 7.

7. Si J $\geq$ N ir a 20. Caso contrario ir a 8.

8. Hacer J = J+1, X(J) = mínimo $_{I}\{R(I)/A(I,J)\}$. Hacer para todo - I = 1,....,M, H(I,J) = H(I,IND) - R(I) * (RO(I,IND + 1) - RO(I, J+1)) + DIRO(I,J) * A(I,J) * X(J). Si para I = 1,..., M algún H(I,J) < 0 ir a 21. Caso contrario, hacer para todo -- I = 1,...,N, R(I) = R(I) - A(I,J) * X(J), IND = J. Si J < N - ir a 9. Caso contrario ir a 20.

9. Hacer K igual al número de R(I) que son cero para I = 1,...,M Si K = 0 ir a 10. Caso contrario ir a 11.

10. Hacer para todo I = 1,...,M, ROM(I) = RO(I,J+1) - H(I,J)/ /R(I). Ir a 7.

11. Si K < M ir a 22. Caso contrario ir a 12.

12. Se ha encontrado una solución admisible. Hacer CH = $\min_I \{H(I,J)\}$. Ir a 13.

13. Hacer para todo L = 1,...,N, XO(L) = X(L). Ir a 14.

14. Hacer ZO = Z + CH. Hacer para todo I,J, H(I,J) = H(I,J) - - CH. Ir a 1.

15. Fin del algoritmo. El óptimo es XO,ZO.

16. Si J = 1 ir a 17. Caso contrario ir a 4.

17. Hacer ZI = Z, Z = (ZI + ZO)/2. Si Z = ZO ir a 15. Caso contrario ir a 18.

18. Hacer para todo I,J, X(J) = XO(J), R(I) = RV(I), H(I,J) = = HV(I,J). Hacer CZ = Z - ZO. Ir a 3.

19. Hacer para todo I = 1,...,M, R(I) = R(I) + X(J) * A(I,J). Hacer X(J) = 0. Ir a 16.

20. Si para todo I = 2,...,M, R(I) $\geq$ LANDA(I) * R(1) ir a 12. Caso contrario hacer J = N e ir a 5.

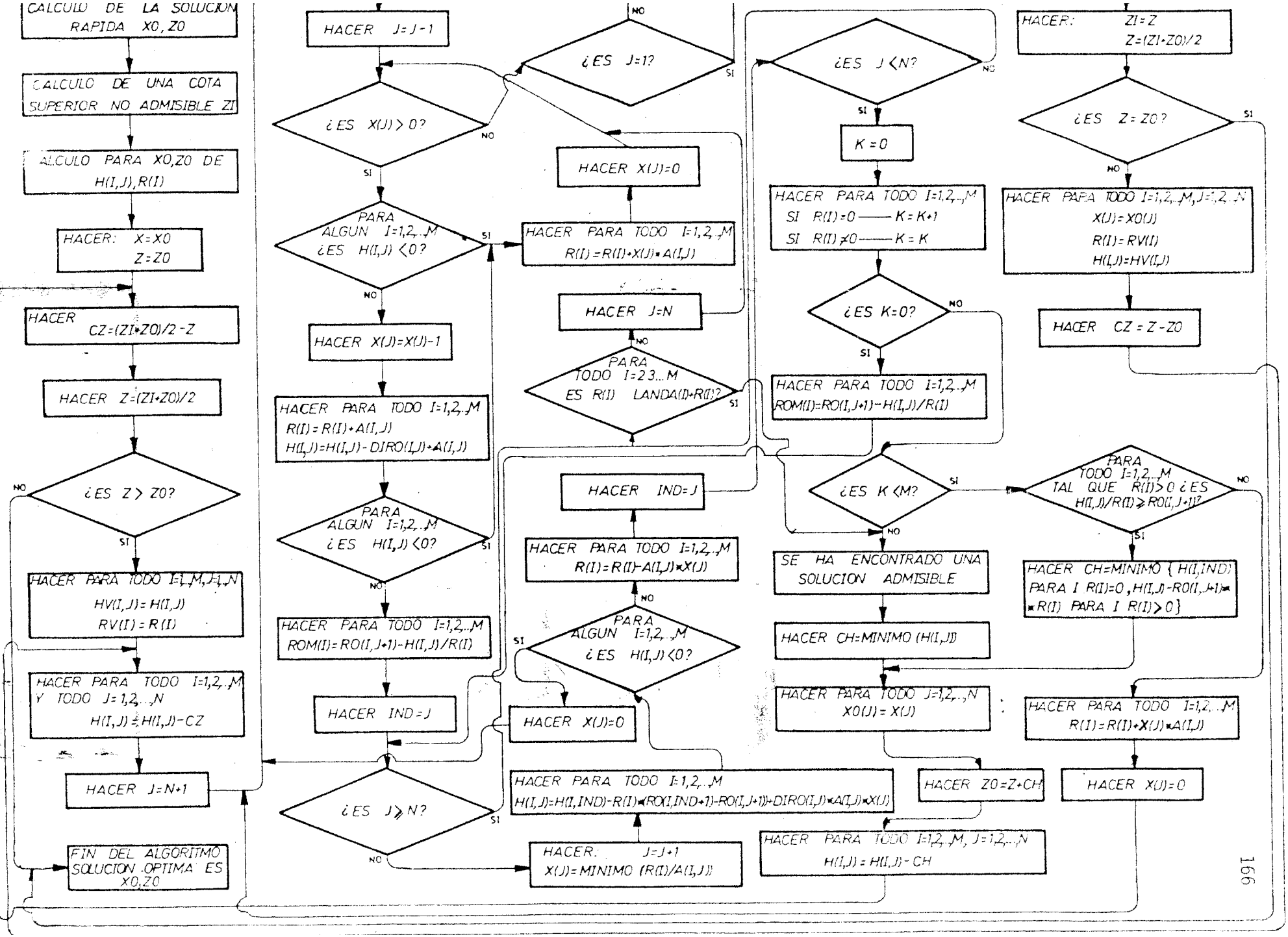
21. Hacer X(J) = 0 . Ir a 4.

22. Si para todo I = 2,...,M, R(I) $\geq$ LANDA(I) * R(1) ir a 23. Caso contrario ir a 5.

23. Si para todo I = 1,...,M tal que R(I) > 0 es H(I,J)/R(I) $\geq$ $\geq$ RO(I,J+1) ir a 24. Caso contrario ir a 25.

24. Hacer CH = mínimo {H(I,IND) para I tal que R(I) = 0, H(I,J)- I - RO(I,J+1) * R(I) para I tal que R(I) > 0}. Ir a 13.

25. Hacer para todo I = 1,...,M, R(I) = R(I) + X(J) * A(I,J). Hacer X(J) = 0. Ir a 4.


CALCULO DE LA SOLUCION RAPIDA X0, Z0
CALCULO DE UNA COTA SUPERIOR NO ADMISIBLE ZI
CALCULO PARA X0,Z0 DE H(I,J),R(I)
HACER: X=X0 Z=Z0
HACER CZ=(ZI+Z0)/2 -Z
HACER Z=(ZI+Z0)/2
¿ES Z > Z0?
SI
NO
HACER PARA TODO I=1,M,J=1,N HV(I,J)=H(I,J) RV(I)=R(I)
HACER PARA TODO I=1,2,..M Y TODO J=1,2,..N H(I,J)=H(I,J)-CZ
HACER J=N+1
FIN DEL ALGORITMO SOLUCION OPTIMA ES X0,Z0
HACER J=J-1
¿ES X(J) > 0?
HACER X(J)=0
¿ES J=1?
PARA ALGUN I=1,2,..M ¿ES H(I,J) <0?
HACER PARA TODO I=1,2,..M R(I)=R(I)+X(J)*A(I,J)
HACER J=N
PARA TODO I=2 3...M ES R(I) LANDA(I)+R(I)?
HACER X(J)=X(J)-1
HACER PARA TODO I=1,2,..M R(I)=R(I)+A(I,J) H(I,J)=H(I,J)-DIRO(I,J)*A(I,J)
PARA ALGUN I=1,2,..M ¿ES H(I,J) <0?
HACER PARA TODO I=1,2,..M ROM(I)=RO(I,J+1)-H(I,J)/R(I)
HACER IND=J
¿ES J ≥ N?
HACER: J=J+1 X(J)=MINIMO (R(I)/A(I,J))
HACER IND=J
HACER PARA TODO I=1,2,..M R(I)=R(I)-A(I,J)*X(J)
PARA ALGUN I=1,2,..M ¿ES H(I,J) <0?
HACER X(J)=0
HACER PARA TODO I=1,2,..M H(I,J)=H(I,IND)-R(I)*(RO(I,IND+1)-RO(I,J+1))+DIRO(I,J)*A(I,J)*X(J)
¿ES J <N?
K=0
HACER PARA TODO I=1,2,..M SI R(I)=0 —— K=K+1 SI R(I)≠0 —— K=K
¿ES K=0?
HACER PARA TODO I=1,2,..M ROM(I)=RO(I,J+1)-H(I,J)/R(I)
¿ES K <M?
SE HA ENCONTRADO UNA SOLUCION ADMISIBLE
HACER CH=MINIMO (H(I,J))
HACER PARA TODO J=1,2,..N X0(J)=X(J)
HACER Z0=Z+CH
HACER PARA TODO I=1,2,..M, J=1,2,..N H(I,J)=H(I,J)-CH
HACER: ZI=Z Z=(ZI+Z0)/2
¿ES Z=Z0?
HACER PARA TODO I=1,2,..M,J=1,2..N X(J)=X0(J) R(I)=RV(I) H(I,J)=HV(I,J)
HACER CZ=Z-Z0
PARA TODO I=1,2,..M TAL QUE R(I)>0 ¿ES H(I,J)/R(I) ≥ RO(I,J+1)?
HACER CH=MINIMO {H(I,IND) PARA I R(I)=0, H(I,J)-RO(I,J+1)* *R(I) PARA I R(I)>0}
HACER PARA TODO I=1,2,..M R(I)=R(I)+X(J)*A(I,J)
HACER X(J)=0

CAPITULO VI

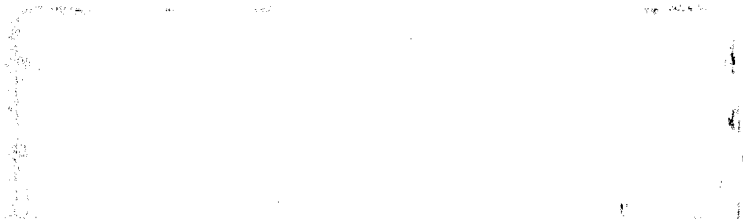
EXTENSION DE LOS METODOS DE ENUMERACION LEXICOGRAFICA A PROBLEMAS ENTEROS.

6.1. Introducción

En el capítulo anterior se han desarrollado varios resultados que nos han permitido implementar algoritmos de enumeración lexicográfica para problemas lineales enteros de características muy específicas. En este capítulo, se expone como transformar un problema lineal entero acotado cualquiera en positivo. Asi mismo y dado que los algoritmos vistos parten de la solución lexicográficamente máxima se presenta un método para obtenerla.

6.2. Transformación de un Problema Lineal Entero en Positivo

Sea el problema lineal entero

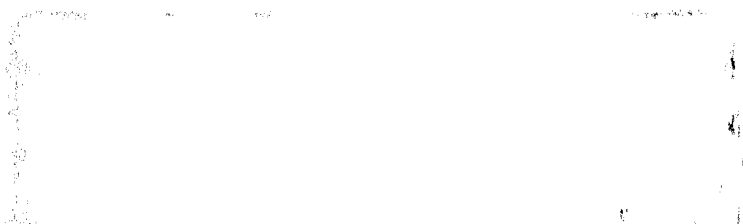


$$\begin{aligned} &\max \quad cx \\ (6.1) \qquad &Ax = b \\ &x \geq 0 \text{ entero} \end{aligned}$$

donde $c = (c_j)$, $b = (b_i)$, $A = (a_{ij})$ y $x = (x_j)$ para $i = 1,\ldots,m$, $j = 1,\ldots,n$, y tal que la región descrita por las restricciones es acotada, es inmediato que se le puede añadir a este problema una nueva restricción en la que intervengan todas las variables $x_1,\ldots,x_n$ sin afectar al conjunto de sus soluciones admisibles. Bastaría para ello resolver el problema lineal continuo

$$\begin{aligned} &\max \quad px \\ (6.2) \qquad &Ax = b \\ &x \geq 0 \end{aligned}$$

donde p es el vector de coeficientes positivos de esta nueva restricción. Siendo x^* el óptimo de (6.2), al añadir la restricción $px \leq px^*$ a (6.1) no se modifica su conjunto de soluciones admisibles. Esto equivale a elegir arbitrariamente un hiperplano px, --



con todos los coeficientes p_j positivos, y buscar su intersección x* con la región continua admisible de (6.1).

