UNIVERSIDAD DE SEVILLA

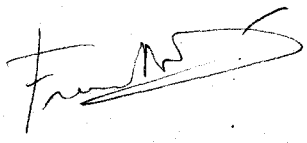
FACULTAD DE MATEMATICAS

EFICIENCIA Y ESTRUCTURAS DINAMICAS:

PUNTOS PARETO ENTEROS

Visado en Sevilla

El Director



Memoria que presenta Antonio Pozo Chia para optar al grado de Doctor en Ciencias Matemáticas

Fdo. Francisco R. Fernandez García

Fdo. Antonio Pozo Chía

EFICIENCIA Y ESTRUCTURAS DINAMICAS:

PUNTOS PARETO ENTEROS

ANTONIO POZO CHIA

Autorizo la consulta de esta memoria

Quiero expresar mi agradecimiento a los compañeros del Departamento de Estadística e Investigación Operativa por la colaboración prestada en la realización del presente trabajo.

Sevilla Noviembre de 1985

Los problemas de programación entera han experimentado un importante desarrollo en las dos últimas décadas, y sus aplicaciones se han extendido a múltiples campos, si bien es cierto que, con bastante frecuencia, los desarrollos teóricos resultan parcialmente satisfactorios en sus aplicaciones, debido a que la mayoría de los problemas enteros resultan ser NP-completos o NP-duros.

Las primeras aplicaciones de la programación entera fueron desarrolladas por Dantzing en 1954 y por Markowitz y Manne en 1957; éstas sirvieron de base a Gomory en 1958 para desarrollar el primer método de planos de corte finito para el caso entero puro, y en 1960 al caso entero mixto. Inspirado en la idea de los planos de corte, aparecieron posteriormente los métodos asintóticos.

En 1965, Bertier y Roy presentan el primer trabajo teórico general de los métodos de ramificación y acotación, que experimentarían un gran auge en los años siguientes con Balas y Mitten. Un caso particular de estos métodos, son los de enumeración implícita, que fueron iniciados por Balas y Glover en 1965 y después por Geoffrion, y que son aplicables a los problemas cero-uno.

Desde estos planteamientos iniciales, numerosos autores han ampliado los métodos anteriores y desarrollado otros nuevos, basados fundamentalmente en el álgebra de Boole, la teoría de grupos y la relajación lagrangiana (Geoffrion,1974 y Shapiro,1980); algunos resultados muy alentadores proceden

también de la aplicación de determinados heurísticos a problemas de algún tipo concreto.

La complejidad de la mayoría de los problemas de programación entera ha sido establecida recientemente en los trabajos de Gurari (1977), Karp (1975), Sahni (1976) y otros.

El problema que planteamos en nuestro trabajo es la búsqueda de los puntos eficientes enteros de un poliedro, aplicando una generalización de los métodos de enumeración implícita y usando estructuras dinámicas.

Dichos métodos enumerativos son desarrollados para ser aplicados a problemas lineales aunque no sean del tipo cero-uno, y la enumeración implícita se hace sobre un entorno de la superficie eficiente del poliedro.

La obtención de los puntos eficientes, exige disponer de estructuras dinámicas de datos adecuadas, que filtren dichos puntos de entre los enumerados, para que en cada momento, el espacio requerido sea mínimo.

Las dificultades que aparecen en el proceso, aparte del manejo de las estructuras de datos, es que no sabemos si el conjunto de puntos de coordenadas enteras, pertenecientes al poliedro, es no vacío, y si así fuese, tampoco sabemos la frecuencia con que aparecen dichos puntos. En algunos casos, que llamaremos poliedros denso enteros, se demuestra que puede pasarse de un punto a otro por un camino interior al poliedro, pero en los casos que pueda plantearse la duda de que los puntos enteros estén aislados o no existan, habrá que estudiar nuevos procedimientos y demostrar que con ellos, la accesibilidad a cualquier punto eficiente de coordenadas enteras está garantizada.

En el capítulo 1, hacemos una revisión de las estructuras dinámicas de datos que han sido utilizadas en otros problemas de búsqueda multidimensional, fundamentalmente, listas lineales, árboles ordenados, q-árboles y kd-árboles.

En el capítulo 2, aplicamos las estructuras anteriores a la localización de puntos eficientes, y se hace un estudio del rendimiento de cada una de ellas obteniéndose el kd-árbol como la estructura más eficiente en nuestro problema.

En el capítulo 3, demostramos que para los poliedros denso enteros, podemos partir de cualquier punto factible y siguiendo un camino uno a uno direccional contenido en el poliedro, podemos llegar a cualquier punto eficiente entero; a lo largo de dicho camino, los puntos visitados serán almacenados en una estructura kd-árbol de forma que al terminar la enumeración la estructura tiene almacenada todos los puntos eficientes enteros.

Para poliedros más generales el proceso se complica debido a que puede que no existan puntos enteros factibles, o que estén aislados en el poliedro; en este caso, estudiamos distintas relajaciones que permitirán obtener sus puntos eficientes enteros, o detectar si no existe ninguno, siguiendo un camino uno a uno direccional contenido en el poliedro relajado; los puntos visitados que sean factibles iran a la estructura kd-árbol que almacenará los puntos eficientes.

Dada la complejidad, anteriormente citada, de los problemas enteros, si el poliedro tuviese un número muy elevado de puntos factibles enteros, el proceso podría hacerse muy largo, pero entonces podemos aplicar los procedimientos estudiados a zonas definidas por unos márgenes de variación para cada una de las coordenadas.

I.- Estructuras dinámicas

1.1.- Introducción

Cuando en la resolución de un problema se requiere el uso del ordenador, necesitamos con frecuencia, almacenar conjuntos de datos en una estructura apropiada con el fin de que las consultas y actualizaciones puedan ser hechas con la mayor eficiencia posible. En este sentido, dos tipos de eficiencia son particularmente importantes:

i.- La cantidad de tiempo necesario

ii.- El volumen de memoria que ocuparemos

Estas ideas, que hasta hace pocos años dependian de la particular habilidad de cada individuo, se han ido sistematizando y analizando en conexión con los algoritmos utilizados, formando una rama denominada "Diseño y análisis de algoritmos".

En el objetivo de minimizar ambos conceptos las estructuras desempeñan un papel fundamental como veremos a lo largo de este capítulo. Estas estructuras podemos clasificarlas en dos tipos: estáticas y dinámicas, aunque en ocasiones, suele considerarse un tercer tipo mezcla de los dos anteriores, las estructuras semidinámicas.

Las estructuras estáticas son construidas para un conjunto fijo de elementos, sobre el que unicamente pueden hacerse consultas; su complejidad se mide por el tiempo de construcción, memoria ocupada y tiempo de consulta.

Las estructuras dinámicas permiten inserciones y borrados de elementos; su complejidad se mide por el tiempo de actualización, tiempo de consulta y volumen de memoria ocupada.

Desde que en 1975 se introduce la llamada Geometría Computacional, el uso de las estructuras dinámicas para las búsquedas multidimensionales han experimentado un gran desarrollo.

En este capítulo haremos una somera revisión de las estructuras estáticas fundamentales:

- Datos individuales
- Matrices
- Registros
- Conjuntos
- Ficheros

y haremos un estudio más detallado de las estructuras dinámicas:

- De reconstrucción local
- De reconstrucción parcial
- De reconstrucción total

aunque nos limitaremos al estudio de las dos primeras ya que el tipo tercero resulta, en general, poco eficiente.

Antes de iniciar el estudio de las estructuras, daremos unas claves sobre la notación y parte del vocabulario que emplearemos.

Si x es un número real usaremos:

$[x]$ redondeo al entero más próximo a x

$\lceil x \rceil$ menor número entero mayor o igual que x o redondeo por exceso de x

$\lfloor x \rfloor$ mayor número entero menor o igual que x o redondeo por defecto de x

Si $X(x_1, x_2, \ldots x_n)$ es un punto de R

$$\lceil X \rceil = (\lceil x_1 \rceil , \lceil x_2 \rceil , \ldots \lceil x_n \rceil)$$
$$\lceil X \rceil = (\lceil x_1 \rceil , \lceil x_2 \rceil , \ldots \lceil x_{\acute{n}} \rceil)$$
$$\lfloor X \rfloor = (\lfloor x_1 \rfloor , \lfloor x_2 \rfloor , \ldots \lfloor x_n \rfloor)$$

El concepto de tiempo medio conviene precisarlo; si una estructura tiene en el momento presente n elementos, diremos que permite actualizaciones en un tiempo medio F(n), cuando con cada actualización existe un número n´ tal que el total de tiempo necesario para esta actualización y todas las n´-1 anteriores está acotado por n´F(n), es decir, F(n) acotaría el valor medio del tiempo empleado en las n´-1 actualizaciones precedentes y la actual, bien sean inserciones o borrados o mezcla de ambas.

Esta definición cubre la mayoría de las definiciones de tiempo medio en la literatura sobre estructuras dinámicas de datos. Algunas definiciones pretenden tener en cuenta todas las actualizaciones N, habidas hasta el momento (n´=N sería este valor y por consiguiente un caso particular del anterior); pero esta definición referida a N es muy débil pues pensemos en una estructura que tuviese muchos más puntos en el pasado que en el presente.

Las cotas de eficiencia de las estructuras de datos son en general, expresadas en términos de n que sería el número de puntos presentes del conjunto. A veces, las cotas son expresadas en términos de m que indicará el máximo número de elementos que ha tenido el conjunto; también suele expresarse en términos de N que representará el número de actualizaciones habidas desde que se iniciara la estructura vacía.

Para la estimación de cotas usaremos las siguientes notaciones; sea f(n) y g(n) dos funciones enteras para $n \geq 0$:

$g(n)$ es $O(f(n))$ si existe una constante c tal que $g(n) \leq cf(n)$

$g(n)$ es $\Omega(f(n))$ si existe una constante $c > 0$ tal que $g(n) \geq cf(n)$

$g(n)$ es $\theta(f(n))$ si existen constantes c,d con $c > 0$ tal que $cf(n) \leq g(n) \leq df(n)$

$g(n)$ es $o(f(n))$ si para toda constante $c > 0$ es $g(n) \leq cf(n)$

Además, si S es una estructura de datos con n puntos

Q (n) será el tiempo necesario para realizar una consulta sobre S

P (n) el tiempo necesario para construir S

D (n) el tiempo necesario para borrar un elemento de S

M (n) la cantidad de memoria ocupada por S

Supondremos que todas las funciones anteriores son no decrecientes, aunque serán funciones "smooth", es decir, verificarán que $f(O(n))=O(f(n))$.

Todas estas cotas serán consideradas en el peor de los casos; cuando usemos tiempos medios lo indicaremos expresamente.

1.2.- Estructuras estáticas

Las estructuras estáticas fundamentales son las siguientes:

a.- Datos individuales: usualmente son números enteros,

reales o complejos, caracteres o cadenas de caracteres y datos booleanos.

b.- Matrices: son ordenaciones de datos individuales, de un mismo tipo en una, dos o más dimensiones.

c.- Registros: son estructuras que pueden contener datos de distintos tipos.

d.- Conjuntos: son estructuras que contienen datos de un mismo tipo y tales que las operaciones definidas entre ellos son las propias de los conjuntos: unión, intersección, diferencia y la relación de pertenencia.

e.-Ficheros: la característica fundamental de las estructuras anteriores es su cardinalidad finita, por lo que presentan pocas dificultades de implementación. Los ficheros son estructuras más avanzadas dado que su tamaño puede crecer imprevisiblemente y, de alguna manera, deben disponer de algún sistema de asignación dinámica de memoria que permita ir archivando nuevos valores o que libere memoria si desaparecen elementos.

Existen tres tipos fundamentales de ficheros:

e1.- Secuenciales: en este tipo de ficheros, los elementos se van guardando uno a continuación de otro como si fuese una pila:

elemento 1

elemento 2

-

-

elemento k

-

Un elemento nuevo debe colocarse, tras el último que ya estaba en el fichero y solo podemos borrar por el final. Podemos hacer consultas sobre todos los demás elementos "secuencialmente" pero no hacer inserciones o borrados ya que esto solamente puede hacerse por el final.