Si los coeficientes de la función objetivo $c_1,\ldots,c_n$ - son todos positivos, la inclusión de la restricción superflua -- $px \leq px^*$ permite que, mediante combinaciones lineales de cada una de las restricciones de (6.1) con ella, obtengamos una formula--- ción equivalente del problema original como un problema lineal entero positivo. En efecto, el problema (6.1) es equivalente a

$$\begin{aligned} &\max \quad cx \\ &\overline{A}x = b \\ &x \geq 0 \text{ entero} \end{aligned} \tag{6.3}$$

donde $\overline{a}_{ij} = a_{ij} + \lambda_i p_j$. Eligiendo λ_i de forma que

$$\lambda_i \geq \max_j \left\{ \frac{c_j a_{ij-1} + c_{j-1} a_{ij}}{c_j p_{j-1} - c_{j-1} p_j} , - \frac{a_{ij}}{p_j} , - \frac{a_{ij-1}}{p_{j-1}} \right\}$$

el problema (6.3) es entero positivo con $\rho_j^i \leq \rho_{j+1}^i$ para todo i,j.

Finalmente, si algún coeficiente c_j de la función objetivo es negativo, basta con realizar previamente un simple cambio de variable, $y_j = u_j - x_j$, donde u_j es una cota superior de la variable x_j.

6.3. Obtención de Solución Lexicográficamente Máxima

Para el problema knapsack se define como solución rápida a la solución admisible lexicográficamente máxima para una determinada ordenación de las variables. Extendiendo esta definición, denominamos solución rápida de un problema lineal entero a la solución admisible lexicográficamente máxima con respecto a una ordenación específica, que puede ser cualquiera,de las variables. Es evidente que para cada una de las n! posibles ordenaciones de las variables existirá una solución rápida.

Proposición 6.1.

Sea el problema lineal entero

$$\max \quad cx$$

$$(6.4) \qquad \sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i = 1,\ldots,m)$$

$$x \geq 0 \text{ entero}$$

donde $a_{ij} > o$ para cada i,j, y x^o una solución cualquiera admisible.

1. Si x^p es la solución rápida del problema de una restricción

$$\max \quad cx$$

$$(6.5) \qquad \sum_{j=1}^{n} a_{pj} x_j = b_p$$

$$x_j \geq 0 \text{ entero}$$

para p fijo ($1 \leq p \leq m$), entonces $x^o \leq x^p$.

2. Si además x^k es la solución rápida de

$$\max \quad cx$$

$$(6.6) \qquad \sum_{j=1}^{n} a_{kj} x_j = b_k$$

$$x \geq 0$$

$$x \leq x^p$$

entonces $x^o \leq x^k$.

Demostración :

1. Obvio debido a que (6.5) es una relajación de (6.4).

2. Dado que el problema (6.6) es una relajación de

$$\max \quad cx$$

$$\sum_{j=1}^{n} a_{kj} x_j = b_k$$

$$\sum_{j=1}^{n} a_{pj} x_j = b_p$$

$$x_j \geq 0 \text{ entero}$$

que a su vez lo es de (6.4).//

Para el conjunto de vectores $x^{o,k} = (x_1, \ldots, x^o_{n-k+1}, \ldots, x^o_n)$ mantenemos la definición

$$R^i_k(x^o) = b_i - \sum_{j=n-k+1}^{n} a_{ij}\, x^o_j$$

Preposición 6.2.

Sea x^o una solución admisible para (6.4), entonces :

1. $R^i_k(x^o) \geq R^i_{k-1}(x^o) \geq 0$ para todo k,i

2. Para todo k

$$x^o_{n-k} \leq \min_i |R^i_k(x^o)/a_{ik}|$$

Demostración :

Ambos resultados se siguen de la definición de $R^i_k(x^o)$ y de que $a_{ij} > 0$.//

Llamando B a la matriz cuadrada

$$B \equiv \begin{vmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mm} \end{vmatrix}$$

y D a su determinante, si B es regular

$$B^{-1} = \frac{1}{D} \begin{vmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & & \vdots \\ y_{m1} & \cdots & y_{mm} \end{vmatrix}$$

donde y_{ij} son enteros. Definimos

$$g_{ij} = \sum_{k=1}^{m} y_{ik}\, a_{kj}$$

y, dado $x^{o,k}$, la función

$$T_k^i(x^o) = \sum_{j=1}^{m} R_k^j(x^o) \cdot y_{ij}$$

Por sgn(a) representamos el signo del número real a.

Preposición 6.3.

Sea el conjunto de vectores enteros $x^{o,n-m}$, tal que - $R^i_{n-m}(x^o) \geq 0$ para todo i, e incluya una cota superior lexicográfica de cualquier vector admisible para (6.4). Entonces, la solución - rápida pertenece a $x^{o,n-m}$ si y solo si para todo i :

1. $T^i_{n-m}(x^o)$ es divisible por D, y

2. $\text{sgn}\ (T^{i}_{n-m}(x^{o})) \equiv \text{sgn}\ (D)$

Demostración :

Dado $x^{o,n-m}$, están fijadas todas las coordenadas de -- los vectores pertenecientes a este conjunto salvo las m primeras, que han de ser enteras y positivas. Las m primeras coordenadas -- del vector x^{o} (pertenecen a $x^{o,n-m}$) han de satisfacer

$$x^{o}_{j} = T^{j}_{n-m}(x^{o})/D \qquad (j = 1,\ldots,m)$$

para que $Ax^{o} = b$.

Luego $T^{j}_{n-m}(x^{o})$ ha de ser divisible por D y de su mismo signo.//

Corolario 6.1.

Sea $x^{o,k}$ ($k<n-m$) un conjunto de vectores enteros que - incluye una cota superior lexicográfica de (6.4). Si para algún i

1. sgn $(T^i_k(x^o)) \neq$ sgn (D), y

2. para todo $j = m+1,..,n-k+1$, es $g_{ij} = 0$ ó $sgn(g_{ij}) \equiv sgn(D)$

entonces el vector $(0,...,0, x^o_{n-k+1},...,x^o_n)$ es una cota superior lexicográfica no admisible.

Demostración :

Por definición, para $p > m$

$$T^i_p(x^o) = \sum_{j=1}^{m} R^i_p(x^o) \cdot y_{ij}$$

$$= \sum_{j=1}^{m} b_j y_{ij} - \sum_{n-p+1}^{n} g_{ij} x^o_j \quad \text{|definición de } R^i_p(x^o) \text{ y } g_{ij}\text{|}$$

Luego,

$$T^i_{j=m}(x^o) = T^i_k(x^o) - \sum_{j=m+1}^{n-k+1} g_{ij} x^o_j$$

Vease que $(-\sum_{j=m+1}^{n-k+1} g_{ij}\, x_j^o)$ es de signo contrario a D debido a las hipotesis en 2 y a que $x_j^o \geq 0$. Considerando además la hipotesis - en 1, resulta que el signo de $T_k^i(x^o) - \sum_{j=m+1}^{n-k} g_{ij}\, x_j^o$ es distinto al de D. Aplicando la proposición 6.3 se obtiene el resultado.//

Desarrollo del Algoritmo

0. Calcular para todo I,J, Y(I,J), G(I,J). Hacer para todo I=1,...,M, $R(I) = B(I)$, $T(I) = \sum_{k} Y(I,K) * B(K)$ $(k = 1,...,M)$. Hacer $J = 1$.

1. Hacer $X(J) = \text{minimo}_{I} \{R(I)/A(I,J)\}$. Hacer para todo I = 1,...,M, $R(I) = R(I) - X(J) * A(I,J)$, $T(I) = T(I) - G(I,J) * X(J)$. Si para algún I = 1,...,M, R(I) = 0 ir a 11. Caso contrario ir a 2.

2. Hacer J = J + 1. Si J = N-M+1 ir a 3. Caso contrario hacer J = J-1 e ir a 1.

3. Si para algún I = 1,...,M $T(I) \neq 0$ y signo (T(I)) ≠ signo (D) ir a 8. Caso contrario si para todo I = 1,...,M, T(I)/D es entero ir a 4. Si no ir a 5.

4. Se ha encontrado la solución lexicográficamente máxima. Hacer para todo I = 1,...,M, X(N-M+I) = T(I)/D. Fin del algoritmo.

5. Si X(J) = 0 ir a 6. Caso contrario hacer X(J) = X(J)-1. Hacer para todo I = 1,...,M, R(I) = R(I) + A(I,J), T(I) = = T(I) + G(I,J) e ir a 2.

6. Si J = 1 ir a 7. Caso contrario hacer J = J-1 e ir a 5.

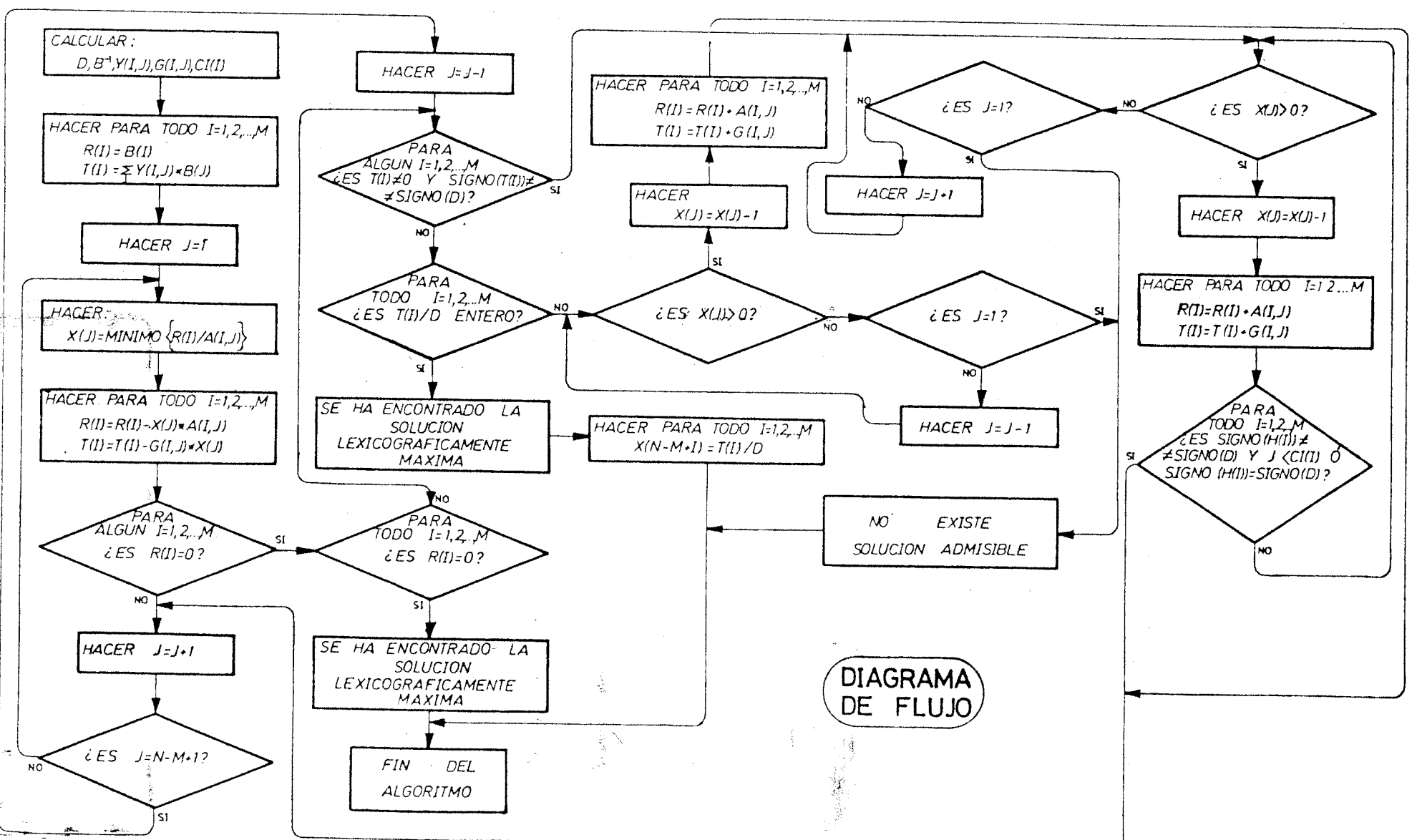
7. No existe ninguna solución admisible. Fin del algoritmo.

8. Si X(J) = 0 ir a 9. Caso contrario hacer X(J) = X(J) - 1. Hacer para todo I = 1,...,M, R(I) = R(I) + A(I,J), T(I) = = T(I) + G(I,J). Ir a 10.

9. Si J = 1 ir a 7. Caso contrario hacer J = J + 1 e ir a 8.

10. Si para I = 1,...,M signo (H(I)) = signo (D) ó singo (H(I)) ≠ signo (D) y J < CI(I) ir a 2. Caso contrario ir a 8.

11. Si para todo I = 1,...,M R(I) = 0 ir a 4. Caso contrario ir a 3.


CALCULAR :
D, B⁻¹, Y(I,J), G(I,J), CI(I)
HACER PARA TODO I=1,2,..,M
R(I) = B(I)
T(I) = Σ Y(I,J)*B(J)
HACER J=1
HACER:
X(J)=MINIMO {R(I)/A(I,J)}
HACER PARA TODO I=1,2,..,M
R(I)=R(I)-X(J)*A(I,J)
T(I)=T(I)-G(I,J)*X(J)
PARA ALGUN I=1,2,..,M ¿ES R(I)=0?
SI
NO
HACER J=J+1
¿ES J=N-M+1?
NO
SI
PARA TODO I=1,2,..,M ¿ES R(I)=0?
SI
SE HA ENCONTRADO LA SOLUCION LEXICOGRAFICAMENTE MAXIMA
FIN DEL ALGORITMO
NO
HACER J=J-1
PARA ALGUN I=1,2,..,M ¿ES T(I)≠0 Y SIGNO(T(I))≠SIGNO(D)?
SI
NO
HACER PARA TODO I=1,2,..,M
R(I) = R(I) + A(I, J)
T(I) = T(I) + G(I, J)
HACER X(J) = X(J) - 1
PARA TODO I=1,2,..,M ¿ES T(I)/D ENTERO?
NO
SI
¿ES X(J)>0?
NO
SI
¿ES J=1?
SI
NO
HACER J=J-1
HACER J=J+1
SE HA ENCONTRADO LA SOLUCION LEXICOGRAFICAMENTE MAXIMA
HACER PARA TODO I=1,2,..,M
X(N-M+I) = T(I)/D
NO EXISTE SOLUCION ADMISIBLE
¿ES J=1?
NO
SI
¿ES X(J)>0?
NO
SI
HACER X(J)=X(J)-1
HACER PARA TODO I=1,2,..,M
R(I)=R(I)+A(I,J)
T(I)=T(I)+G(I,J)
PARA TODO I=1,2,..,M ¿ES SIGNO(H(I))≠SIGNO(D) Y J<CI(I) Ó SIGNO(H(I))=SIGNO(D)?
SI
NO
DIAGRAMA DE FLUJO

CAPITULO VII

EXPERIENCIAS COMPUTACIONALES

7.1. Introducción

Para establecer el comportamiento de los algoritmos -- presentados en los capítulos anteriores se ha realizado un conjunto de experiencias computacionales con problemas de distintas características. Mediante el análisis de estas experiencias se estudia la eficiencia de los algoritmos, ya que su convergencia está garantizada al ser métodos que, en el peor de los casos, enumeran todo el conjunto finito de soluciones admisibles.

En el caso del problema knapsack se han realizado las experiencias comparando los algoritmos de enumeración lexicográfica aquí presentados con los más representativos de los empleados en su resolución. Se ha elegido el algoritmo de Gilmore y Gomory |50| (GG66) de los basados en programación dinámica y los de Gilmore y Gomory |48|, Cabot |19| e Ingargiola y Korsh |79| (GG63, - CA70 y IK77 respectivamente) como los más significativos de enume

ración. Denotaremos al algoritmo de la sección 4.3 con las siglas LR78 y al de la 4.4 con RF77.

Más problemático es establecer juicios relativos al -- comportamiento de los distintos métodos de resolución de proble-- mas lineales enteros, pues ninguno de los hasta ahora conocidos - es plenamente satisfactorio para cualquier tipo de modelo, dependiendo fundamentalmente de las características específicas del -- problema concreto que se analice. En cualquier caso, se han re--- suelto varios problemas con los algoritmos aquí propuestos.

7.2. Experiencias con el Problema Knapsack

En esta sección presentamos las experiencias con el -- problema knapsack.

7.2.1. Características de los Problemas Resueltos.

Con cada uno de los algoritmos se han resuelto problemas knapsack generados aleatoriamente. En primer lugar se describen las características comunes de todos los problemas. Estas han sido las siguientes :

1) Los coeficientes a_i y c_i (i = 1,...,n) han sido generados aleatoriamente de una distribución uniforme entre 10 y -170.

2) Para todo i ≠ j se cumple que $a_i \neq a_j$.

3) El valor $\rho_i = c_i/a_i$ para todo i (i = 1,...,n) ha de ser menor que un cierto valor ρ_{max} fijado en cada caso.

Se justifica a continuación el porque de las caracte-- rísticas señaladas :

1) Debido al tipo de ordenador utilizado (HP-21 MX), - el mayor entero posible es el número 32767. Para que los algoritmos funcionen con la máxima eficiencia, se trabaja con números enteros y para evitar que pueda sobrepasarse el mayor entero en algún cálculo se ha tomado el intervalo 10-170 al generar los coeficientes del problema.

2) Si en un problema knapsack dos variables (i,j) con distinto ρ ($\rho_i \neq \rho_j$) cumplen que $a_i = a_j$ entonces, una de ellas,- la de menor ρ, es cero. Para evitar esta simplificación de los -- problemas es por lo que se obliga en los problemas generados a -- que $a_i \neq a_j$ para todo i ≠ j.

3) Para poder actuar de alguna forma sobre la complejidad del problema generado, se obliga a que todas las $\rho_i = c_i/a_i$ sean menores que un cierto valor ρ_{max}. En un problema knapsack -- cuanto menor sea el intervalo entre la ρ máxima y la ρ mínima mayor será su complejidad si las ρ están distribuidas uniformemente. Considerando el intervalo de valores de donde están obtenidos a_i y c_i, el menor ρ posible es 10/170. Al fijar el ρ máximo de forma externa se está agrandando o reduciendo el intervalo de ρ. Para ρ máximo pequeño, el intervalo es pequeño y la complejidad del problema mayor.

Se han resuelto 30 problemas test de cada una de las posibles combinaciones obtenidas con las siguientes características :

1) El valor de ρ máxima, característica tres de los -- problemas test, ha sido fijada en 1, 3 y libre.

2) El valor del término independiente de la restric--- ción b ha sido fijado en 1000, 1500, 2000, 2500 y un valor obtenido aleatoriamente.

3) El número de variables del problema, N, ha sido 10, 20, 30, 40, 50, 60, 70 y 100.

7.2.2. Análisis de los resultados

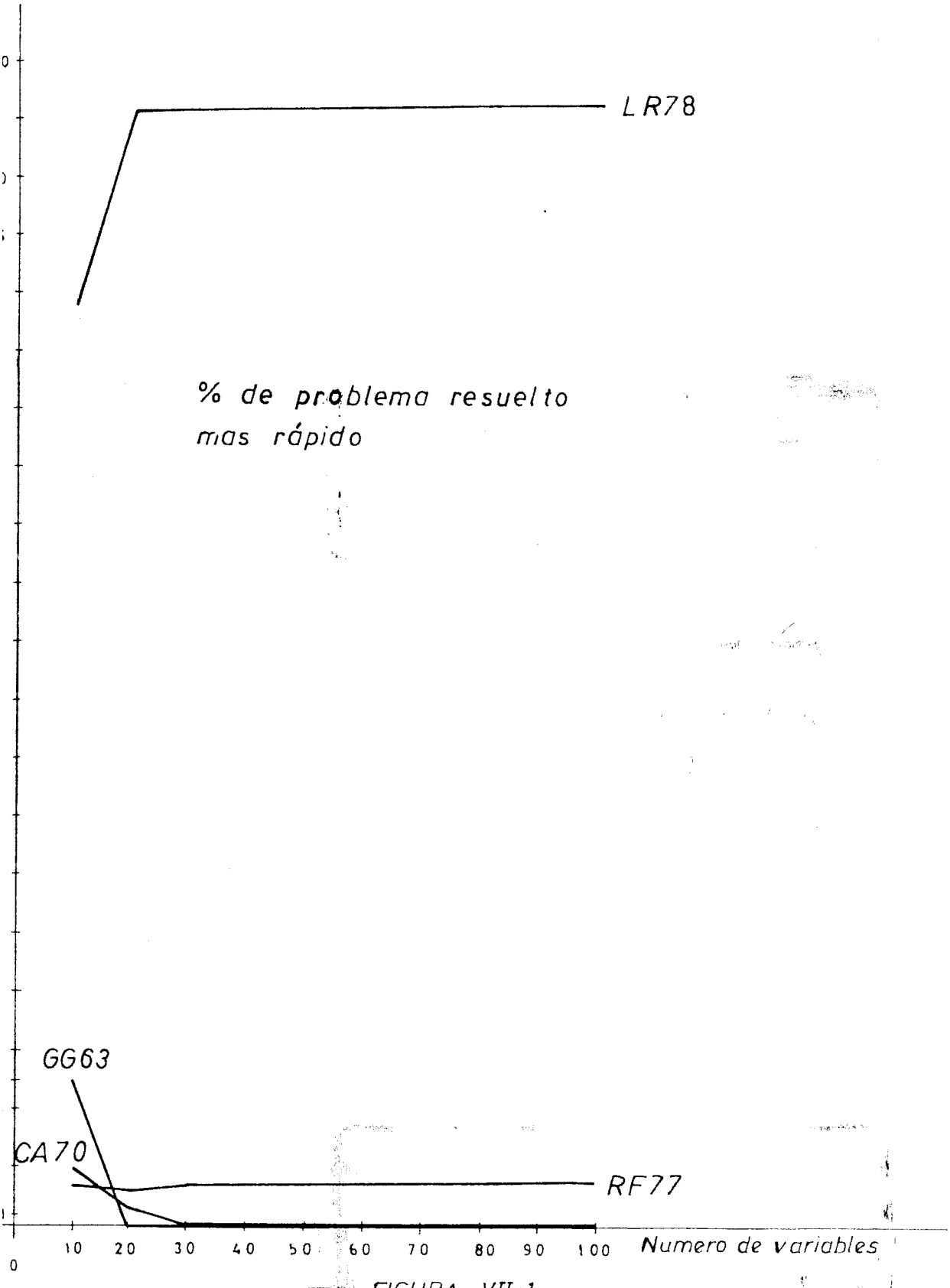
En el anexo 3 se incluye el resumen de todos los resultados obtenidos en la resolución de los problemas knapsack generados aleatoriamente. Para su discusión se exponen primero los resultados globalizados y posteriormente se analizan desagregadamente.

Se han resuelto en total 3600 problemas knapsack test de los distintos tipos ya descritos en el apartado anterior. En la tabla 7.1 se exponen cuantos de estos problemas han sido resueltos más rápido y más lento por cada uno de los algoritmos. Así, los algoritmos desarrollados en esta tesis (RF77 y LR78) han resuelto más rápidamente el 3,55% y el 94,05% respectivamente de todos los problemas resueltos. Por otra parte nunca han sido los más lentos en la resolución de un problema.

Desagregando la tabla 7.1 por tamaño de los problemas se obtienen para 10, 20, 30, 40, 50, 60, 70 y 100 variables las tablas 7.14, 7.15, 7.16, 7.17, 7.18, 7.19, 7.20 y 7.21, respectivamente. Representando gráficamente el porcentaje de veces que cada algoritmo ha sido más rápido (gráfica 7.1) y más lento (gráfica 7.2) se observa como ambos algoritmos RF77 y LR78, exceptuando los problemas de 10 variables son siempre los más rápidos: RF77 -

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	2051	56,97
GG63	57	1,58	-	-
CA70	29	0.80	22	0,61
IK77	-	-	1527	42,41
RF77	128	3,55	-	-
LR78	3386	94,05	-	-

Tabla 7.1


LR78
% de problema resuelto
mas rápido
GG63
CA70
RF77
10
20
30
40
50
60
70
80
90
100
Numero de variables

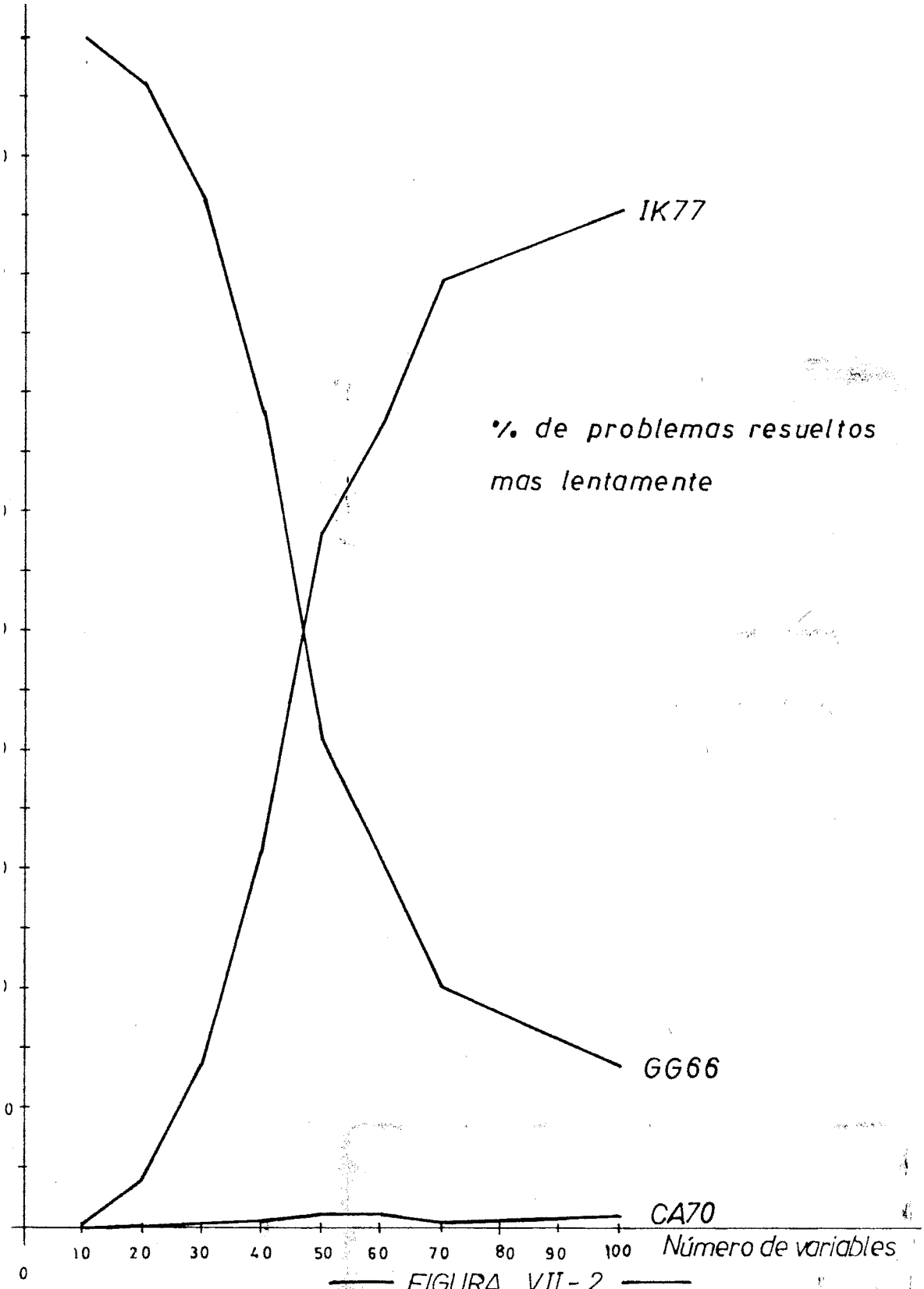

IK77
% de problemas resueltos
mas lentamente
GG66
CA70
10
20
30
40
50
60
70
80
90
100
Número de variables
0

FIGURA VII - 2

el 4% de las veces y LR78 el 96%. En la gráfica 7.2 se observa la progresiva pérdida de eficiencia de IK77 al aumentar el número de variables, pasando de no ser nunca el más lento para 10 variables a serlo el 86% de las veces para 100.