El espacio ocupado en cada momento corresponde a los elementos del fichero, de manera que, si llegan nuevos elementos el espacio aumenta y si se borran, el espacio ocupado por el fichero disminuye.

e2.- Acceso directo: en estos ficheros, cada elemento tiene un índice que habitualmente sigue el orden natural, no permitiendo actualizaciones intermedias; la ventaja fundamental es que para efectuar consultas, el índice que precede a cada elemento permite localizarlo directamente.

e3.- Secuenciales indexados: en este caso, las claves índices no tienen que ser consecutivas; los elementos se almacenan por claves sin que importe el orden de entrada. Permiten insertar claves intermedias entre dos no consecutivas y efectuar el borrado de claves. No obstante, tienen el problema de un gran movimiento de información en las inserciones y borrados intermedios, lo que hace que estas actualizaciones, aunque permitidas, no sean eficientes, haciendo que la estructura sea inapropiada en problemas de actualizaciones frecuentes.

1.3.- Estructuras dinámicas locales

1.3.1.- Fundamentos de las estructuras dinámicas

Las estructuras estáticas no pueden variar a lo largo de un proceso y esto da lugar a serias dificultades; así por ejemplo, si en un vector de N componentes queremos guardar N elementos de forma que queden ordenados según su posición sobre el vector de menor a mayor, cada vez que insertemos un elemento nos veremos obligados a desplazar una posición a la derecha todos los ya insertados que sean mayores que él; esta cantidad de desplazamientos es el primer problema que se plantea, dado que para un N grande puede suponer un consumo de tiempo considerable.

Un segundo problema es, que una vez almacenados los N elementos ordenados, no podemos insertar ningún elemento más sobre ese vector, dado que la reserva de las N componentes es inalterable en el proceso.

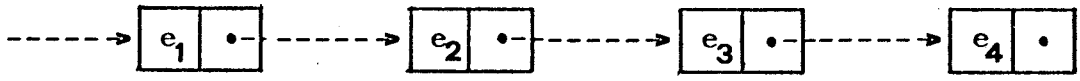
El tercer problema es, que si a lo largo de una tarea, debemos repetir el proceso de ordenación anterior para un número de elementos menor que N, la reserva de memoria será de N posiciones elementales en cada momento, lo que hace que para valores mucho menores que N la estructura no sea eficiente, debido a la desproporción existente entre lo necesario y lo reservado.

Estos tres problemas quedan resueltos si usamos una estructura dinámica cuya propiedad fundamental es que puede variar de tamaño dentro de un proceso, eficientemente; estas

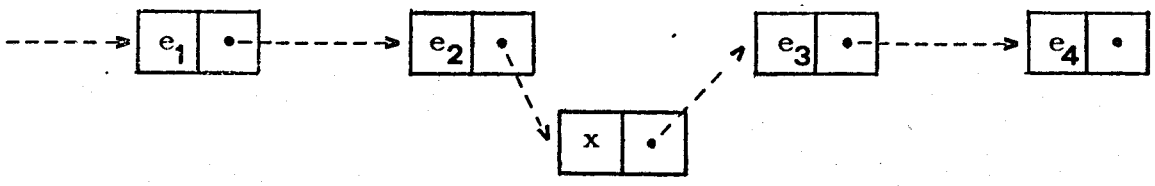
estructuras no tienen asignada una cantidad fija de memoria, sino que hay una asignación dinámica en la que a cada elemento nuevo se le asigna una dirección en el momento de ser introducido; de esta forma, cada elemento puede ser almacenado con la dirección del elemento que le sigue; así, los elementos:

$$e_1 < e_2 < e_3 < e_4$$

quedarían:



de esta manera si x es un elemento a insertar tal que $e_2 < x < e_3$ todo se reduce a un cambio de direcciones:



esto sería un ejemplo de una lista ordenada enlazada, cuyo tamaño puede variar sin más que cambiar determinadas direcciones.

1.3.2.- Estructuras de reconstrucción local

Son estructuras en las que cualquier tipo de actualización solo exige una reordenación local de alguna de sus direcciones; estas estructuras son fundamentalmente, las listas lineales y ciertos tipos de árboles.

En las listas lineales, una actualización, sea inserción o borrado, solo supone la reorientación de una (borrado) o dos (inserción) direcciones; son pues, estructuras de reconstrucción local para cualquier conjunto de elementos en el que haya definida una relación de orden total, por lo que puede ser aplicado al caso en que los elementos son vectores usando el

orden lexicográfico.

Para el estudio de los árboles daremos previamente algunas definiciones; un árbol se define por:

i.- La estructura vacía

ii.- La que resulta de añadir un elemento (que se llamará nodo) a un número finito de árboles (que llamaremos subárboles).

Podemos adoptar una definición de árbol más explícita, diciendo que un árbol es un conjunto finito T de uno o más nodos tales que:

i.- Existe un nodo especial llamado raiz del árbol

ii.- Los nodos restantes están agrupados en m conjuntos disjuntos ($m > 0$) $T_1, T_2, \ldots, T_m$ y cada uno de estos conjuntos es a su vez un árbol. Los árbole T_i, $i=1,\ldots,m$ se llaman subárboles de la raiz.

Es evidente, según la definición recursiva anterior que cada nodo de un árbol es la raiz de algún subárbol contenido en el árbol total.

Grado de un nodo es el número de subárboles que tiene; un nodo de grado cero se llama nodo terminal u hoja; un nodo no terminal lo llamaremos nodo interior.

El nivel de un árbol se define por:

i.- La raiz del árbol tiene nivel 1

ii.- Si un nodo tiene nivel k, las raices de sus subárboles tienen nivel k+1.

Un nodo y es descendiente de otro x si y pertenece al subárbol de raiz x; en particular, y es descendiente directo de x si y es raiz de algún subárbol de x.

Análogamente se define antecesor y antecesor directo.

Según la definición, si un nodo tiene nivel k, sus descendientes directos tienen nivel k+1.

Profundidad o altura de un árbol, será el máximo de los niveles de todos los nodos del árbol.

Grado de un árbol es el máximo de los grados de todos los nodos del árbol.

Observemos que el nivel de un nodo, es el número de segmentos a recorrer para llegar a él partiendo de la raiz y suponiendo que para llegar a la raiz hemos de recorrer uno de tales segmentos; en tal caso, el nivel de un nodo x coincide con la longitud de camino de x.

La longitud de camino interno de un árbol es la suma de todas las longitudes de camino de cada uno de los nodos, es decir:

$$L = \sum_i n \cdot i$$

siendo n el número de nodos del nivel i; la longitud de camino medio es

$$C = L / n$$

siendo n el número total de nodos del árbol

Un árbol binario es un árbol de grado dos, es decir, un árbol binario es un conjunto finito de elementos llamados nodos, que o bien está vacío, o está formado por un nodo llamado raiz con dos árboles binarios disjuntos, llamados subárbol izquierdo y derecho de la raiz.

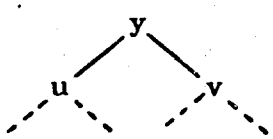
Los árboles de grado superior a dos suelen llamarse árboles multicaminos.

Arbol binario ordenado es aquel que para cada nodo se verifica, que es mayor que los nodos del subárbol izquierdo y menor que los del subárbol derecho; nuevamente vemos que un árbol

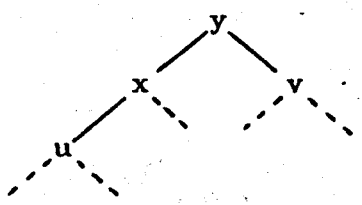
binario ordenado es otra estructura capaz de contener un conjunto de elementos con un orden total.

En este tipo de árbol, la búsqueda de un elemento x, partiendo de la raiz, es inmediata comparando simplemente con el nodo y sobre el que nos encontramos; si es y=x ya está localizado el elemento buscado; si x>y se continúa la búsqueda sobre el subárbol derecho de y, mientras que si es x<y se haría sobre el izquierdo.

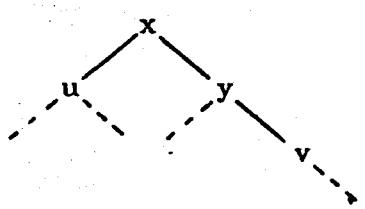
Para insertar un nuevo elemento x en un árbol ordenado, seguiremos un procedimiento análogo al de búsqueda hasta llegar a un elemento $y \neq x$ tal que:



con x<v y x>u; en este caso, si x<y el nuevo árbol sería:



o bien:

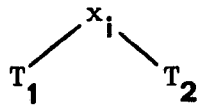


si x>y se obtendrían estructuras simétricas.

Para borrar un elemento x de un árbol ordenado procederíamos

a su localización; si x tiene un único descendiente directo y, sustituiremos x por y, con lo que x queda eliminado; si x tiene un subárbol izquierdo y otro derecho no vacíos, lo sustituiremos por el nodo más a la derecha del subárbol izquierdo o por el nodo más a la izquierda del subárbol derecho; en cualquiera de los casos, se ha hecho una reconstrucción local y el árbol queda ordenado tras su actualización.

Veamos cuál sería la longitud de camino media de un árbol ordenado con n elementos $x_1 < x_2 < \ldots < x_n$ suponiendo que cualquier distribución es igualmente probable; uno de estos casos:



x_i es la raiz; el subárbol T_1 tiene i-1 nodos y el T_2 n-i nodos; sea c_j la longitud de camino media de un árbol con j nodos; lo que buscamos es c_n; c_n^i será la longitud de camino media del árbol de n nodos y raiz x_i; así pues:

$$c_n = (\sum_i c_n^i)/n$$

pero los nodos del árbol de raiz x_i pueden dividirse en tres partes: los i-1 nodos de T_1 que tienen una longitud de camino media $c_{i-1}+1$ y la longitud de camino interno $(i-1)(c_{i-1}+1)$; la raiz, cuya longitud de camino es 1, y los n-i nodos de T_2 cuya longitud de camino media es $c_{n-i}+1$ y longitud de camino interno $(n-i)(c_{n-i}+1)$.

Así pues, la longitud de camino interno del árbol de raiz x_i es:

$$(i-1)(c_{i-1}+1)+1+(n-i)(c_{n-i}+1)$$

con lo que:

$$c_n^i = ((i-1)(c_{i-1}+1)+1+(n-i)(c_{n-i}+1))/n$$

$$c_n = \sum_{i=1}^{n}((i-1)(c_{i-1}+1)+1+(n-i)(c_{n-i}+1))/n^2$$

$$= (n + \sum_{i=1}^{n}((i-1)(c_{i-1}+1)+(n-i)(c_{n-i}+1))/n^2$$

$$= 1/n + (\sum_{i=1}^{n}(i-1)(c_{i-1}+1)+ \sum_{i=1}^{n}(n-i)(c_{n-i}+1))/n^2$$

$$= 1/n + (2\sum_{i=1}^{n}(i-1)(c_{i-1}+1))/n^2$$

$$= 1/n + (2\sum_{i=1}^{n-1} i(c_i+1))/n^2$$

$$= 1 + 2(\sum_{i=1}^{n-1} ic_i)/n^2$$

obtengamos una relación de recurrencia en la que c_n sea función unicamente de c_{n-1}:

$$c_n = 1 + 2(n-1)c_{n-1}/n^2 + 2(\sum_{i=1}^{n-2} ic_i)/n^2$$

$$c_{n-1} = 1 + 2(\sum_{i=1}^{n-2} ic_i)/(n-1)^2$$

$$\sum_{i=1}^{n-2} ic_i = (n-1)^2 (c_{n-1}-1)/2$$

luego

$$c_n = ((n^2-1)c_{n-1} + 2n-1)/n^2$$

puede comprobarse que c puede expresarse en forma no recursiva, en términos de la función armónica:

$$Hn = 1+1/2+1/3+------+1/n$$

resultando:

$$a_n = 2(n+1)Hn/n - 3$$

esta expresión puede aproximarse usando la fórmula de Euler:

$$Hn = \gamma + Ln + 1/(12n^2) + \text{----}$$

con lo que para n grande:

$$c_n = 2(n+1)(\gamma + Ln + 1/(12n^2) + \text{----})/n - 3$$

$$= 2(\gamma + Ln + 1/(12n^2) + \ ) + 2(\gamma + Ln + 1/(12n^2) + \ )/n - 3$$

$$c_n \simeq 2(Ln + \ ) - 3 = 2Ln\text{-cte.} \qquad (\gamma = 0.557..)$$

Esta expresión es muy interesante para comparar la eficiencia de este tipo de árboles ordenados, con los árboles ordenados equilibrados que estudiamos a continuación.