Respecto a los tiempos empleados, no tiene sentido el considerar los tiempos medios totales de cada uno de los algoritmos para los 3600 problemas resueltos, ya que existen distintos - comportamiento según el número de variables del problema y, como se verá posteriormente, según la complejidad e incluso el tamaño knapsack. En la gráfica 7.3 (tabla 7.2) se tiene los tiempos me-- dios (en segundos) de cada uno de ellos para los distintos núme-- ros de variables. A la vista de la gráfica 7.3 se puede señalar :

1) El algoritmo LR78 es el más rápido para cualquier - tamaño de problema.

2) A partir de 25 variables, RF77 es el mejor después de LR78.

3) A partir de 60 variables el tiempo medio de IK77 es superior al de GG66.


Tiempo medio
en segundos.
GG66
IK77
CA70
GG63
RF77
LR78
10
20
30
40
50
60
70
80
90
100
Numero de variables

FIGURA VII-3

	Número de Variables							
	10	20	30	40	50	60	70	100
GG66	0,624	1,167	1,865	2,594	3,091	3,705	4,340	7,053
GG63	0,065	0,163	0,321	0,569	0,802	1,033	1,404	2,784
CA70	0,068	0,200	0,443	0,755	1,073	1,244	1,621	2,966
IK77	0,186	0,559	1,296	1,945	2,945	3,687	5,097	9,728
RF77	0,079	0,176	0,290	0,456	0,563	0,718	0,900	1,544
LR78	0,045	0,102	0,174	0,275	0,378	0,501	0,655	1,223

Tabla 7.2

4) El aumento de los tiempos de todos los algoritmos - tiende a una linea recta. Considerando que es una escala logaritmica quiere ello decir que el tiempo medio tiende a ser una fun-- ción exponencial del número de variables.

Se han desagregado los resultados de la gráfica 7.3 en dos gráficas. La gráfica 7.4 es la de tiempos medios para problemas de ρ máxima menor o igual a 1 y la gráfica 7.5 es la obtenida para el ρ máxima libre. Es decir la gráfica 7.4 es la correspon-- diente a los problemas de mayor complejidad y la gráfica 7.5 a -- los de menor.

Lo más significativo de ambas gráficas es, por una parte, la fuerte pérdida de eficiencia de CA70, GG66 y IK77 al aumentar la complejidad de los problemas; por otra, la confirmación de los resultados de Cabot (1970). En los resultados presentados por Cabot con problemas generados aleatoriamente su algoritmo resultaba el más eficiente. En efecto esto era cierto pero únicamente para problemas donde no se restringen las ρ en un intervalo pequeño. Si las ρ son próximas el algoritmo GG63 es superior. Para problemas pequeños y con libertad de ρ, el algoritmo CA70 llega a ser - incluso superior a RF77. Para problemas pequeños y ρ máxima res-- tringida GG63 es superior a RF77. En ambos casos, ρ restringida o no, y para todos los tamaños de problemas LR78 es al algoritmo -- más eficiente.

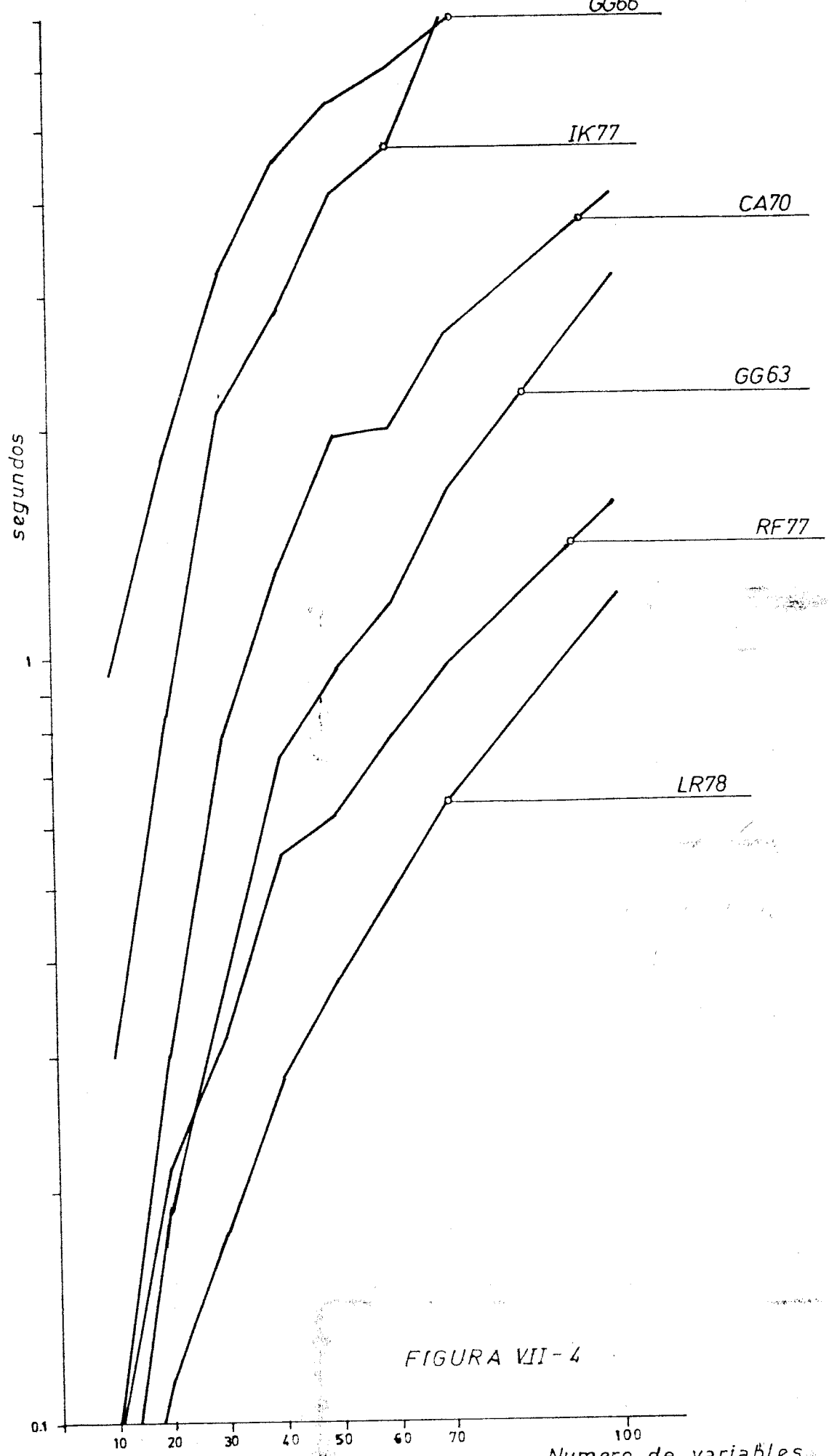
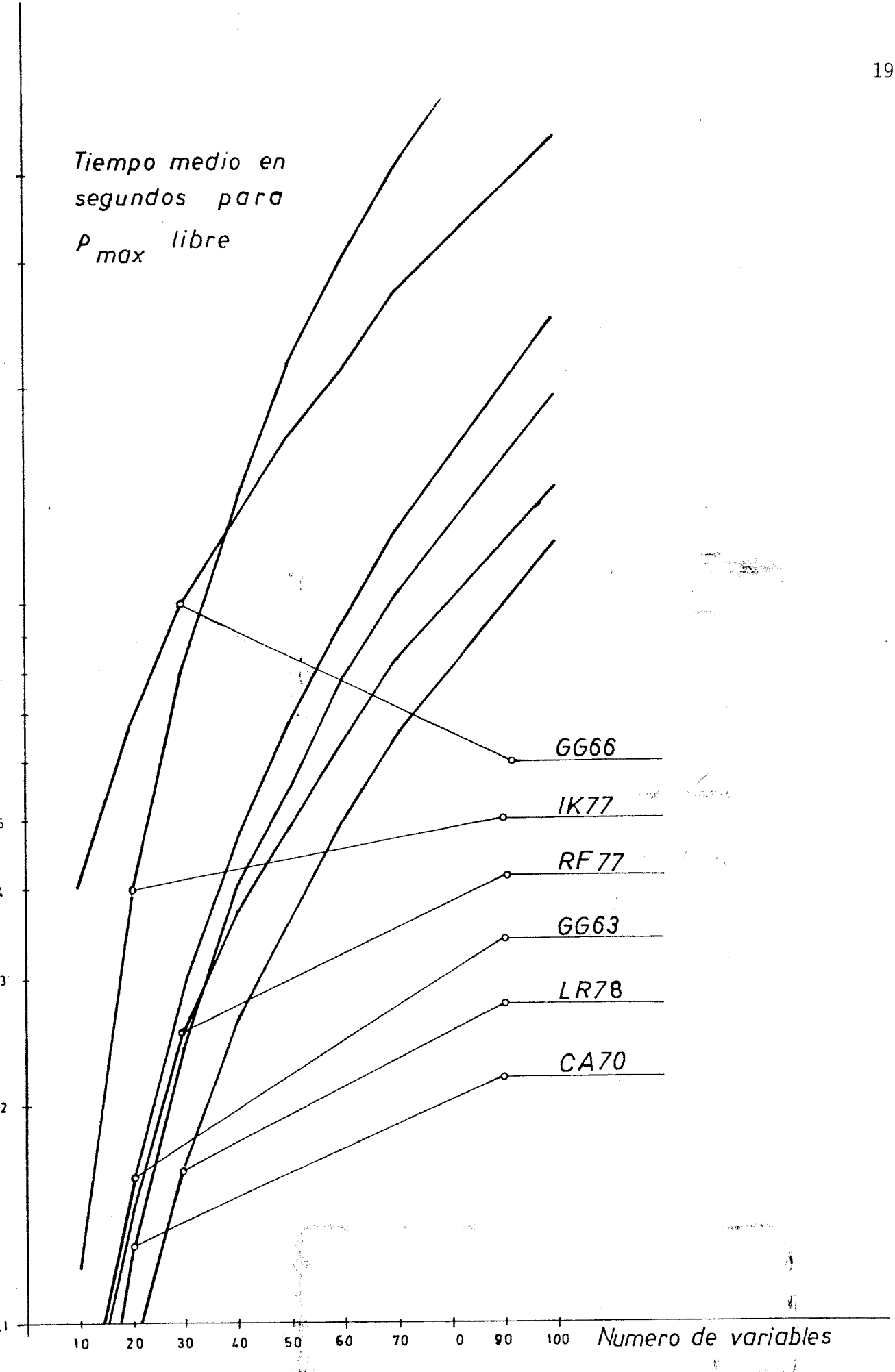

GG66
IK77
CA70
GG63
RF77
LR78
segundos
0.1
10
20
30
40
50
60
70
100
Numero de variables

FIGURA VII-4


Tiempo medio en segundos para p_{max} libre
GG66
IK77
RF77
GG63
LR78
CA70
10
20
30
40
50
60
70
0
90
100
Numero de variables

En las tablas 7.3, 7.4, 7.5. 7.6, 7.7, 7.8, 7.9 y 7.10 se tienen los tiempos medios en segundos, desagregados según la - complejidad, para problemas de 10, 20, 30, 40, 50, 60, 70 y 100 - variables, respectivamente.

Para establecer rigurosamente la incidencia del estrechamiento de la banda de ρ en el funcionamiento de los algoritmos se representa en el gráfica 7.6, para cada algoritmo, el porcentaje de incremento de tiempo de la gráfica 7.4 respecto de la gráfica 7.5. En otras palabras, en la gráfica 7.6 (tabla 7.11) se representa el porcentaje de aumento de tiempos medios de los problemas sin restricción en ρ máximo a los problemas restringidos a ρ máximo menor o igual a 1.

Como ya se había puesto de manifiesto, se confirma el fuerte efecto que sobre los algoritmos GG66, CA70 y IK77 tiene -- el aumento de la complejidad del problema. Por otra parte es de - destacar la estabilidad de LR78, no siendo prácticamente afectados los tiempos por este motivo, sobre todo para problemas de más de 20 variables.

En la gráfica 7.6 se observa que el comportamiento de todos los algoritmos es muy parecido. En todos ellos puede considerarse que hasta un cierto valor de N (número de variables del - problema) crece la incidencia hasta llegar a un máximo. A continuación decrece. El punto donde se produce el máximo, es decir el

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	0,400	0.510	0,964	0,624
GG63	0,071	0,053	0,071	0,065
CA70	0,048	0,054	0,100	0,068
IK77	0,119	0,075	0,302	0,186
RF77	0,680	0,137	0,095	0,068
LR78	0,040	0,042	0,051	0,045

Número de Variables : 10

Tabla 7.3

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	0,673	0,961	1,867	1,167
GG63	0,158	0,144	0,188	1,163
CA70	0,128	0,169	0,303	0,200
IK77	0,384	0,468	0,826	0,559
RF77	0,145	0,169	0,215	0,176
LR78	0,093	0,099	0,116	0,102

Número de Variables : 20

Tabla 7.4

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	1,001	1,383	1,211	1,865
GG63	0,285	0,295	0,384	0,321
CA70	0,242	0,304	0,784	0,443
IK77	0,810	1,009	2,070	1,296
RF77	0,254	0,296	0,321	0,290
LR78	0,168	0,176	0,178	0,174

Número de Variables : 30

Tabla 7.5

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	1,215	1,969	4,499	2,594
GG63	0,466	0,502	0,738	0,569
CA70	0,404	0,568	1,294	0,755
IK77	1,344	1,663	1,829	1,945
RF77	0,376	0,445	0,548	0,456
LR78	0,261	0,276	0,286	0,275

Número de Variables : 40

Tabla 7.6

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	1,708	2,210	5,356	3,091
GG63	0.679	0,768	0,959	0,802
CA70	0,549	0,711	1,960	1,073
IK77	2,132	2,616	4,087	2,945
RF77	0,492	0,579	0,620	0,563
LR78	0,364	0,388	0,382	0,378

Número de Variables : 50

Tabla 7.7

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	2,107	3,060	5,948	3,705
GG63	0,937	0,984	1,179	1,033
CA70	0,785	0,936	2,011	1,244
IK77	2,970	3,361	4,729	3,687
RF77	0,644	0,721	0,787	0,718
LR78	0,498	0,501	0,506	0,501

Número de Variables : 60

Tabla 7.8

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	2,697	3,546	6,776	4,340
GG63	1,260	1,315	1,636	1,404
CA70	1,015	1,188	2,660	1,621
IK77	4,069	4,324	6,899	5,097
RF77	0,831	0,887	0,980	0,899
LR78	0,655	0,658	0,651	0,655

Número de Variables : 70

Tabla 7.9

	ρ_{max} libre	$\rho_{max} \leq 3$	$\rho_{max} \leq 1$	Total
GG66	4,483	6,408	10,267	7,053
GG63	2,471	2,726	3,156	2,784
CA70	1,967	2,805	4,126	2,966
IK77	8,341	8,968	11,873	9,728
RF77	1,436	1,617	1,580	1,544
LR78	1,218	1,244	1,208	1,223

Número de Variables : 100

Tabla 7.10

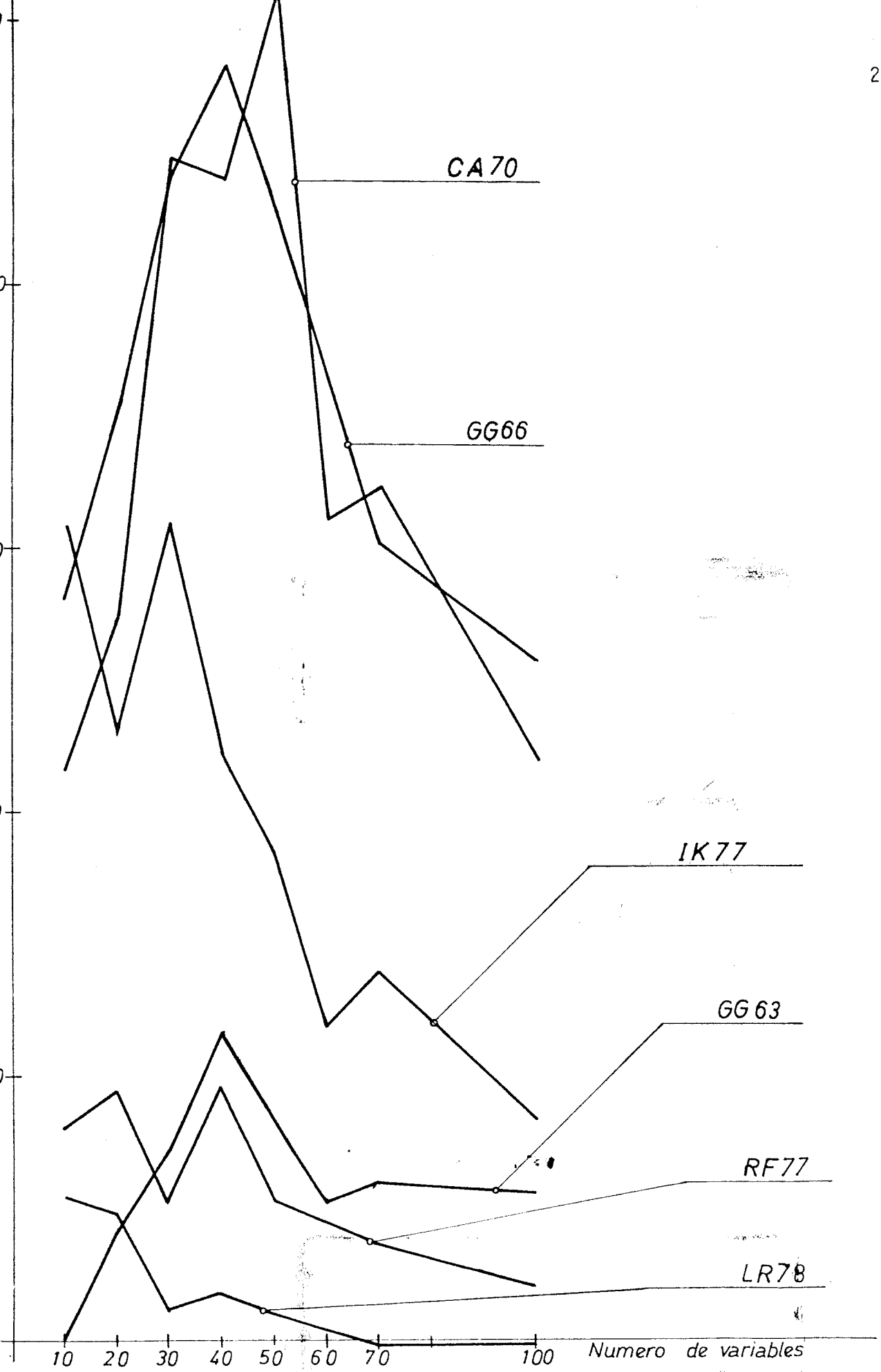

CA70
GG66
IK77
GG63
RF77
LR78
10
20
30
40
50
60
70
100
Numero de variables

FIGURA VII-6

Número de Variables

	10	20	30	40	50	60	70	100
GG66	141	177	221	242	214	182	151	129
GG63	0	20	35	58	41	26	30	28
CA70	108	137	224	220	257	156	162	110
IK77	154	115	155	111	92	59	70	42
RF77	40	47	26	48	20	22	18	10
LR78	27	24	6	9	5	2	-1	-1

Tabla 7.11

punto de máxima incidencia, varia de un algoritmo a otro. Dicho - máximo se encuentra :

- Para GG66 : 40 variables
- Para GG63 : 40 variables
- Para CA70 : 50 variables
- Para IK77 : 10 a 30 variables
- Para RF77 : 20 a 40 variables
- Para LR78 : 10 variables, no apareciendo en este algoritmo crecimiento inicial de la incidencia. Quizás no se presente o quizás se presente para problemas - de menos de 10 variables.

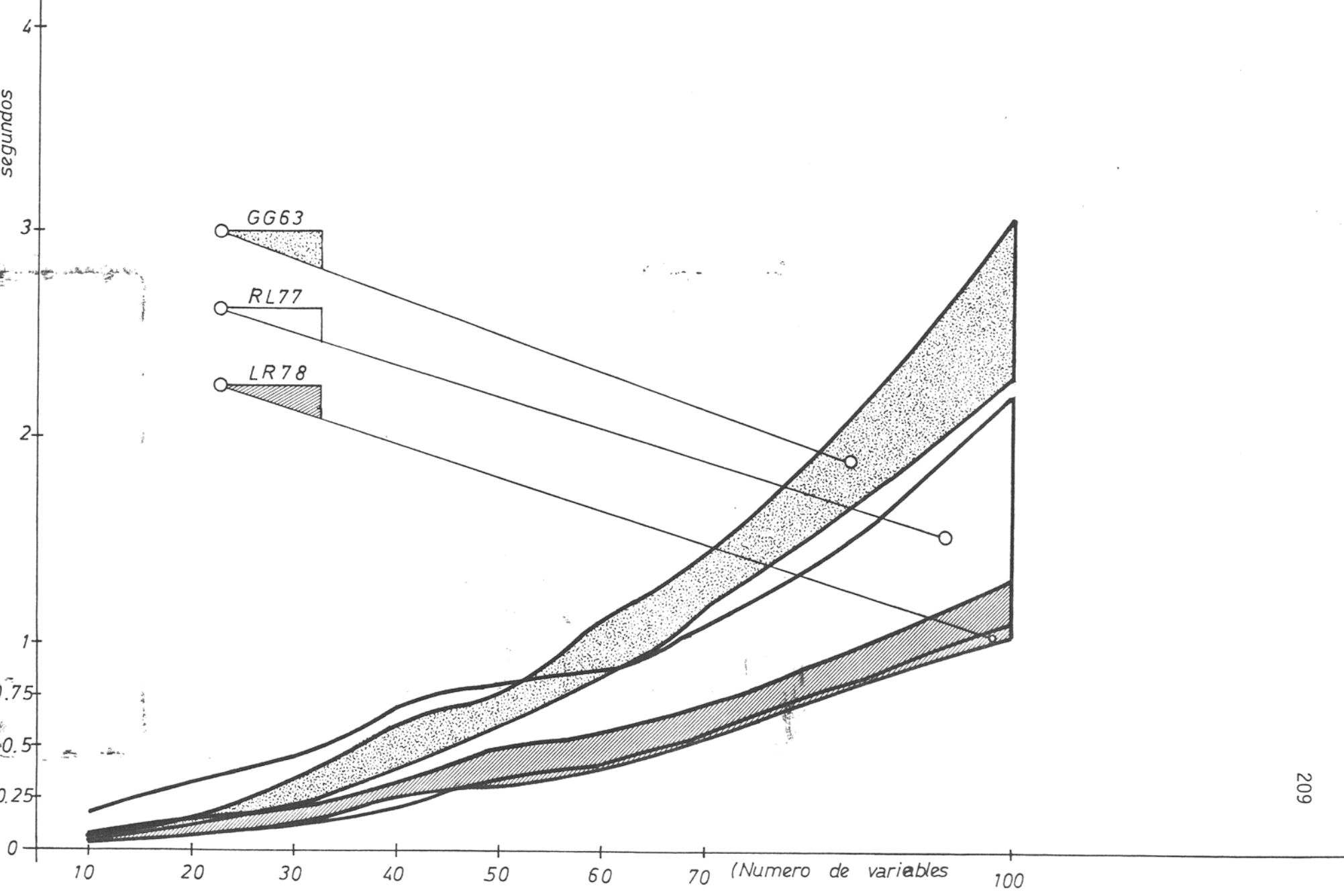
A continuación va a considerarse otro detalle digno de tenerse en cuenta en el funcionamiento de un algoritmo: la dispersión de resultados. Se realizará un estudio únicamente para los - dos algoritmos presentados y para el algoritmo en general más eficiente, GG63. Se representan en una gráfica los tiempos máximo y mínimo empleado por cada algoritmo en resolver los problemas para distinto número de variables. Según lo ancha que sea la banda ob-

tenida así será la dispersión. Se exponen los resultados obtenidos para problemas con ρ máximo libre (gráfica 7.7) y con ρ máximo menor o igual que 1 (gráfica 7.8).

Es de destacar que tanto en la gráfica 7.7 como en la 7.8 la banda de tiempos de LR78 es muy estrecha. Por ejemplo, un problema de 100 variables, cualquiera que sea su complejidad, el algoritmo LR78 lo resolverá entre 1 y 1,40 segundo, mientras GG63 puede tardar entre 2,30 y 4,25 segundos, y RF77 entre 1,1 y 3,1 segundos.