Un árbol binario está equilibrado si para cada nodo, la altura del subárbol derecho y la del izquierdo difieren a lo sumo en una unidad.

Un árbol binario está perfectamente equilibrado si para cada nodo, el número de nodos del subárbol derecho y el izquierdo difieren a lo sumo en una unidad.

Los árboles equilibrados son importantes ya que la longitud de camino puede ser sensiblemente menor que la del árbol ordenado con los mismos elementos sin criterios de equilibrio. En efecto, puesto que la longitud de camino media de un árbol equilibrado es mayor o igual que la del árbol ordenado perfectamente equilibrado, designando por c' la longitud de camino media del árbol equilibrado y por c'' la del perfectamente equilibrado se tiene:

$$\lim c_n / c'_n \leqslant \lim c_n / c''_n$$
$$= \lim 2Ln/\log_2 n = 2L2 = 1.386..$$

esto significa que con el árbol equilibrado se puede conseguir una mejora en la longitud de camino media de hasta el 39% sobre el árbol sin criterios de equilibrio.

No obstante, si queremos mantener el equilibrio del árbol con las distintas actualizaciones, nos veremos obligados a llevar un control del equilibrio de cada nodo y proceder al reequilibrado cuando éste se pierda; esto puede conseguirse usando rotaciones de uno de los tipos indicados en la figura 1.

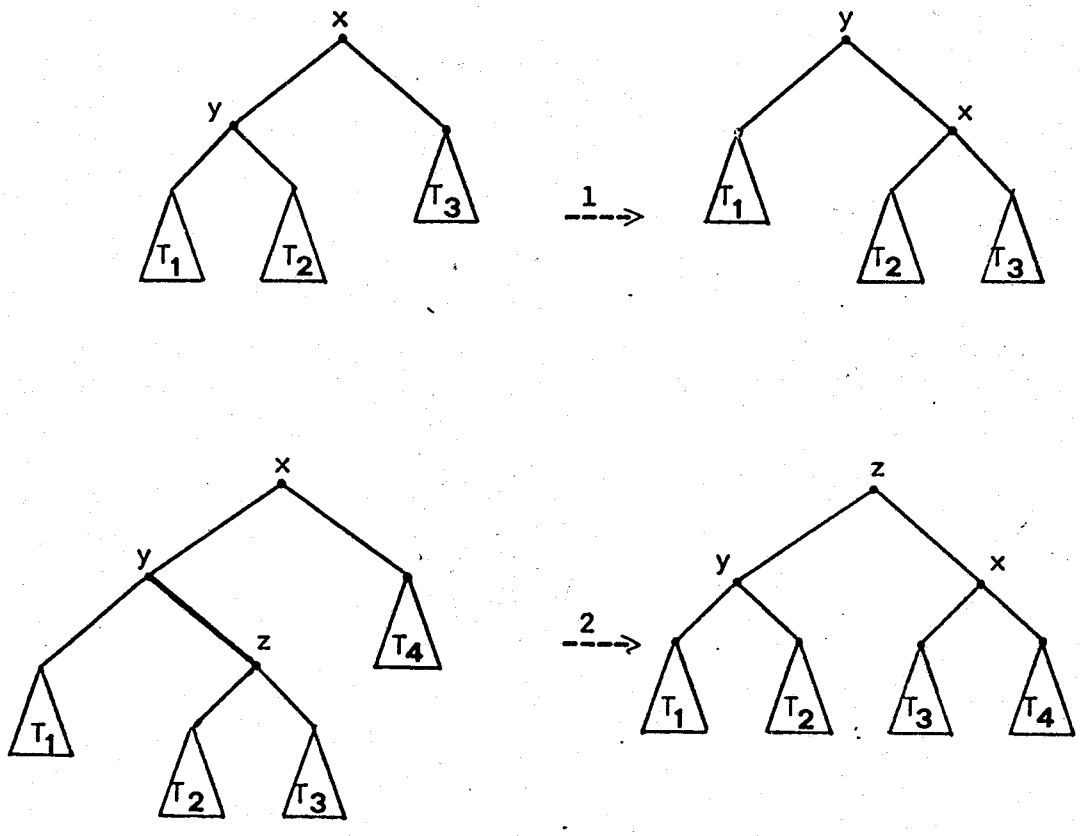


Fig. 1

Estos giros sirven para mantener los árboles equilibrados en altura tal como han sido definidos. No obstante, existen otros tipos de equilibrios.

Equilibrio en peso: cuando la razón entre el número de nodos del subárbol derecho de cada nodo y el total de descendientes de dicho nodo pertenece a un intervalo determinado;por ejemplo al (1/3,2/3). Un ejemplo típico son los árboles BB(α).

Equilibrio en grado: para cada nodo la altura del subárbol izquierdo y la del derecho han de ser iguales, a costa de que varíe el grado de los nodos internos; por ejemplo lo árboles 2-3.

Equilibrio en camino: se limita la longitud del camino más largo de la raiz a los hojas; por ejemplo los árboles αBB.

Vimos anteriormente que la mayor ventaja esperada de camino medio del árbol ordenado perfectamnte equilibrado, sobre el árbol ordenado era del 39%, sin embargo, el mantenimiento del equilibrio, hace que la eficiencia real de los árboles equilibrados sea muy próxima en término medio a la eficiencia del árbol ordenado; basta tener en cuenta que el desequilibrio provocado al borrar un elemento, puede afectar a todos los nodos comprendidos entre la posición de dicho elemento y la raiz.

1.4.- Estructuras dinámicas de reconstrucción parcial

1.4.1.- Q-árboles

Los q-árboles (quad tree) fueron introducidos por Finkel y Bentley (1974) como una primera estructura para resolver los problemas de búsqueda en rango y otros problemas multidimensionales.

Sea $X = \{X_1, X_2, \ldots, X_n\}$ un conjunto de puntos de un espacio k-dimensional E^k ; un q-árbol es un árbol de grado 2^k construido tomando un punto cualquiera $X_i \in X$ como raiz del q-árbol; X_i divide E^k en 2^k cuadrantes y por lo tanto, el conjunto X en 2^k subconjuntos, cada uno de los cuales formará un subárbol de X_i. Cada cuadrante es dividido nuevamente tomando un punto del subconjunto que contiene y así sucesivamente hasta que cada subcuadrante tenga a lo sumo un punto de X.

La figura 2 muestra un q-árbol en dos dimensiones.

Para que esta construcción sea posible vemos que no puede existir una pareja de puntos con alguna de sus coordenadas iguales, ya que, tal como hemos descrito la estructura, cada división ha de pasar por un único punto del conjunto X.

Los q-árboles presentan algunos inconvenientes además del citado anteriormente; en primer lugar, aunque un nodo de un q-árbol k-dimensional puede tener 2^k descendientes directos, el q-árbol puede tener altura $\log_2 n$ cuando, por ejemplo, todos los puntos están en la diagonal principal en el caso del plano, como indica la figura 3. Esta altura es muy superior a la que sería la altura óptima dada por $\log_{2^k} n$.

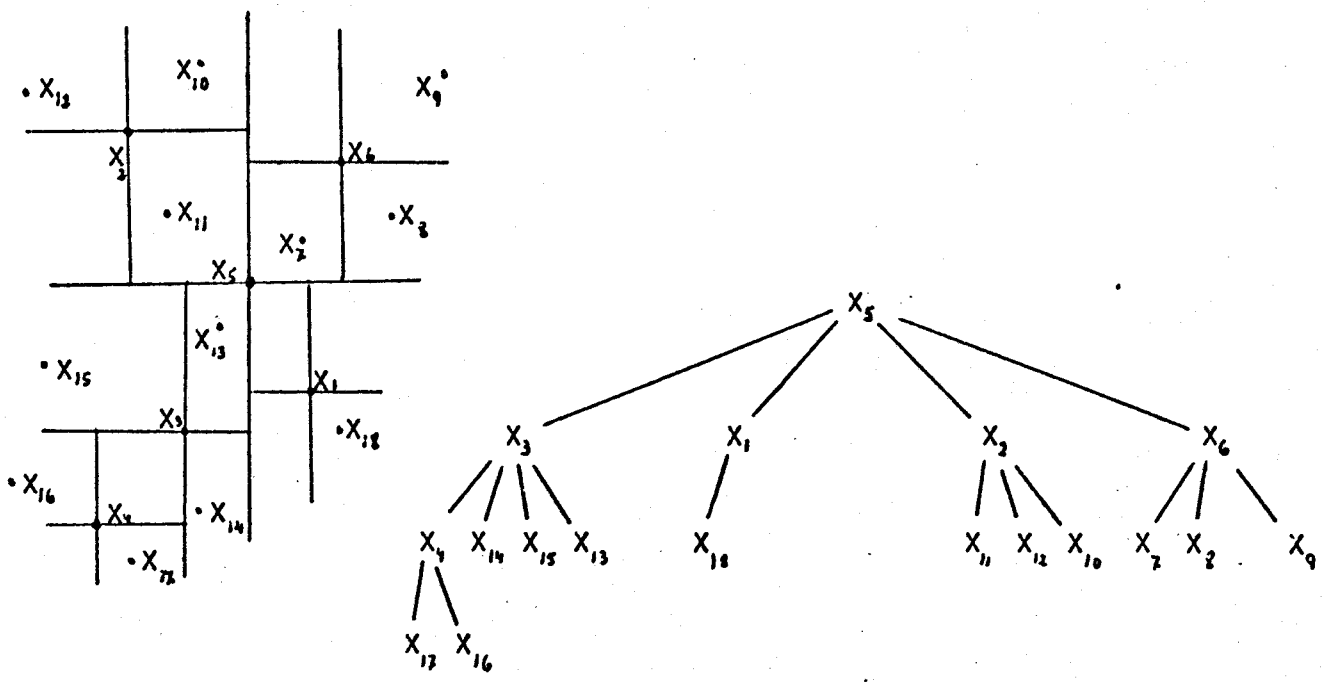


Figura 2

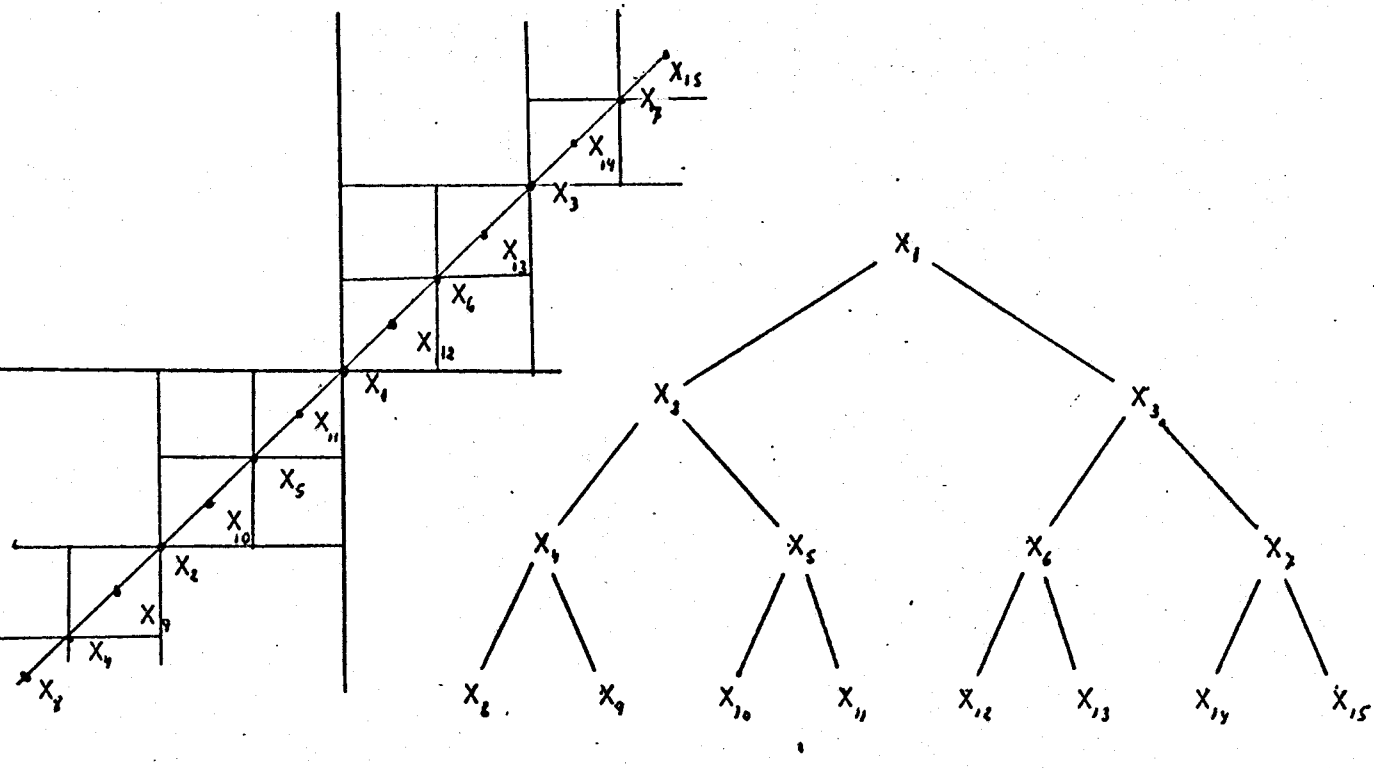


Figura 3

En segundo lugar, aunque es posible insertar puntos en un q-árbol (sin reequilibrado) es, a veces, muy difícil borrar un punto de la estructura (aún sin reequilibrado); por ejemplo, cuando borramos la raiz, es muy difícil tomar un nuevo punto como raiz que provoque la misma división, y en cualquier caso obligaría a reconstruir partes del q-árbol.