Por último hay que considerar la incidencia del tamaño knapsack (término independiente de la restricción) en el funcionamiento de los algoritmos. Para ello representamos el porcentaje de incremento en tiempo al pasar b de 1000 a 2500, en la resolución de problemas con ρ máxima libre (gráfica 7.9) y ρ máxima menor o igual a 1 (gráfica 7.10). Los valores correspondientes a la gráfica 7.9 están recogidos en la tabla 7.12 y los valores correspondientes a la gráfica 7.10 está recogidos en la tabla 7.13.

Del análisis de la gráfica 7.9 se pueden establecer dos grupos de algoritmos. En los algoritmos GG63 y GG66, al aumentar el tamaño knapsack aumenta el tiempo empleado en la resolución del problema y los algoritmos CA70, IK77, RF77 y LR78 sufren el efecto contrario.



$\rho_{max} \leq 13$

—FIGURA VII-7—

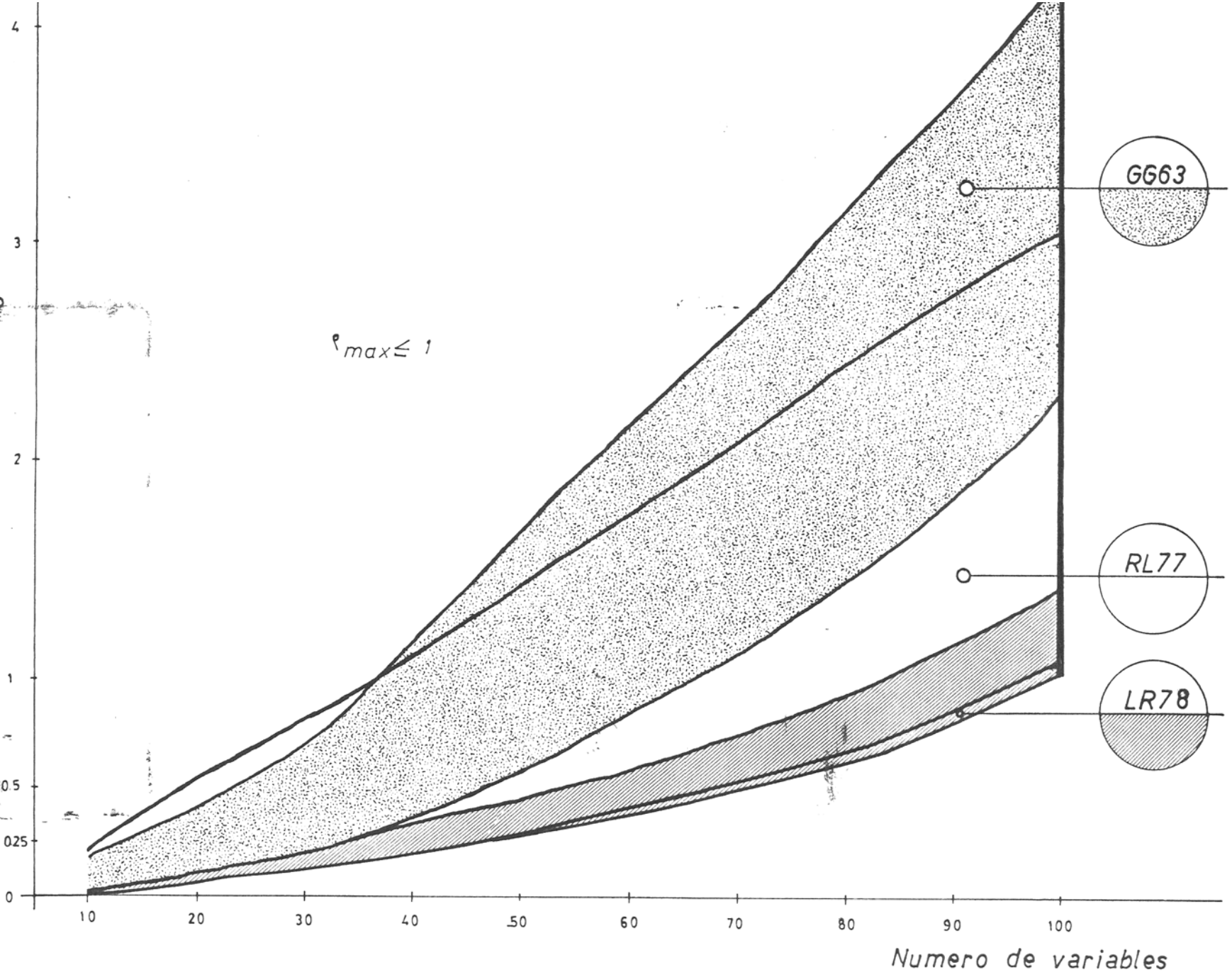

4
3
2
1
0.5
0.25
0
$\rho_{max} \leq 1$
GG63
RL77
LR78
10
20
30
40
50
60
70
80
90
100
Numero de variables

FIGURA VII-8

Para problemas con ϱ maxima libre
% de aumento de tiempo al pasar b
de 1000 a 2500.

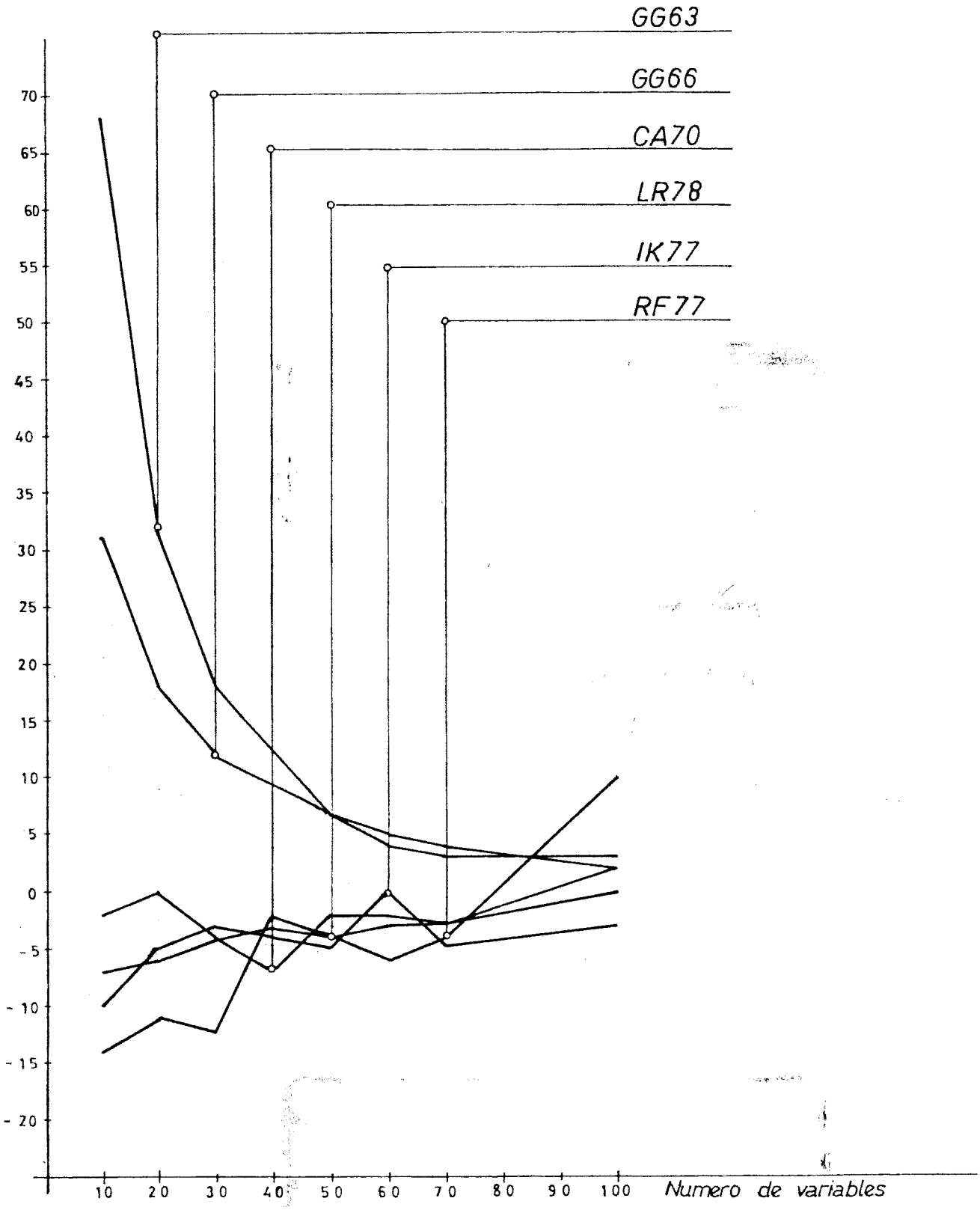
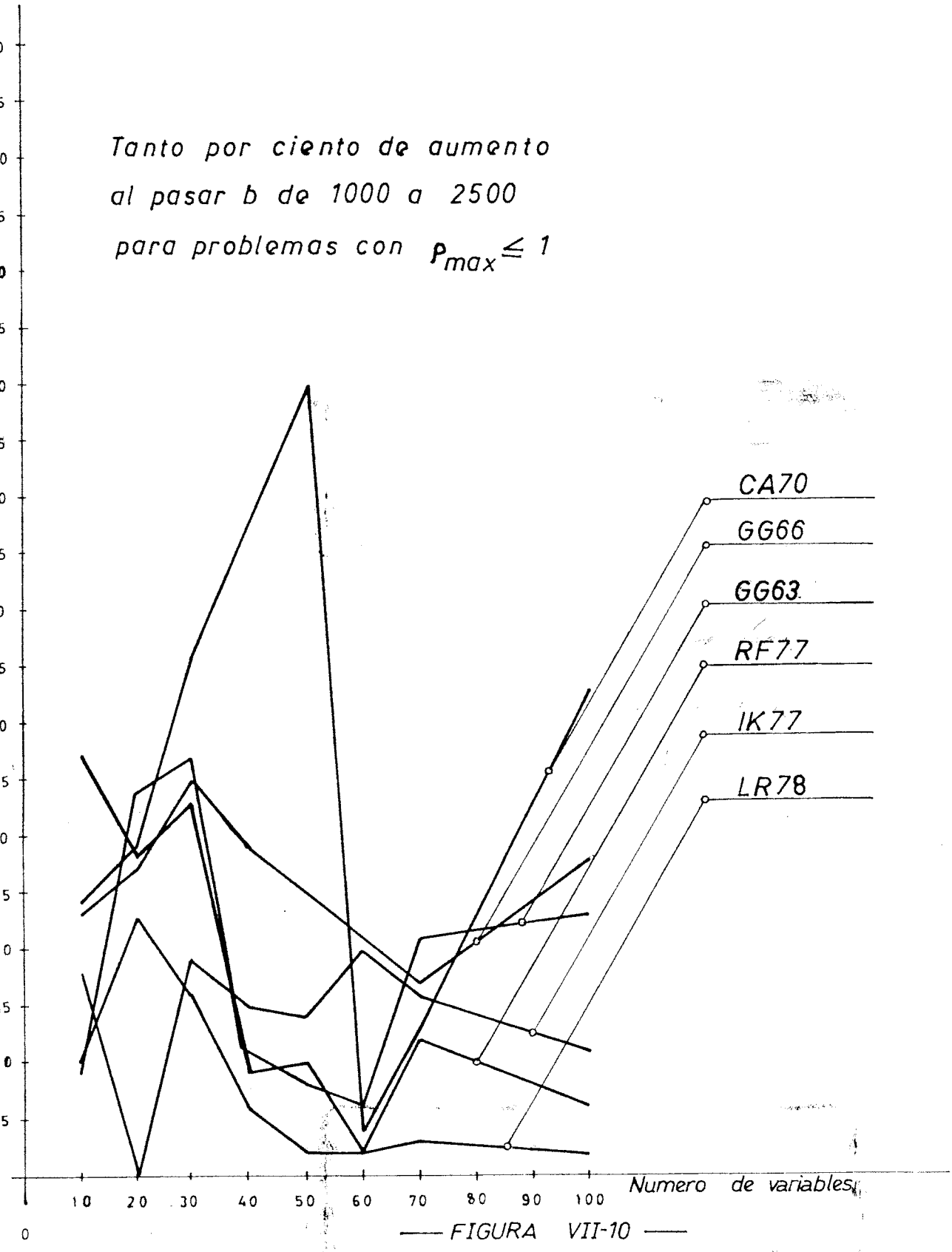


FIGURA VII-9

Tanto por ciento de aumento al pasar b de 1000 a 2500 para problemas con $p_{max} \leq 1$



— FIGURA VII-10 —

	Número de Variables							
	10	20	30	40	50	60	70	100
GG66	31%	18%	12%	9%	7%	5%	4%	2,5%
GG63	68%	32%	18%	12%	7%	4%	3%	3%
CA70	-2%	0%	-4%	-7%	-2%	-2%	-3%	2%
RF77	-14%	-11%	-12,5%	-2,5%	-15%	-6%	-4%	10%
IK77	-10%	-5%	-3%	-4%	-5%	0%	-5%	-3%
LR78	-7%	-6%	-4%	-3%	-4%	-3%	-3%	0%

Tabla 7.12

	Número de Variables							
	10	20	30	40	50	60	70	100
GG66	23%	27%	35%	29%	25%	21%	17%	28%
GG63	37%	28%	33%	11%	8%	6%	21%	23%
CA70	24%	29%	46%	57%	70%	4%	13%	43%
RF77	9%	34%	37%	9%	10%	2%	12%	6%
IK77	18%	0%	19%	15%	14%	20%	16%	11%
LR78	10%	23%	16%	6%	2%	2%	3%	2%

Tabla 7.13

El efecto es más fuerte en el primer grupo que en el - segundo y solo para problemas de pocas variables. Para problemas de 40 ó más variables al aumentar el tamaño knapsack el tiempo no se ve afectado practicamente. En el intervalo -10% , +10% se en-- cuentran todos los algoritmos.

Es de destacar que el incremento de tiempo en los al-- goritmos GG66 y GG63 al aumentar el tamaño knapsack de 1000 a -- 2500 es independiente del número de variables : Es constante y -- del orden de 0,115 segundos para GG66 y 0,044 segundos para GG63.

Para los problemas con ρ máxima menor ó igual que 1 -- (gráfica 7.10, tabla 7.13) se puede señalar como linea general -- que todos los algoritmos son afectados por el aumento del tamaño knapsack y que el efecto es más fuerte, igual que en el caso ante rior, para problemas de pocas variables.

Los algoritmos más afectados, en el presente caso, por el aumento de b, son GG66, GG63 y CA70. Puede considerarse que - el incremento de tiempo oscila entre un 20 y un 40%, mientras que para los restantes algoritmos IK77, RF77 y LR78 el incremento os- cila entre un 0 y un 20% siendo el algoritmo LR78 el menos afecta do.

Como conclusión del análisis del comportamiento de los algoritmos puede señalarse :

GG66 - El menos eficiente para problemas pequeños. Para problemas grandes tiende a ser, comparativamente, cada vez más eficiente. Le afecta fuertemente la disminución del intervalo de ρ y moderadamente el aumento del tamaño knapsack.

GG63 - Excluido los dos presentados en esta tesis es el mejor. No resulta afectado por el aumento de la complejidad del problema y moderadamente por el aumento del tamaño knapsack.

CA70 - Para problemas sin limitación en las ρ es superior a GG63. Sin embargo resulta fuertemente afectado por el aumento de la complejidad del problema.

IK77 - Para problemas pequeños funciona bien pero al aumentar el número de variables llega a ser el más ineficiente.

RF77 - Para problemas pequeños de hasta 30 variables es superado o por GG63 ó por CA70. Es poco sensible al aumento de la complejidad y del tamaño del knapsack.

LR78 - El más eficiente en todos los supuestos. Puede considerarse que el tiempo empleado en la resolución de un problema es solo función del número de variables de este.

7.3. Experiencias Computacionales con Problemas Lineales Enteros

En esta sección comentaremos brevemente los resultados obtenidos al emplear los algoritmos desarrollados en las secciones 5.3 y 5.5. La implementación en FORTRAN-IV se incluye en el anexo 3. Se han resuelto problemas de Haldi |68|, problemas test de IBM |116| y problemas generados aleatoriamente. Señalaremos los incovenientes encontrados al resolver los anteriores problemas con los dos algoritmos propuestos.

En el primer algoritmo, desarrollado para modelos con restricciones de igualdad, se pueden presentar dos tipos de dificultades :

- El cálculo de la solución lexicograficamente máxima es, en algunos problemas, laborioso. El esfuerzo de computacional es directamente proporcional a n-m, siendo n el número de variables y m el número de restricciones.

- Cuando, en el problema a resolver, existe un conjunto de aportaciones marginales (ρ_j) de valores próximos se detecta una falta de eficiencia del criterio de eliminación. Esta pérdida de eficiencia es lógica y se presenta también en el algoritmo de Gilmore y Gomory (GG63) para el problema knapsack, del cual es -- una extensión.

El segundo algoritmo resuelve eficientemente problemas lineales enteros positivos. Sin embargo, al resolver un problema lineal entero cualquier, surge el problema de su transformación - en positivo, debido a que las restricciones, en este caso, están planteadas con signo menor o igual.

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	449	99,78
GG63	57	12,66	-	-
CA70	22	4,88	-	-
IK77	-	-	1	0,22
RF77	16	3,55	-	-
LR78	355	78,88	-	-

Tabla 7.14

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	432	96
GG63	-	-	-	-
CA70	7	1,55	-	-
IK77	-	-	18	4
RF77	13	2,88	-	-
LR78	430	95,55	-	-

Tabla 7.15

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	388	86,22
GG63	-	-	-	-
CA70	-	-	1	0,22
IK77	-	-	61	13,55
RF77	16	3,55	-	-
LR78	434	96, 44	-	-

Tabla 7.16

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	305	67,77
GG63	-	-	-	-
CA70	-	-	3	0,66
IK77	-	-	142	31,55
RF77	18	4	-	-
LR78	432	96	-	-

Tabla 7.17

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	186	41,33
GG63	-	-	-	-
CA70	-	-	5	1,11
IK77	-	-	259	57, 55
RF77	16	3,55	-	-
LR78	434	96,44	-	-

Tabla 7.18

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	141	31,33
GG63	-	-	-	-
CA70	-	-	6	1,33
IK77	-	-	303	67,33
RF77	17	3,77	-	-
LR78	433	96,22	-	-

Tabla 7.19

	Más rápida		Más lento	
	Veces	%	Veces	%
GG66	-	-	89	19,77
GG63	-	-	-	-
CA70	-	-	3	0,66
IK77	-	-	358	79,55
RF77	14	3,11	-	-
LR78	436	96,88	-	-

Tabla 7.20

	Más rápido		Más lento	
	Veces	%	Veces	%
GG66	-	-	61	13,55
GG63	-	-	-	-
CA70	-	-	4	0,88
IK77	-	-	385	85,55
RF77	18	4	-	-
LR78	432	96	-	-

Tabla 7.21

CAPITULO VIII

CONCLUSIONES Y REFERENCIAS

CONCLUSIONES

1) Se ha estudiado la aplicación de los métodos de enumeración implicita a problemas lineales enteros de varias características.

2) Como paso previo a este estudio se ha efectuado una revisión crítica de los métodos actuales para la resolución de este tipo de modelos, analizándose con especial detalle los correspondientes al problema knapsack.

3) Se han presentado dos métodos de enumeración lexicográfica de soluciones admisibles para el problema knapsack.

3.1. El método denominado de acotaciones sucesivas es original, obteniéndose un algoritmo que es especialmente robusto. De acuerdo con las experiencias computacionales llevadas a cabo, su comportamiento es superior para todo tipo de problemas, a los algoritmos actualmente empleados.

3.2. El otro método presentado corresponde a una modificación del de Cabot y emplea también acotaciones sucesivas de la función objetivo, incrementando de forma muy apreciable la eficiencia del mismo.

4) Se ha analizado una nueva transformación que simplifica la resolución de cierto tipo de problema knapsack.

5) Se han presentado dos métodos originales de enumeración implicita para problemas lineales enteros con todos los coeficientes positivos.

5.1. El primer método propuesto extiende el criterio - empleado por Gilmore y Gomory para problemas con una sola restricción al caso de varias.

5.2. El de acotaciones sucesivas generaliza los resultados obtenidos para el problema knapsack mantenniendo criterios de eliminación similares.

6) Se ha planteado la transformación de cualquier problema lineal entero acotado en positivo extendiéndose así la aplicación de los anteriores algoritmos.

7) Se incluye un algoritmo para el cálculo de la solución lexicográficamente máxima con la que se inician todos los algoritmos de enumeración aquí presentados.

Como vías complementarias de desarrollo de este trabajo se ofrecen las siguientes :

Aplicación de los resultados al problema knapsack 0-1.

La extensión inmediata de los métodos de enumeración - implicita aquí propuestos a problemas lineales enteros :

1) Con región admisible no acotada.

2) Sin necesidad de transformarlos en positivos.

La profundización en las transformaciones propuestas - para convertir los problemas lineales enteros acotados en positivos, analizando las consecuencias de estas para la efectividad de los algoritmos aquí desarrollados.

Extensión de los resultados a problemas lineales mix-- tos.

REFERENCIAS

1. ABADIE, J., (1970)
"Integer and Nonlinear Programming".
American Elsevier, Nueva York.

2. BALAS, E., (1971)
"Intersection cuts - A new type of cutting for integer programming".
Operation Research. 19, 19-39.

3. BALAS, E., BOWMAN, V.L., GLOVER, F. y SOMMER, D. (1971)
"An Intersection cut from the dual of the unit hipercube".
Operation Research. 19, 40-44.

4. BALINSKI, M.L., (1965)
"Integer programming: Methods, uses, computation".
Management Science. 12, 253-313.

5. BALINSKI, M.L., y SPIELBERG (1968)
"Methods for Integer Programming: algebraic, combinatorial enumerative".
En Progress in Operation Research (J. Wiley (1969)).

6. BEALE, E.M.L. (1965)
"Survey of Integer Programming".
Operational Research Quarterly 16, 219-228.

7. BEALE, E.M.L., y SMALL, R.E. (1965)
"Mixed integer programming by a branch-and-bound techniques".
Proceeding IFIP Congress. Nueva York, Mayo 1965.

8. BELLMAN, R.E., (1957)
"Dynamic Programming"
Princeton University Press, Princeton.

9. BELLMAN, R.E. (1957)
"Comment on Dantzig's paper on discrete variable extremun problems".
Operation Research. 5, 723-724.

10. BELLMAN, R.E. y DREYFUS, S.E. (1962)
"Applied Dynamic Programming".
Princeton University Press, Princeton.

11. BELLMORE, M. y NEMHAUSER, G.L. (1968)
"The traveling salesman problem: A survey".
Operation Research. 16, 538-558.

12. BEN-ISRAEL y CHARNES, A. (1962)
"On some problems in diophantine programming".
Cahiers du Centre D'estudes de Recherche Operationelle, 4
215-280.

13. BOWMAN, V.J., y NEMHAUSER, G.L. (1970)
"A finiteness proof for modified Dantzig cuts in integer programming".
Naval Research Logistic Quarterly. 17, 309-313.

14. BOWMAN, V.J. y NEMHAUSER, G.L. (1971)
"Deep cuts in integer programming".
Opsearch. 8, 89-111.

15. BRADLEY, G.H. (1971)
"Algorithms for Hermite and Smith normal matrices and linear diophantine equations".
Mathematical Computer. 25, 897-908.

16. BRADLEY, G.H. (1971)
"Transformation of integer programs to knapsack problems"
Discrete Mathematik. 1, 29-45.

17. BRADLEY, G.H. (1971)
"Equivalent integer programs and canonical problems".
Management Science. 17, 354-366.

18. BROOK, R. y GEOFFRION, A.M. (1966)
"Finding Everett's Lagrange multipliers by linear programming"
Operation Research. 14, 1149-1153.

19. CABOT, V.A. (1970)
"An enumeration algorithm for knapsack problems"
Operation Research. 18, 306-311.

20. CABOT, V.A., y HURTER, A.P. (1968)
"An approach to 0-1 integer programming".
Operation Research. 16, 1206-1211.

21. CHANG, L. y KORSH, J.F. (1976)
"Canonical Coin Changing and Greedy Solutions"
Journal Association of Computer Machinery. 23, 418-422.

22. CHARNES, A. y COOPER, W.W. (1961)
"Management Models and Industrial Applications of Linear Programming".
Vols. 1 and 2, John Wiley & Sons.

23. COMPANYS, R. y BARCELO (1977)
"Panoramica actual de las técnicas Branch-and-Bound en programación en números enteros".
Seminario sobre Programación Matemática. Centro de Cálculo de la Universidad Complutense. Madrid. Noviembre 1977.

24. COROMINAS, A., y COMPANYS, R. (1977)
"Procedimiento generalizado de Branch and Bound".
Questió. 1, 49-62.

25. DAKIN, R.J. (1965)
"A tree-search algorithm for mixed integer programming pro--blems".
The Computer Journal. 8, 250-255.

26. DANO, S. (1968)
"Linear Programming in industry: Theory and Applications".
Springer-Verlag.

27. DANTZIG, G.B. (1957)
"Discrete variable extremum problems".
Operation Research. 5, 266-277.

28. DANTZIG, G.B. (1959)
"Notes on solving linear programs in integers".
Naval Research Logistic Quarterly. 6, 75-76.

29. DANTZIG, G.B. (1960)
"On the significance of solving linear programming problems - with some integer variables".
Econometrica 28, 30-44.

30. DANTZIG, G.B. (1963)
"Linear Programming and Extensions".
Princeton University Press, Princeton.

31. DANTZIG, G.B., FULKERSON, D.R. y JOHNSON, S.M. (1954)
"Solution of a large scale travelling salesman problem".
Operation Research. 2, 393-410.

32. DIJKSTRA, E.W. (1959)
"A note on two problems in connection with graphs".
Numerische Mathematik. 1, 269-271.

33. DORFMAN, R., SAMUELSON, P.A. y SOLOW, R.M. (1962)
"Programación lineal y análisis económico".
Aguilar.

34. DREYFUS, S.E. (1969)
"An Appraisal of Some Shortest-Path Algorithms".
Operation Research. 17, 395-412.

35. ELMAGHRABY, S.E. y WIG, M.K. (1970)
"On the treatment of cutting stock problems as diophantine programs".
Department of Industrial Engienier de North Carolina State University.