Para salvar estas dificultades modificaremos los q-árboles en pseudo q-árboles, que serán variantes de los primeros pero de búsqueda por hojas y que estudiaremos en el siguiente apartado. Mientras tanto, veremos algunas propiedades de los q-árboles.

En lo que sigue, supondremos que no existen dos puntos con alguna de sus coordenadas igúales, es decir, que para cualquier número real a y cualquier valor de i $1 \leqslant i \leqslant k$ hay a lo sumo un punto de X con coordenada i igual al valor a; esta condición es necesaria, ya que de lo contrario sería imposible elegir los puntos de la manera apropiada. Además los promedios de tiempo de inserción y borrado están referidos a un total de N actualizaciones sobre una estructura inicialmente vacía.

Veamos seguidamente como se realizan las inserciones y borrados en q-árboles; mientras que las inserciones no implicarán alteraciones en la estructura original, los borrados supondrán ciertas modificaciones.

Insertar un punto en un q-árbol es muy simple; debemos localizar el menor subcuadrante en el que se encuentra, por una sencilla búsqueda por el q-árbol. Localizado el subcuadrante, lo insertamos en él si es que se encuentra vacío; en otro caso usamos el punto ya existente como punto de división. El problema

que surge es conservar la altura del q-árbol del orden $\log_2 n$ a lo sumo; en el algoritmo de inserción dado, algunas inserciones exigirán un reequilibrado del q-árbol, lo cual es bastante costoso; veremos como para conseguir eficiencia tendremos que hacer algunas concesiones respecto a la exigencia en altura.

Teorema 1.- Sea ε un número fijo del intervalo abierto (0,1); existe un algoritmo que ejecuta N inserciones sobre una estructura inicialmente vacía, del tipo q-árbol tal que su altura es siempre menor o igual que $\log_{2-\varepsilon} n+O(1)$, y el tiempo promedio por inserción es $O(\log^2 N/\varepsilon)$ (n es el número de puntos de la estructura en un momento cualquiera).

Demostración: Para alcanzar la cota de altura de $\log_{2-\varepsilon} n+O(1)$ ponemos una condición aún más fuerte sobre el q-árbol, a saber, que para cualquier nodo interior con un total de n_0 puntos en sus subárboles, cualquier subárbol contiene a lo sumo $\lceil n_0/(2-\varepsilon)\rceil$ puntos. Tal q-árbol tiene siempre una altura máxima de $\log_{2-\varepsilon} n+O(1)$.

El método de inserción usará el hecho de que para cualquier configuración de n puntos se puede construir un q-árbol tal que para cualquier nodo interno con n_0 puntos en sus subárboles y cualquier subárbol contiene a lo sumo $\lceil n_0/2\rceil$ puntos, se puede construir en O(nlogn) tiempo.

Esto puede ser verificado por repetidas divisiones en cada subcuadrante tomando un punto cuya coordenada a lo largo de un eje es la mediana de todos los puntos de ese cuadrante. Cuando deseemos insertar un punto X determinaremos, en primer lugar, el subcuadrante al que pertenece; si no hay puntos en ese subcuadrante, insertaremos X en él, y en otro caso usaremos el

punto presente en ese subcuadrante como punto de subdivisión e insertaremos X en el apropiado subcuadrante vacío creado. En este último caso, es muy posible que el equilibrio del q-árbol haya sido perturbado; esto solo puede haber sucedido en el camino que va desde la raiz hasta el nuevo punto.

Si esto ocurre, habrá que determinar el nodo interno más alto X_j que está desequilibrado y reconstruiremos el subárbol completo bajo él, como un q-árbol perfectamente equilibrado según indicamos anteriormente.

Para obtener una cota del tiempo promedio del proceso, observemos un nodo interno X_j con n_0 puntos en sus subárboles partiendo del momento en que es construido. Para n_0 pequeño, acumulamos el tiempo de reconstrucción, que lleva O(1) en este caso, al tiempo de inserción que causa. Supondremos pues, que n_0 es bastante grande; sabemos que en el momento en que X_j es costruido, cada subárbol tiene a lo sumo:

$$\lceil n_0/2 \rceil \leqslant n_0/2 + 1/2$$

puntos; necesitamos equilibrar X_j cuando uno de sus subárboles contiene más de $1/(2-\varepsilon)$ veces el número total de puntos bajo X_j en ese momento.

Si esto sucede después de la inserción i-ésima en el subárbol X_j (desde el momento inicial o desde el último reequilibrado), entonces necesariamente:

$$\lceil n_0/2 \rceil + i > \lceil (n_0+i)/(2-\varepsilon) \rceil$$

y por tanto

$$i - i/(2-\varepsilon) > n_0/(2-\varepsilon) - n_0/2 - 1/2$$

de aquí se deduce facilmente que:

$$i > \varepsilon n_0/(2(1-\varepsilon)) - (2-\varepsilon)/(2(1-\varepsilon))$$

El costo total de reconstrucción del subárbol X_j para su reequilibrado es $O((i+n_0)\log(i+n_0))$; dividiendo este costo entre las inserciones, después de las cuales tiene lugar la reconstrucción queda un costo de $O((i+n_0)\log(i+n_0)/i)$ por inserción.

Para $n_0 > 3(2-\varepsilon)/\varepsilon$ se tiene:

$$\frac{i+n_0}{i}\log(i+n_0) = (1+\frac{n_0}{i})\log(i+n_0) <$$

$$(1 + \frac{n_0}{\frac{\varepsilon}{2(1-\varepsilon)}n_0 - \frac{2-\varepsilon}{2(1-\varepsilon)}})\log(i+n_0) <$$

$$(1 + \frac{3(1-\varepsilon)}{\varepsilon})\log(i+n_0) < \frac{3}{\varepsilon}\log(i+n_0)$$

de aquí que la cantidad de tiempo por inserción esté acotada por $O(\log(i+n_0)/\varepsilon)$; conviene hacer notar que los costos son cargados unicamente a las inserciones en el subárbol X_j que es el último recorstruido. De aqui que, ningún coste más pueda ser cargado en ese proceso por el reequilibrado del nodo y de cualquiera de sus nodos descendientes (cualquier nodo interno bajo X_j es también reequilibrado con X_j).

Solamente los nodos internos de nivel más alto (sobre el camino de X_j a la raiz) pueden cargar más costo a esas inserciones; sigue que cada inserción puede ser cargada a lo sumo una vez en cada nivel. La altura del q-árbol es a lo sumo:

$$\log_{2-\varepsilon} n + O(1) = \log_{2-\varepsilon} N + O(1) = \log N/\log(2-\varepsilon) + O(1)$$

como ε es una constante menor que 1, esto es O(logN). Así, en un proceso, el nivel es alcanzado a lo sumo en O(logN) tiempo y la carga es a lo sumo $O(\log N/\varepsilon)$ cada vez; de aquí que el tiempo promedio sea $O(\log^2 N/\varepsilon)$.

c.q.d.

Queda así probado que es posible efectuar inserciones en un q-árbol eficientemente, cuando se permita un pequeño crecimiento en altura; en general, el promedio de tiempo necesario para inserciones es probable que sea mucho menor que el indicado, porque los reequilibrados no son tan frecuentes cuando las distintas ramas crecen con análoga rapidez.

Aunque sobre los q-árboles pueden efectuarse inserciones eficientemente, los borrados presentan problemas; veremos como cambiando ligeramente la estructura podemos obtener también eficiencia en los borrados.

1.4.2.- Pseudo q-árboles

Los Pseudo q-árboles son muy parecidos a los q-árboles ordinarios; en los q-árboles, los puntos del conjunto X son usados como puntos de división del conjunto y así dividir el espacio E^k en cuadrantes y luego los cuadrantes en subcuadrantes y así sucesivamente. De esta forma, es muy posible que al borrar un punto X_i, cualquier otro punto que tomemos para reemplazarle divida el conjunto en partes muy diferentes de lo que lo hacía X_i.

Finkel y Bentley (1974) sugieren reinsertar en el q-árbol todos los descendientes del punto X_i eliminado; esto puede llevar cierto tiempo sin más que pensar en el caso de que borremos la raiz del q-árbol. Samet (1980) muestra para el caso de dos dimensiones que este número de reinserciones puede decrecer eligiendo el nuevo punto de división de una forma muy especial, pero la realidad es que si consideramos el peor de los casos, lleva aún más tiempo.

Tenemos pues, que el problema de borrar un elemento X_i de un q-árbol procede de la misión que tenemos asignada al punto, de dividir el espacio; para evitar esto, admitiremos que sirvan como puntos de división, puntos cualesquiera del espacio E^k sin que tengan que pertenecer al conjunto X, es decir, tomaremos como nodos internos puntos de E^k que dividirán E^k y por lo tanto el conjunto X; estos puntos arbitrarios dividirán el espacio en cuadrantes, los cuadrantes en subcuadrantes y así sucesivamente, hasta que cualquier subcuadrante contenga a lo sumo un punto del conjunto; los puntos del conjunto X ocuparán las hojas de la nueva estructura llamada pseudo q-árbol.

Por ejemplo, para el caso de dos dimensiones de la figura 4, los puntos $X_1, X_2, \ldots, X_{12}$ son los puntos del conjunto, e $Y_1, Y_2, \ldots Y_6$ los puntos de división.

Dada la similitud entre los pseudo q-árboles y los q-árboles ordinarios podemos hacer sobre ellos consultas de forma análoga a como lo hacíamos en los q-árboles; además, junto al hecho que veremos más adelante de que sobre un pseudo q-árbol pueden hacerse inserciones y borrados eficientemente, es interesante hacer notar que para un mismo conjunto de puntos pueden construirse pseudo q-árboles con menor altura que el

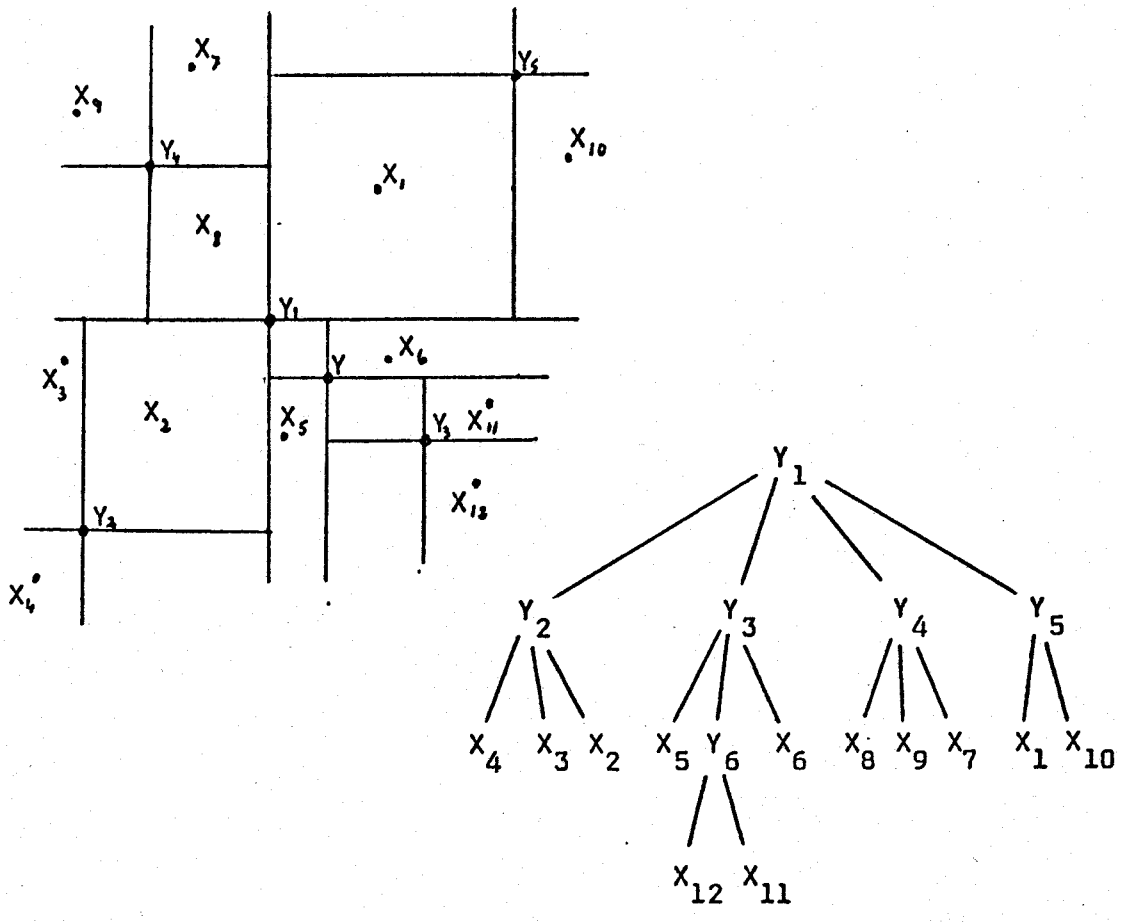


Figura 4

correspondiente q-árbol. Este resultado proviene del hecho de que podemos escoger como punto de división cualquier punto del espacio y así podemos tomar en cada etapa uno que divida el conjunto de la forma más equilibrada.

Teorema 2.- Dado un conjunto de puntos $X = \{X_1, X_2, \ldots, X_n\}$ del espacio E^k, existe un punto de división $Y(y_1, y_2, \ldots, y_k)$ tal que cualquier cuadrante inducido por Y contiene a lo sumo $\lceil n/(k+1) \rceil$ puntos de X.

Demostración: Podemos elegir la coordenada y de modo que $\lceil n/(k+1) \rceil$ puntos del conjunto, tengan la primera coordenada menor que y y el resto de los puntos $\lfloor nk/(k+1) \rfloor$ tienen su primera coordenada mayor que y_1; tal y existe ya que hemos asumido que no hay dos puntos con alguna de sus coordenadas iguales.

Hemos determinado así un hiperplano (puntos con primera coordenada igual a y_1) que separa X en dos partes, una con $\lceil n/(k+1) \rceil$ puntos y otra con $\lfloor nk/(k+1) \rfloor$ puntos; quedan por determinaraún k-1 coordenadas de Y; para determinar el valor de y_2 observamos unicamente los puntos cuya primera coordenada es mayor que y_1, ya que la otra parte del conjunto es lo bastante pequeña y no importa como la dividirán los siguientes hiperplanos.

Consideremos así, el conjunto formado por $\lfloor nk/(k+1) \rfloor$; elegimos y_2 de manera que $\lceil n/(k+1) \rceil$ de esos puntos tienen su segunda coordenada menor que y_2; continuamos con la porción $\lfloor n(k-1)/(k+1) \rfloor$ que tiene su segunda coordenada mayor que y_2 (y primera coordenada mayor que y_1) y elegimos y_3 repitiendo el mismo razonamiento; al final elegimos y_k de manera que divida un conjunto con $\lfloor 2n/(k+1) \rfloor$ puntos, en dos partes cada una con un número de puntos menor o igual que $\lceil n/(k+1) \rceil$, con lo cual termina el proceso.

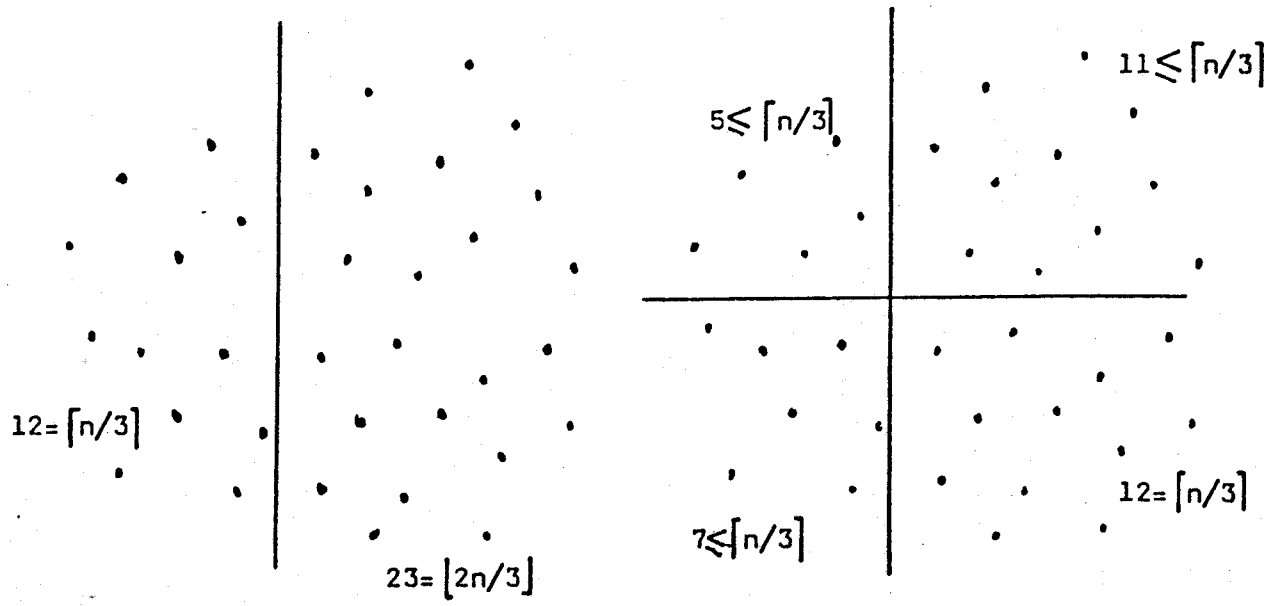
Así, hemos obtenido el punto $Y(y_1, y_2, \ldots, y_k)$, que divide el espacio en cuadrantes cada uno conteniendo a lo sumo $\lceil n/(k+1) \rceil$ puntos de X.

c.q.d.

Por ejemplo, en el caso de dos dimensiones con 35 puntos repartidos como indica la figura 5, la selección de $Y(y_1, y_2)$ sería la indicada.

Corolario: Dado el conjunto $X = \{ X_1, X_2, \ldots, X_n \}$ de puntos de E^k, existe un pseudo qárbol para X con altura máxima $\lceil \log_{k+1} n \rceil$.

Demostración: Aplicando el teorema anterior al conjunto X, existe un punto Y que divide el espacio en cuadrantes conteniendo



Elección de y_1 Elección de y_2

Figura 5

cada uno $\lceil n/(k+1) \rceil$ puntos como máximo; tomemos Y como raiz del pseudo q-árbol y procedamos de la misma forma con cada cuadrante. Si en un cuadrante quedan n_0 puntos existe un punto de E^k que lo divide en subcuadrantes conteniendo cada uno de ellos $\lceil n_0/(k+1) \rceil$ puntos a lo sumo. Dichos puntos de división serán los hijos de Y; es importante ver que cada uno de los nuevos cuadrantes contiene como máximo:

$$\lceil n_0/(k+1) \rceil \leq \lceil 1/(k+1) \lceil n/(k+1) \rceil \rceil = \lceil n/(k+1)^2 \rceil$$

Aplicando este procedimiento reiteradamente tenemos el pseudo q-árbol deseado; como el número de puntos en los subcuadrantes de nivel h está acotado por $\lceil n/(k+1)^h \rceil$ y denotando por H la máxima altura alcanzable del pseudo q-árbol, en el nivel H habrá como máximo un punto por subcuadrante y por lo tanto:

$$\lceil n/(k+1)^H \rceil = 1 \quad ===> \quad n/(k+1)^H \leqslant 1 \quad ===$$

$$n \leqslant (k+1)^H \quad ===> \quad \log_{k+1} n \leqslant H$$

y en el nivel anterior:

$$1 < \lceil n/(k+1)^{H-1} \rceil \quad ===> \quad n/(k+1)^{H-1} > 1 \quad ===$$

$$n > (K+1)^{H-1} \quad ===> \quad \log_{k+1} n > H-1$$

luego

$$H-1 < \log_{k+1} n \leqslant H$$

de donde $H = \lceil \log_{k+1} n \rceil$ es una cota superior de la altura.

c.q.d.

Aunque $\lceil \log_{k+1} n \rceil$ es una cota superior de altura, ésta puede ser alcanzada como demuestra el siguiente teorema.

Teorema 3.- Existe un conjunto $X = \{X_1, X_2, \ldots, X_n\}$ de n puntos de E^k tal que no existe un pseudo q-árbol para tal conjunto, con altura menor que $\lceil \log_{k+1} n \rceil$.

Demostración: Tomemos los puntos $X_1, X_2, \ldots, X_n$ tales que si

$$X_i = (x_{i1}, x_{i2}, \ldots, x_{ik})$$

se verifique que $x_{ij} < x_{i+1j}$ para cualquier valor de i,j enteros en los intervalos $[1,n]$ y $[1,k]$ respectivamente, es decir si:

$$X_1(x_{11}, x_{12}, \ldots, x_{1k})$$
$$X_2(x_{21}, x_{22}, \ldots, x_{2k})$$
$$\text{----------}$$
$$\text{----------}$$
$$X_n(x_{n1}, x_{n2}, \ldots, x_{nk})$$

se verifica:

$$x_{11} < x_{21} \text{ --- } < x_{n1}$$
$$x_{12} < x_{22} \text{ --- } < x_{n2}$$
$$\text{----------}$$
$$\text{----------}$$
$$x_{1k} < x_{2k} \text{ --- } < x_{nk}$$

existe por lo tanto, una dominancia absoluta de cada punto respecto de los anteriores.

Consideremos cualquier pseudo q-árbol para éste conjunto; si Y es su raiz, divide el conjunto en 2^k cuadrantes; probaremos que unicamente k+1 de estos cuadrantes pueden contener puntos del conjunto; para ello recorramos los puntos desde X_1 a X_h ; al pasar de un cuadrante a otro cruzamos uno de los hiperplanos inducidos por Y, y dado que los puntos que siguen tienen coordenadas mayores, nunca más, en el recorrido de X_1 a X_n, volveremos a cruzar dicho hiperplano. De aquí, que como Y define k hiperplanos, podemos cambiar de cuadrante k veces a lo sumo, por lo que como máximo habrá k+1 cuadrantes ocupados.

De lo anterior sigue, que el menor de esos cuadrantes contiene un número de puntos mayor o igual que $\lceil n/(k+1) \rceil$; además, por la misma razón, los puntos en este cuadrante deben originar los subcuadrantes con al menos $\lceil n/(k+1) \rceil$ puntos. Repitiendo este argumento, se tiene un pseudo q-árbol con altura al menos de $\lceil \log_{k+1} n \rceil$.

c.q.d.

Un ejemplo en dos dimensiones es el expresado en la fig. 6.