36. ESCUDERO, L.F. (1976)
"Programación lineal: continua, entera, mixta y bivalente".
Ediciones Deusto. Bilbao.

37. ESCUDERO, L.F. (1977)
"Panorámica actual de programación matemática".
Seminario sobre Programación matemática. Centro de Cálculo de la Universidad Complutense. Madrid. Noviembre 1977.

38. EVERETT, H. (1963)
"Generalized Lagrange multiplier method for solving problems of optimum allocation of resources".
Operation Research. 11, 399-417.

39. FAALAND, B. (1972)
"On the number of solutions to a diophantine equation".
Journal of Combinatorial Theory. 13, 170-175.

40. FAALAND, B. (1973)
"Solution of value independent knapsack problem by partitioning".
Operation Research. 21, 332-33.

41. FORD, L.R. Jr. y FULKERSON, D.R. (1962)
"Flows in Networks".
Princeton University Press. Princeton.

42. FRAZER, J.R. (1968)
"Applied linear programming".
Prentice-Hall.

43. GARFINKEL, R.S. y NEMHAUSER, G.L. (1972)
"Integer Programming".
Wiley, New York.

44. GASS, S.I. (1958)
"Linear Programming. Methods and Applications".
Mc-Graw-Hill.

45. GEOFFRION, A.M. (1974)
"Lagrangean Relaxation for Integer Programming".
Mathematical Programming Study 2. North Holland.

46. GEOFFRION, A.M. y MARSTEN, R.E. (1972)
"Integer programming: A framework and state-of-the-art survey"
Management Science. 18, 465-491.

47. GILMORE, P.C. y GOMORY, R.E. (1961)
"A linear approach to the cutting stock problem".
Operation Research. 9, 849-859.

48. GILMORE, P.C. y GOMORY, R.E. (1963)
"A linear programming approach to the cutting stock problem - Part II".
Operation Research. 11, 863-888.

49. GILMORE, P.C. y GOMORY, R.E. (1965)
"Multistage cutting stock problems of two or more dimensions".
Operation Research. 13, 94-120.

50. GILMORE, P.C. y GOMORY, R.E. (1966)
"The theory of computation of knapsack functions".
Operation Research. 14, 1045-1074.

51. GLOVER, F. (1965)
"A bound escalation method for the solution of integer linear programs".
Cahiers du Centre D'estudes de Recherche Operationelle.
6, 131-168.

52. GLOVER, F. (1966)
"Generalized cuts in diophantine programming".
Management Science. 13, 254-268.

53. GLOVER, F. (1967)
"Stronger cuts in integer programming".
Operation Research. 15, 1174-1176.

54. GLOVER, F. (1968)
"A new foundation for a simplified primal integer programming algorithm".
Operation Research. 16, 727-740.

55. GLOVER, F. y WOOLSEY, R.E. (1970)
"Aggregating diophantine equations".
Rep. No. 70-4. University of Colorado, Boulder.

56. GOMORY. R.E. (1958)
"Outline of an algorithm for integer solutions to linear programs".
Bulletin of America Mathematic Society. 64, 275-278.

57. GOMORY, R.E. (1963)
"All-Integer Integer Programming Algorithm".
193-206. en Muth and Thompson (1963).

58. GOMORY, R.E. (1963)
"An algorithm for integer solutions to linear programs".
En Recent Advances in Mathematical Programming (R.L. Graves y P. Wolfe. eds.). McGraw-Hill. New York.

59. GOMORY, R.E. (1965)
"On the relation between integer and non-integer solutions to linear programs".
Proceeding of National Academy of Science. U.S. 53, 260-265.

60. GOMORY, R.E. y HOFFMAN, A.F. (1963)
"On the convergence of an integer programming process".
Naval Research Logistic Quarterly. 10, 121-123.

61. GORRY, G.A. y SHAPIRO, J.F. (1971)
"An adaptive group theoretic algorithm for integer programming"
Management Science. 17, 285-306.

62. GORRY, G.A.., SHAPIRO, J.F. y WOLSEY, L.A. (1972)
"Relaxation methods for pure and mixed integer programming problems".
Management Science. 18, 229-239.

63. GRAVES, R.L. y WOLFE, P. (1963) (editores)
"Recent advances in Matematical Programming".
McGraw-Hill. New York.

64. GREENBERG, H. (1969)
"An algorithm for the computation of knapsack functions".
Journal of Mathematical Analysis Applied. 26, 159-162.

65. GREENBERG, H. (1969)
"A dynamic programming solution to linear integer programs".
Journal of Mathematical Analysis Applied. 26, 454-459.

66. GRENBERG, H. (1971)
"Integer Programming".
Academic Press, New York.

67. GREENBERG, H., y HEGERICH, R.L. (1970)
"A branch search algorithm for the knapsack problem".
Management Science. 16, 327-332.

68. HALDI, J. (1964)
"25 integer programming test problems".
Working Paper No. 43. Graduate Schol of Business.,
Stanford University.

69. HELLER, I., y TOMPKING, C.B. (1958)
"An extension of a theorem of Dantzig".
En Linear Inequalities and Related Systems. (H.W. Kuhn and
A.W. Tucker, eds.). Princeton University Press. Princeton.

70. HILLIER, F.S., y LIEDERMAN, G.L. (1970)
"Operations Research".
Holden-Day Inc. San Francisco.

71. HILLIER, F.S. (1969)
"Efficient heuristic procedures for integer linear programming with an interior".
Operation Research. 17, 600-637.

72. HILLIER, F.S. (1969)
"A bound-and-scan algorithm for pure integer linear programming with general variables".
Operation Research. 17, 638-679.

73. HOFFMAN, A.J., y KRUSKAL, J.B. (1958)
"Integral boundary points of convex polyhedra".
En Linear Inequalities and Related Systems (H.W. Kuhn and A. W. Tucker, eds.). Princeton University Press.

74. HOROWITZ, E., y SAHMI, S. (1974)
"Computing partitions with applications to the knapsack problems".
Journal Association of Computing Machinery. 21, 277-292.

75. HU, T.C. (1969)
"Integer Programming and Network Flows".
Addison-Wesley, Reading, Massachusetts.

76. HU, T.C. (1970)
"On the asymptotic integer algorithm".
Linear Algebra Applied. 3, 279-294.

77. HU, T.C., y LENARD, M.L. (1976)
"Optimality of a Heuristic Algorithm for a Class of Knapsack Problems".
Operation Research. 24, 193-196.

78. INGARGIOLA, G.P., y KORSH, J.F. (1973)
"A Reduction Algorithm for 0-1 Single Knapsack Problems".
Management Science. 20, 460-463.

79. INGARGIOLA, G.P., y KORSH, J.F. (1977)
"A General Algorithm for One-Dimensional Knapsack Problems".
Operation Research. 25, 752-759.

80. JEROSLOW, R.G. (1971)
"Comments on integer hulls of two linear constraints".
Operation Research. 19, 1061-1069.

81. KAUFMAN, A., y HENRY-LABORDERE, A. (1977)
"Integer and Mixed Programming: Theory and Applications".
Academic Press.

82. KOLESAR, P.J. (1967)
"A branch and bound algorithm for the knapsack problem".
Management Science. 13, 723-735.

83. LAND, A.H., y DOIG, A.G. (1960)
"An automatic method for solving discrete programming problems".
Econometrica. 28, 497-520.

84. LARRAÑETA, J. (1977)
"Programación Lineal y Grafos".
Publicaciones de la Universidad de Sevilla.

85. LARRAÑETA, J. (1977)
"Temas de Investigación Operativa".
Departamento de publicaciones de la E.S.I.I. Sevilla.

86. LAWLER, E.L. y WOOD, D.E. (1966)
"Branch-and-bound methods: A survey".
Operation Research. 14, 699-719.

87. LITTLE, J.D.C., MURTY, K.G., SWEENEY, D.W. y KAREL, C. (1963)
"An algoritm for the travelling salesman problem"
Operation Research. 11, 979-989.

88. LORIE, J., y SAVAGE, L.J. (1955)
"Three problems in capital rationing".
Journal of Bussines. 28, 229-239.

89. MAGAZINE, M., NEMHAUSER, G.L. y TROTTER, L.E. (1975)
"When the Greedy Solution Solves a Class of Knapsack Problems".
Operation Research. 23, 207-217.

90. MATHEWS, G. (1897)
"On the partition of numbers".
Proceeding of London Mathematical Society. 28, 486-490.

91. MATHIS, S.J., Jr. (1971)
"A counterexample to the rudimentary primal integer programming algorithm".
Operation Research. 19, 1518-1522.

92. MITRA, G. (1976)
"Theory and applications of mathematical programming".
Academic Press.

93. MORDELL, L.S. (1969)
"Diophantine Equations".
Academic Press.

94. NAUSS, R.M. (1976)
"A Efficient Algorithm for the Knapsack 0-1 Problem".
Management Science. 23, 27-31.

95. NEMHAUSER, G.L. (1966)
"Introduction to Dynamic Programming".
Wiley. New York.

96. PLANE, D.T., y McMILLAN, C. (1971)
"Discrete Optimization: Integer Programming and Network Analysis for Management Decisions".
Prentice-Hall, Englewood Cliffs, New Jersey.

97. RUBIN, D.S. (1970)
"On the unlimited number of faces in integer hulls of linear programs with a single constraint".
Operation Research. 18, 940-946.

98. RUBIN, D.S. y GRAVES, R.L. (1972)
"Strengthened Dantzig cuts for integer programming".
Operation Research. 20, 178-182.

99. RUIZ de FCO, F. (1977)
"Algoritmo de Enumeración para el Problema Knapsack".
X Reunión nacional de la S.E.I.O., Madrid.

100. RUIZ de FCO,F. y LARRAÑETA, J. (1978)
"Computational Experiences with the knapsack Problem".
Proceeding of II Conference on Applied Numerical Modeling.

101. SAATY, T.L. (1970)
"Optimization in Integers and Related Extremal Problems".
McGraw-Hill, New York.

102. SALKIN, H.M., y DEKLUYVER, C.A. (1973)
"The Knapsack Problem: A Survey".
Technical Memo No. 281, Department of Operations Research.
Case Western Reserve University.

103. SALKIN, H.M. (1973)
"A Brief Survey of Algorithm and Recent Results in Integer Programming".
Opsearch. 10, 81-123.

104. SALKIN, H.M. y DEKLUYVER, C.A. (1975)
"The knapsack Problem: A Survey".
Naval Research Logistic Quarterly. 22, 127-144.

105. SALKIN, H.M. (1975)
"Integer Programming".
Addison-Wesley, Reading, Massachusetts.

106. SHAPIRO, J.F. (1968)
"Dynamic programming algorithms for the integer programming problem-I: The programming problem viewed as a knapsack type problem".
Operation Research. 16, 103-121.

107. SHAPIRO, J.F. (1968)
"Group theoretic algorithms for the integer programming - problem-II: Extension to a general algorithm".
Operation Research. 16, 928-947.

108. SHAPIRO, J.F. (1968)
"Shortest route methods for finite state space deterministic dynamic programming problems".
SIAM. Journal on Applied Mathematics. 16, 1232-1250.

109. SHAPIRO, J.F., y WAGNER, H.M. (1967)
"A finite renewal algorithm for the knapsack and turnpike models".
Operation Research. 15, 319-341.

110. SIMONNARD, M. (1966)
"Linear Programming".
Prentice-Hall, Englewood Cliffs, New Jersey.

111. TAHA, H.A. (1971)
"A Class of Convexity Cuts for 0-1 Linear Programs with Application to the Minimization of Certain Concave - Problems".
Rep. No. 71-3. Dept. of Industrial Engineering, University of Arkansas.

112. TAHA, H.A. (1975)
"Integer Programming: Theory, Appplications and Computations".
Academic Press.

113. THESEN, A. (1978)
"Computer Methods in Operation Research".
Academic Press.

114. TIEN, B.N., y HU, T.C. (1977)
"Error Bounds and the Applicability of the Greedy Solution to the Coin-Changing Problem".
Operation Research. 25, 404-418.

115. TOMLIN, J.A. (1970)
"Branch and bound methods for integer and non-convex Programming".
Integer and Nonlinear Programming (J. Abadie, ed.). Amer. Elsevier.

116. TRAUTH, C.A. y WOOLSEY, R.E. (1969)
"Integer lineal programming: A study in computational efficiency".
Management Science. 15, 481-493.

117. VEINOTT, A.F., Jr., y DANTZIG, G.B. (1968)
"Integral Extreme Points".
SIAM Review. 10, 371-372.

118. WAGNER, H.M. (1969)
"Principles of Operations Research".
Prentice-Hall.

119. WEINGARTNER, H.M. (1963)
"Mathematical Programming and the Analysis of Capital Budgeting Problems".
Prentice-Hall.

120. WEINGARTNER, H.M. (1966)
"Capital budgeting of interrelated projects: Survey and synthesis".
Management Science. 12, 485-516.

121. WOLSEY, L.A. (1971)
"Extensions of the group theoretic approach in integer programming".
Management Science. 18, 74-83.

122. YOUNG, R.D. (1968)
"A simplified primal (all-integer) integer programming algorithm".
Operation Research. 16, 750-782.