Veamos como la estructura pseudo q-árbol puede ser

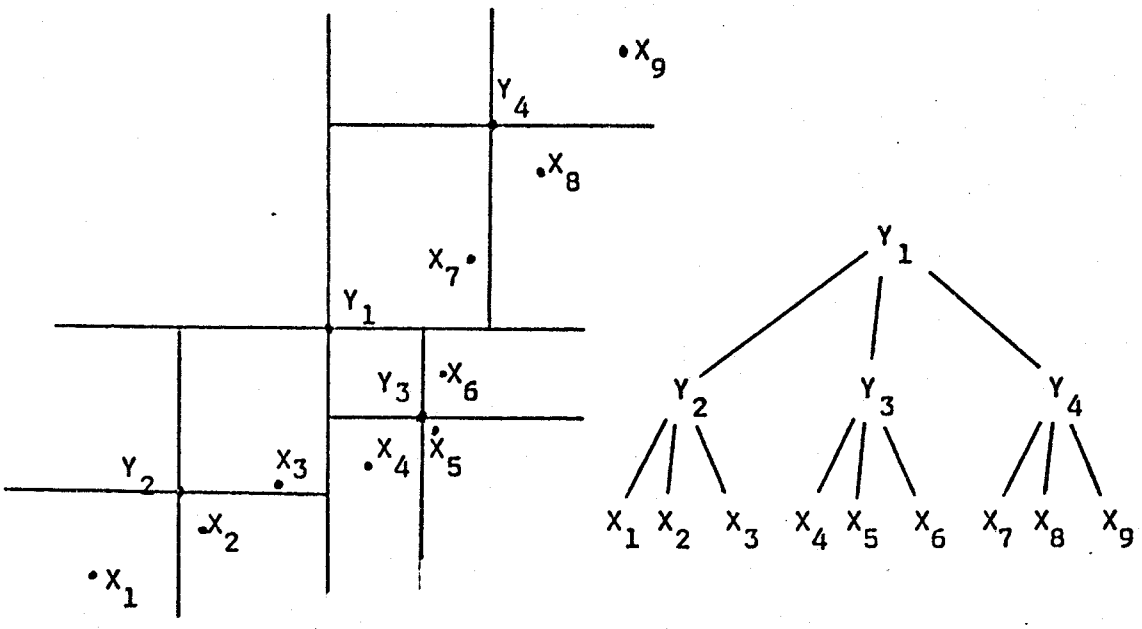


Figura 6

construida eficientemente.

Teorema 4.- Dado un conjunto de n puntos $X = \{X_1, X_2, \ldots, X_n\}$ del espacio E^k, un podemos construir un pseudo q-árbol para ese conjunto con altura máxima $\lceil \log_{k+1} n \rceil$ en tiempo O(nlogn).

Demostración: Usando el teorema 2 y su corolario, la elección del punto Y que sirve como raiz, consiste en encontrar k veces el i-ésimo elemento de un conjunto con respecto a un orden, lo cual lleva un tiempo O(n) (Wirth,1980); la división del conjunto en cuadrantes lleva O(n) también; luego el primer nivel del pseudo q-árbol lleva O(n) tiempo.

La división del cuadrante que contiene n_0 puntos lleva $O(n_0)$ por idéntico razonamiento; como todos los cuadrantes juntos contienen n puntos, el total de tiempo necesario para el segundo nivel está acotado por O(n).

Este mismo argumento sirve para cada nivel del pseudo q-árbol; puesto que la altura está acotada por $\lceil \log_{k+1} n \rceil$, la cota

de tiempo O(nlogn) es válida para la estructura total.

c.q.d.

En general, los pseudo q-árboles tienen menor altura que su correspondiente q-árbol ya que para cada conjunto de puntos puede construirse un pseudo q-árbol con altura máxima una unidad mayor que el mejor posible q-árbol.

Veamos a continuación como efectuar la inserción y borrado de elementos en un pseudo q-árbol siempre que no se exija reequilibrados en el árbol; los algoritmos que seguiremos serán los siguientes:

a.- Inserción de un punto X_i

i.- Buscamos el subcuadrante al que pertenece X_i

ii.- Si no existe ningún punto del conjunto presente en dicho subcuadrante, insertaremos allí X_i añadiendo una hoja. Si ya existe un punto X_j tomaríamos un punto Y del segmento $X_i X_j$ y lo usaríamos como nuevo punto de división. Sustituimos X_j por Y y añadimos X_i y X_j como hojas apropiadas de Y.

b.- Borrar un punto X_i

i.- Buscamos X_i en el pseudo q-árbol

ii.- Cuando X_i es encontrado borrar esa hoja (si X_i no existe, terminar)

iii.- Si al borrar X_i, su antecesor inmediato Y tiene un solo descendiente directo X_j, reemplazar Y por X_j.

En ambos casos el tiempo empleado es del orden de la altura del pseudo q-árbol, aunque no hay garantía de que el equilibrio permanezca.

Para mantener un pseudo q-árbol equilibrado aplicaremos las

técnicas de la reconstrucción parcial; sea ε una constante $0 < \varepsilon < k$; para cada nodo α con n_α puntos, exigiremos que cada subárbol de tenga como máximo $\lceil n_\alpha/(k+1-\varepsilon) \rceil$ puntos del conjunto; facilmente puede verificarse que su altura es a lo sumo $\lceil \log_{k+1-\varepsilon} n_\alpha \rceil + O(1)$, donde $O(1)$ es la constante correspondiente a la altura del nodo .

Si elegimos ε próximo a cero, la cota de altura se aproxima a la cota óptima $\lceil \log_{k+1} n_\alpha \rceil$ según se vio anteriormente; diremos que tiene un equilibrio perfecto si cada subárbol de tiene a lo sumo $\lceil n_\alpha/(k+1) \rceil$ puntos del conjunto; cuanto más se aproxime ε a k, la exigencia de equilibrio es menor, y si $\varepsilon = k$ la distribución es libre.

Teorema 5.- Para cualquier número fijo ε con $0 < \varepsilon < k$ existe un algoritmo que ejecuta N inserciones y borrados en un pseudo q-árbol inicialmente vacío, tal que su altura es a lo sumo $\lceil \log_{k+1-\varepsilon} n \rceil + O(1)$ (siendo n el número de puntos presentes en el pseudo q-árbol) y el tiempo medio por actualización está acotado por $O(\log^2 N/\varepsilon)$.

Demostración: Supongamos que un nodo α está en equilibrio perfecto con n'_α puntos; supongamos además que hemos hecho Ni inserciones y Nb borrados, antes de que el nodo α pierda el equilibrio (el más alto de los que pierden el equilibrio con la última actualización) y sea $n_\alpha = n'_\alpha + Ni - Nb$, es decir, el número de puntos bajo el nodo en el momento en que pierde el equilibrio, es n_α.

La pérdida de equilibrio se debe a que uno de sus subárboles tiene más de $\lceil n_\alpha/(k+1-\varepsilon) \rceil$ puntos; como el tamaño de cada subárbol era menor o igual a $\lceil n'_\alpha/(k+1) \rceil$ cuando α estaba en equilibrio

perfecto, ningún subárbol puede ser mayor de $n'_\alpha/(k+1) + Ni$ y por lo tanto:

$$\lceil n'_\alpha/(k+1)\rceil + Ni > \lceil n'_\alpha/(k+1-\varepsilon)\rceil \quad ==>$$

$$\lceil (n_\alpha - Ni + Nb)/(k+1)\rceil + Ni > \lceil n_\alpha/(k+1-\varepsilon)\rceil \quad ==>$$

$$n_\alpha/(k+1) + kNi/(k+1) + Nb/(k+1) + 1 > n_\alpha/(k+1-\varepsilon) \quad ==>$$

$$Ni + Nb > \varepsilon\, n_\alpha/((k+1)(k+1-\varepsilon)) - 1$$

El coste total para reconstruir α es $O(n_\alpha \log n_\alpha)$; dividiendo este coste por el número de actualizaciones hechas Ni + Nb nos resulta que el coste por actualización de α es $O(n_\alpha \log n_\alpha/(Ni+Nb))$.

Asumiendo que n es bastante grande y teniendo en cuenta la identidad:

$$\frac{a}{ba-1} = \frac{1}{b} + \frac{a/b}{ba-1} \quad \text{tomando } a=n \text{ y } b=\frac{\varepsilon}{(k+1)(k+1-\varepsilon)}$$

se tiene:

$$\frac{n}{Ni+Nb}\log n_\alpha < \frac{n_\alpha}{\dfrac{\varepsilon}{(k+1)(k+1-\varepsilon)}n_\alpha - 1}\log n =$$

$$\left(\frac{(k+1)(k+1-\varepsilon)}{\varepsilon} + \frac{\dfrac{(k+1)(k+1-\varepsilon)}{\varepsilon}}{\dfrac{\varepsilon}{(k+1)(k+1-\varepsilon)}n_\alpha - 1}\right)\log n_\alpha$$

$$\left(\frac{(k+1)^2 - \varepsilon(k+1)}{\varepsilon} + 1\right)\log n_\alpha = \left(\frac{(k+1)^2}{\varepsilon} - \varepsilon - 1 + 1\right)\log n_\alpha <$$

$$\frac{(k+1)^2}{\varepsilon}\log n_\alpha$$

Luego cada actualización de α tiene un costo medio acotado por $(k+1)^2 \log n_\alpha/\varepsilon$ y por lo tanto acotado por $(k+1)^2\log N/\varepsilon$; como esta expresión no depende del nodo, el costo medio de una actualización está acotado por $(k+1)^2 \log N/\varepsilon$ por lo que dicho costo es $O(\log N/\varepsilon)$. Con esto, todos los subárboles de α quedan equilibrados, pero en la reconstrucción de α cada nodo debe recorrer los nodos desde la raiz hasta α, lo que añade un costo para cada nodo a la reconstrucción, del orden de la altura del árbol que es a lo sumo:

$$\log_{k+1-\varepsilon} n + O(1) \leqslant \log_{k+1-\varepsilon} N + O(1) =$$
$$\log N/\log(k+1-\varepsilon) + O(1) = O(\log N)$$

Luego el costo promedio viene dado por:

$$O(\log N/\varepsilon) * (O(\log N) = O(\log^2 N/\varepsilon)$$

c.q.d.

En general, el tiempo medio es mucho más pequeño que el indicado porque:

a.- Este tiempo depende del número n de puntos que tenga el árbol en cada momento y no de N que representa el número total de actualizaciones.

b.- Cuando el crecimiento de los subárboles se mantiene con cierto reparto normal, no hay necesidad de reequilibrados en la mayoría de las actualizaciones.

De todo ello se deduce que el pseudo q-árbol es una estructura más potente que el q-árbol para llevar a cabo actualizaciones, aunque como veremos más adelante, al aplicarla a la búsqueda de vectores eficientes pierde su potencia con respecto a los q-árboles.

1.4.3.- Epq-árboles

En los teoremas desarrollados para los pseudo q-árboles hemos tenido que suponer que no existían dos puntos con el mismo valor en la misma coordenada, ya que, de no ser así, no podemos garantizar la división del conjunto en las partes deseadas; por ejemplo, si todos los puntos se encuentran sobre una misma recta paralela a uno de los ejes, cualquier punto de división partirá el conjunto en dos cuadrantes no vacíos como máximo. Los epq-árboles (Extended Pseudo Quad Tree) son una modificación de los pseudo q-árboles para que podamos eliminar la restricción, hasta ahora mantenida sobre los puntos del conjunto, manteniendo la misma eficiencia tanto en consultas como en actualizaciones.

Recordemos que un pseudo q-árbol se construía dividiendo el espacio E^k, eligiendo un punto arbitrario Y y dividiendo el conjunto X en 2^k cuadrantes.

Con la restricción original de que no existan dos puntos con el mismo valor en la misma coordenada, la elección de Y no presenta problema, pero si suprimimos esta condición aparecen algunos. Por lo tanto, vamos a tratar independientemente los puntos que se encuentran sobre los hiperplanos separadores, de los que se encuentran en los cuadrantes abiertos que formarán una

división normal del pseudo q-árbol.

Los k hiperplanos determinados por un nodo Y son k subespacios (k-1)-dimensionales; todos ellos se cortan dos a dos en $\binom{k}{2}$ espacios (k-2)-dimensionales; a su vez, se cortan de tres en tres en $\binom{k}{3}$ espacios (k-3)-dimensionales y asi sucesivamente, cada grupo de k-1 hiperplanos se cortan sobre $\binom{k}{k-1}$ rectas y, finalmente, todos se cortan en el punto Y.

El espacio queda así dividido por el punto Y en 2^k cuadrantes abiertos y un ńumero de subespacios dado por:

$$\binom{k}{1} + \binom{k}{2} + \text{-------+} \binom{k}{k-1} + \binom{k}{k} = 2^k - 1$$

En el epq-árbol el punto Y elegido es la raiz, y sus descendientes directos serían:

a.- Un punto perteneciente a cada uno de los cuadrantes abiertos

b.- Un punto perteneciente a cada uno de los subespacios anteriores (el de dimensión mínima que lo contenga)

Los puntos del apartado a, generarían epq-árboles de dimensión k mientras que los del apartado b, generarín epq-árboles de dimesión menor, exactamente con la dimensión correspondiente al subespacio al que pertenece.

Por ejemplo, para el caso de dos dimensiones, exactamente dos líneas y un punto deberán ser añadidos a cualquier nodo interno, es decir, dos 1-dimensionales epq-árboles y un 0-dimesional epq-árbol. Téngase en cuenta que un epq-árbol 1-dimensional es un árbol binario equilibrado, y un 0-dimensional epq-árbol es un punto.

Teorema 6.- Dado un conjunto de puntos $X = \{X_1, X_2, \ldots, X_n\}$ de E^k se puede construir un epq-árbol con los puntos de X de altura menor o igual que $\lceil \log n \rceil + k$ en $O(n \log n)$ pasos.

Demostración: Designemos como punto de división un punto $Y(y_1, y_2, \ldots, y_k)$ tal que a lo sumo la mitad de los puntos tienen la primera coordenada estrictamente menor que y_1, y como máximo la mitad de los puntos de X tienen la coordenada y estrictamente mayor que y_1. Tal punto Y existe, aunque puede que no pertenezca al conjunto y además puede ocurrir que muchos puntos tengan la primera coordenada igual a y_1. Estos puntos, junto con aquellos otros que tienen alguna otra coordenada coincidente con la correspondiente de Y, deben ser distribuidos sobre la estructura apropiada asociada con Y, tal como dijimos anteriormente. Sea $Z(z_1, z_2, \ldots, z_k)$ uno de tales puntos; la estructura a la cual pertenece Z está perfectamente determinada por aquellas coordenadas y_i de Y que son iguales a las correspondientes z_i de Z.

Esto puede ser detectado en $O(k)$ pasos; después que el punto Y es determinado, lo cual necesita $O(n)$, podemos determinar para cualquier punto X_i el cuadrante en el que se encuentra o la estructura a la que pertenece, en $O(k)$ pasos.

Por lo tanto, el costo total por división del conjunto con el punto Y será $O(kn) = O(n)$; seguidamente dividiríamos cada cuadrante y construiríamos las estructuras asociadas para los nuevos puntos.

De esta forma, la construcción de cualquier nivel en la estructura total, lleva $O(n)$; es inmediato ver que después de cada división, cada cuadrante contiene como máximo la mitad de

los puntos; es posible, que más de la mitad de los puntos vayan en una estructura asociada, lo cual puede suceder k veces como máximo, ya que en cualquier caso, la dimensión decrece al menos una unidad.

De aquí que la altura total del epq-árbol sea a lo sumo de $\lceil \log n \rceil + k$; además, la construcción de la estructura lleva $O(n(\log n+k))$, y como k es fijo, esta cantidad coincide con $O(n \log n)$.

c.q.d.

Los resultados prácticos son, en general, mejores que los obtenidos por el teorema anterior, pero en el peor de los casos, como ocurre en R^2 cuando los puntos están sobre una recta vertical, no podemos mejorar la altura $\lceil \log n \rceil + k$.

Las consultas sobre los epq-árboles, son algo más complejas que para los pseudo q-árboles; por ejemplo, en una consulta tipo rango, hay que comenzar por la raiz Y; comparando Y con el rango, determinaremos los cuadrantes que pudieran interceptar dicho rango, pero en lugar de continuar por cada uno de los subcuadrantes, debemos también determinar, si el rango corta a los subespacios asociados y debemos llevar a cabo una consulta de rango sobre las estructuras asociadas, usando la restricción del rango inicial a los subespacios considerados.

Cuando la configuración de puntos satisface la restricción que exigíamos para pseudo q-árboles ordinarios, entonces los subespacios contienen como máximo un punto y el tiempo necesario para realizar una consulta sobre un epq-árbol es el mismo que sobre los pseudo q-árboles ordinarios. En cambio, si el conjunto X no satisface dicha restricción, la búsqueda sobre los

subespacios lleva un tiempo adicional, pero puesto que el número de subespacios a sondear es menor que el número de cuadrantes que deberán ser sondeados, el tiempo total es del mismo orden que para los pseudo q-árboles (Bentley y Stanat,1975).

Así pues, la búsqueda de rango sobre los epq-árboles, es de complejidad comparable a la de los pseudo q-árboles ordinarios.

Esta estructura también tiene la propiedad de que cualquier subestructura es un epq-árbol, por lo que puede ser construida dinamicamente, en forma recursiva.

Teorema 7.- Dado un número fijo ε del intervalo (0,1), existe un algoritmo que ejecuta N actualizaciones sobre un epq-árbol inicialmente vacío tal que su altura es como máximo $\log_{2-\varepsilon} n+k+O(1)$ (siendo n el número de puntos presentes en el epq-árbol) y en un tiempo medio por actualización acotado por $O(\log^2 N/\varepsilon)$.

Demostración: Para insertar o borrar un elemento X_i determinaremos, en primer lugar, sobre el árbol principal o sobre las estructuras asociadas, donde el elmento X_i debe ser insertado o borrado y ejecutaremos la acción correspondiente. Es posible, que en algún lugar del camino desde X_i a la raiz del árbol principal, el equilibrio haya sido perturbado; debemos determinar en este caso, el nodo más alto que ha perdido el equilibrio y reconstruir el epq-árbol con raiz en dicho nodo. Esto lleva $O(n_0 \log n_0)$ siendo n_0 el número de puntos bajo el nodo desequilibrado. Si cargamos el costo de reconstrucción, sobre el número de actualizaciones, llegamos a que, por este concepto, se tiene un costo de $O(\log n/\varepsilon) \leqslant O(\log N/\varepsilon)$.

Este costo, unido al necesario para recorrer cada nodo desde

la raiz del árbol principal hasta su punto de inserción, lo cual es menor que O(logn) nos lleva a que el tiempo medio por actualización está acotado por $O(\log^2 N/\varepsilon)$.

c.q.d.

Esta generalización de los pseudo q-árboles a los epq-árboles no aumenta la cantidad de almacenamiento requerido (salvo una fracción mínima) por lo que constituye una estructura ventajosa, ya que admite configuraciones más generales, según hemos visto.

1.4.4.- Kd-árboles

La estructura kd-árbol que estudiamos a continuación fue introducida por Bentley(1975) para resolver diversos problemas de búsqueda multidimensionales; para nosotros es de importancia fundamental debido a que, como veremos más adelante, es la estructura de mayor eficiencia en el problema de la búsqueda de los vectores dominantes.

Dado un conjunto $X = \{ X_1, X_2, \ldots, X_n \}$ de puntos de E^k, para construir un kd-árbol con dicho conjunto, comenzamos eligiendo como raiz un punto arbitrario X_i perteneciente al conjunto X; el punto elegido X_i dividirá el espacio E^k en dos partes, según el valor de la primera coordenada, por lo tanto dividirá el conjunto X en dos partes A y B que supondremos disjuntas, es decir, volvemos a partir de la hipótesis de que en X no existen dos puntos con alguna de sus coordenadas idénticas. Así pues, X_i será la raiz del kd-árbol, y los dos conjuntos A y B, serán sus

subárboles.

Un ejemplo en R^2 lo podemos observar la figura 7.

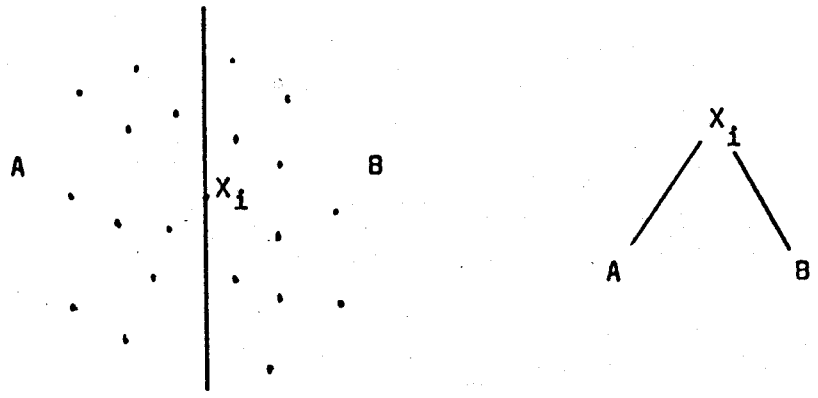


Figura 7

Repetiremos el razonamiento sobre los conjuntos A y B, es decir, en ambos subconjuntos elegimos un punto y los dividimos en dos subconjuntos de análoga forma, pero esta vez con respecto a su segunda coordenada; igualmente, lo haremos con respecto a la tercera, cuarta, etc., hasta llegar a la división con respecto a la coordenada k-ésima, a partir de la cual volveremos a dividir los subconjuntos otra vez por la primera coordenada.

La figura 8 nos muestra un ejemplo en R^2.

Los kd-árboles son árboles binarios y pueden construirse con altura óptima, es decir, de altura $\lceil \log_2 n \rceil$ en O(nlogn), tomando siempre la mediana con respecto a la coordenada de división que estará perfectamente determinada, ya que hemos supuesto que no existen dos puntos con igual valor en alguna de sus coordenadas.

Puesto que cualquier subestructura de un kd-árbol es también un kd-árbol, éstos pueden construirse dinámicamente de forma recursiva; tanto la búsqueda como la inserción de un elemento se lleva a cabo como en cualquier árbol binario, sin embargo, para

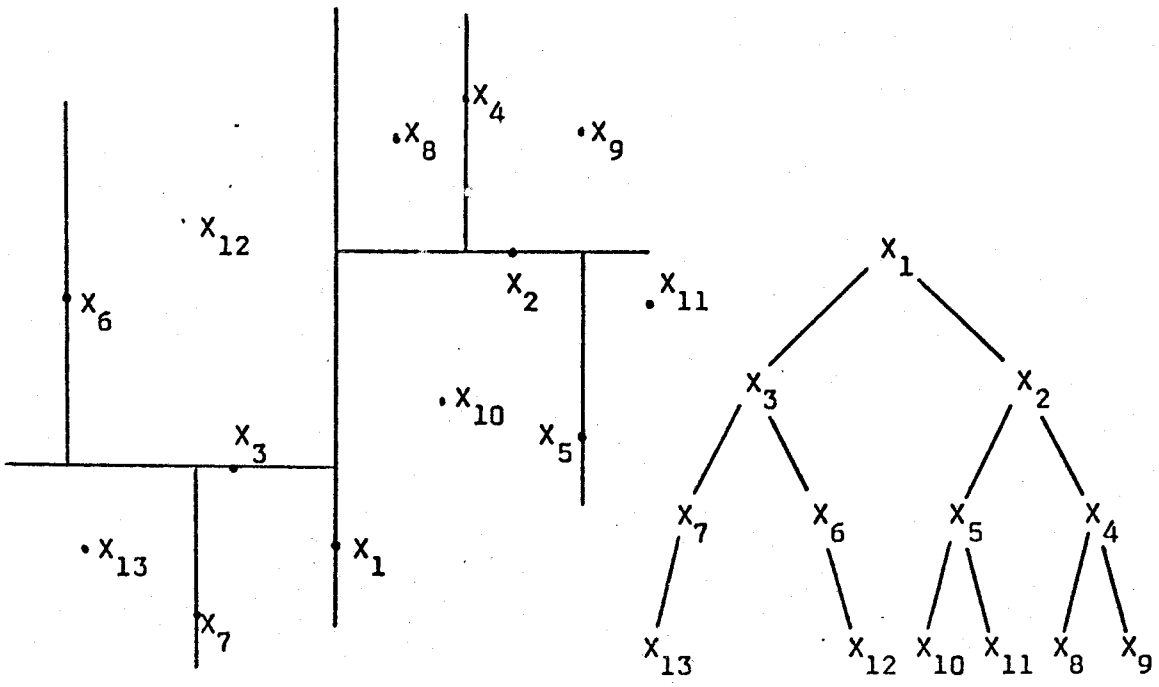


Figura 8

borrar un elemento es necesario reconstruir el subárbol que lo tiene como raiz.

De forma similar a lo que ocurría con los q-árboles, para salvar esta reconstrucción se crea una variante de los kd-árboles llamada pseudo kd-árbol, que serán árboles binarios de búsqueda por hojas; en ellos los puntos de división no tienen que pertenecer al conjunto y se verifican propiedades análogas a los pseudo q-árboles respecto de los q-árboles. Por ejemplo, el teorema siguiente:

Teorema 7.- Dado un número fijo ε en el intervalo (0,1), existe un algoritmo que ejecuta N inserciones en un kd-árbol inicialmente vacío tal que su máxima altura es $\log_{2-\varepsilon} n+O(1)$, siendo n el número de puntos presentes en la estructura, y el tiempo medio por inserción está acotado por $O(\log^2 N/\varepsilon)$.

La demostración es análoga a la del teorema correspondiente para q-árboles.

Como ya hemos dicho, borrar en un kd-árbol así construido puede ser bastante costoso, así que modificaremos la estructura a los pseudo q-árboles.

La figura 9 muestra un ejemplo en R^2 de un pseudo kd-árbol para el conjunto $X = \{X_1, X_2, \ldots, X_{11}\}$, usando como puntos de división $Y_1, Y_2, \ldots, Y_{10}$.

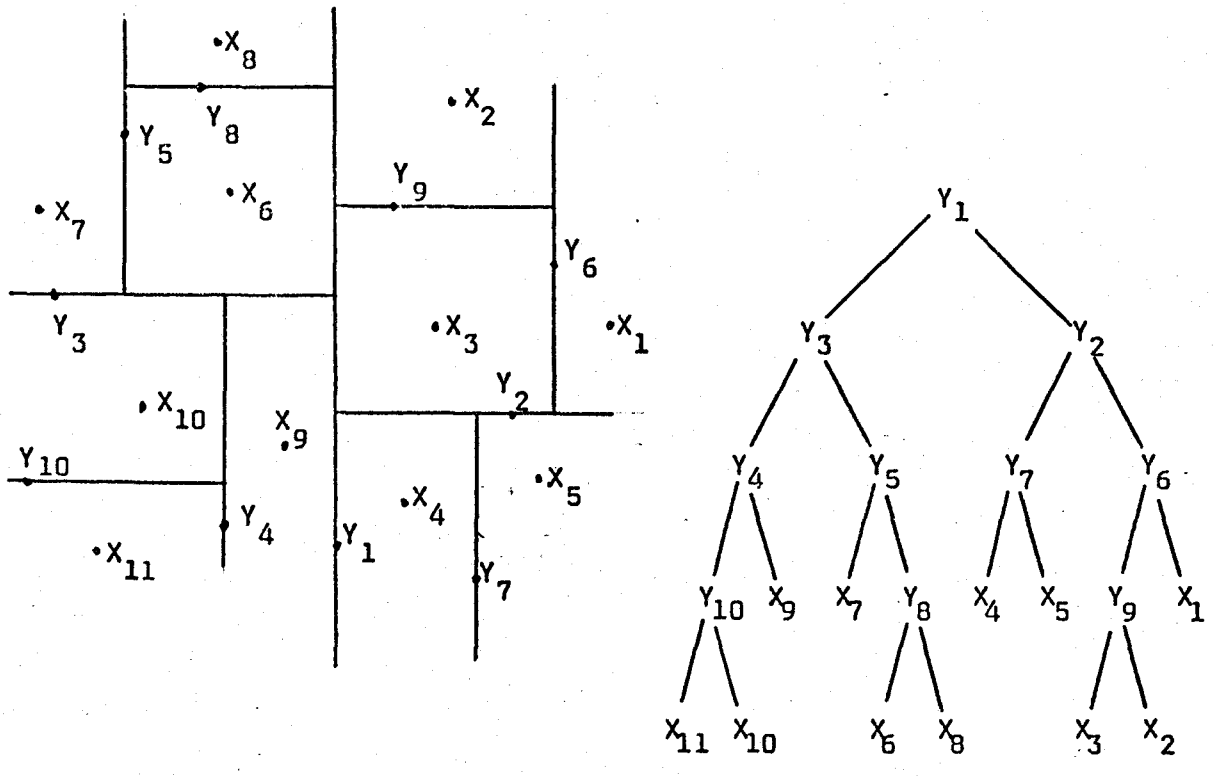


Figura 9

Además, puesto que de los puntos de división Y_i solo necesitamos una de sus coordenadas, no es indispensable que sean necesariamente puntos, sino que su papel podría ser desempeñado por un número real; esta ventaja que disminuiría notablemente la cantidad de almacenamiento requerido, tiene la dificultad de que

su dinamización es más complicada, ya que la estructura soportaría dos tipos de nodos distintos, con lo que la recursividad no sería tan directa.

Siguiendo el mismo camino que para los kd-árboles ordinarios, se puede construir un pseudo kd-árbol óptimo, es decir, un pseudo kd-árbol con altura $\lceil \log_2 n \rceil$ en un tiempo O(nlogn); la única particularidad es que no se tomará la mediana como punto de división, sino algún punto entre la mediana y los puntos más próximos a ella; por lo tanto un pseudo kd-árbol gozará de las mismas propiedades que los kd-árboles ordinarios, pero tiene la ventaja de que suprimimos de los puntos del conjunto la función de división del espacio.

Vemos a continuación, como es posible borrar puntos en un pseudo kd-árbol eficientemente; Willard(1978) ha desarrollado una dinamización de los pseudo kd-árboles (a los cuales llama kd*-árboles) basándose en las posibilidades de descomposición de los problemas que trata, usando bosques de pseudo kd-árboles de diferentes tamaños (Overmars y Leewen,1981). Esta línea de dinamización tiene la desventaja de que el tiempo de consulta tiende a crecer por un factor multiplicativo O(logn).

No obstante, aquí lo haremos, enunciando un teorema análogo al de los pseudo q-árboles.

Teorema 8.- Dado un número ε del intervalo (0,1), existe un algoritmo que ejecuta N actualizaciones sobre un pseudo kd-árbol inicialmente vacío, tal que su altura es como máximo $\log_{2-\varepsilon} n+O(1)$ y el tiempo medio por actualización es $O(\log^2 n/\varepsilon)$.

La demostración es similar a su análogo para pseudo q-árboles.

Tenemos así probado que un pseudo kd-árbol es una estructura dinámica de datos eficiente; aún nos falta eliminar la restricción de que no existan dos puntos en X con igual valor en alguna de sus coordenadas.

Esto puede evitarse creando los pseudo kd-árboles extendidos; cuando dividimos el conjunto con un nodo Y, lo que hacemos es elegir un hiperplano con respecto a una de sus coordenadas; así, la única estructura que debemos añadir a cada nodo interno es la correspondiente a los puntos que se encuentran sobre el hiperplano de división.

Un pseudo kd-árbol extendido es un pseudo kd-árbol, en el cual cualquier nodo interno Y tiene asociado un pseudo kd-árbol con los puntos que se encuentran sobre el hiperplano división originado por Y.

Un 1-dimensional pseudo kd-árbol extendido se define como un árbol binario equilibrado de búsqueda ordinario.

La figura 10 es un ejemplo en R^2:

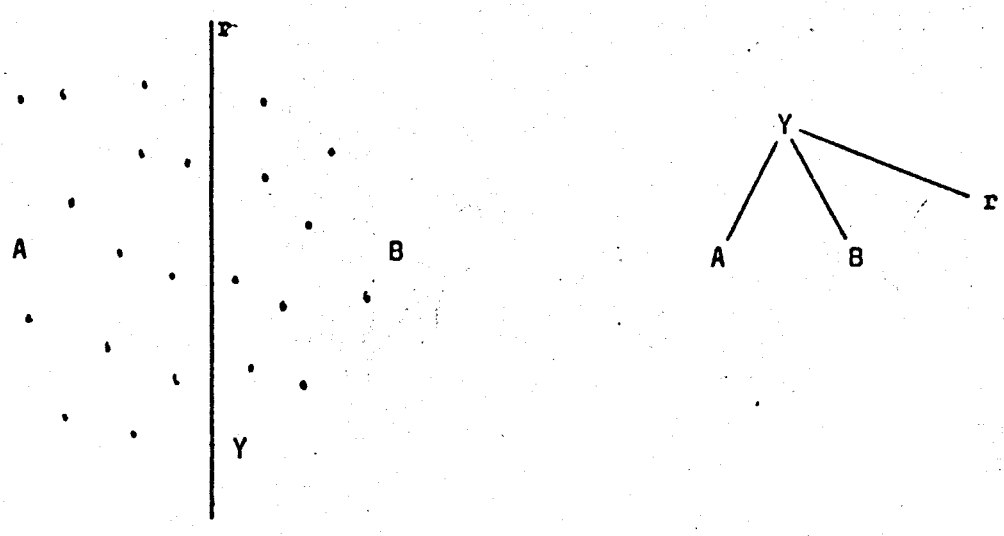


Figura 10

Teorema 9.- Dado el conjunto $X = \{X_1, X_2, \ldots, X_n\}$ de E^k, podemos construir un pseudo kd-árbol extendido con altura máxima $\lceil \log_2 n \rceil + k$ en un tiempo $O(n \log n)$.

La demostración es análoga a los epq-árboles.

La construcción se lleva a cabo por el mismo camino que los pseudo kd-árboles ordinarios, con la única diferencia de que los puntos de igual coordenada a los que están sirviendo de división deben formar parte de la estructura asociada. Fácilmente puede demostrarse que el orden de magnitud de tiempo no crece; además, puesto que no podemos avanzar más de k veces en una estructura asociada, la altura del árbol está acotada por $\lceil \log_2 n \rceil + k$.

Por el mismo razonamiento que para los epq-árboles, la extensión de lo kd-árboles a los pseudo kd-árboles extendidos no aumenta el orden de tiempo de consulta y de las actualizaciones.

Teorema 10.- Dado un número fijo ε del intervalo $(0,1)$, existe un algoritmo que ejecuta N actualizaciones sobre un pseudo kd-árbol extendido, inicialmente vacío, tal que su altura máxima es $\log_{2-\varepsilon} n + O(1)$ y el tiempo medio de cada actualización está acotado por $O(\log^2 N/\varepsilon)$.

La demostración es similar a los epq-árboles.

Con esta extensión, el orden de almacenamiento exigido continúa siendo $O(n)$, de aquí que esta estructura sea válida y eficiente para cualquier conjunto de puntos k-dimensionales.

2.- Vectores eficientes sobre estructuras dinámicas

2.1.- Vectores eficientes sobre listas lineales

2.1.1.- Vectores eficientes y orden lexicográfico

Consideremos el espacio vectorial R^n de vectores de n componentes reales; diremos que para dos elementos $X,Y \in R^n$ es $X \lneqq Y$ si $x_i \leq y_i$ $i=1,\ldots,n$ siendo $X(x_1,x_2,\ldots,x_n)$ $Y(y_1,y_2,\ldots,y_n)$.

Análogamente diremos que $X \lneq Y$ si $x_i \leq y_i$ $i=1,\ldots,n$ y $X \neq Y$, es decir, al menos para una componente la desigualdad debe ser estricta.

Sea $\mathcal{X} = \{X_1, X_2, \ldots, X_s\}$ un conjunto de elementos de R^n; un elemento $X \in \mathcal{X}$ es llamado eficiente, si se verifica que $X \lneq Y$ implica que $Y \notin \mathcal{X}$; al conjunto de vectores eficientes de $\mathcal{X}$ lo designaremos por $eff(\mathcal{X})$, es decir:

$$eff(\mathcal{X}) = \{X \in \mathcal{X} \;/\; X \lneq Y \Longrightarrow Y \notin \mathcal{X}\}$$

a $eff(\mathcal{X})$ lo llamaremos conjunto óptimo de Pareto; la relación de orden $\leq$ recibe el nombre de orden de Pareto y a los elementos de $eff(\mathcal{X})$ puntos eficientes de Pareto.

Si $X \in \mathcal{X}$ y $X \notin eff(\mathcal{X})$, X es un vector dominado

Si $X,Y \in \mathcal{X}$ y $X \lneq Y$, diremos que X es dominado por Y o que Y domina a X; si $eff(\mathcal{X})$ consta de un único elemento, lo llamaremos solución perfecta o ideal.

Veamos como organizar una lista lineal para la búsqueda de los vectores eficientes de $\mathcal{X}$; dado que las relaciones definidas $\lneqq$ y $\leq$ no son órdenes totales en R^n, no nos sirven para organizar una lista lineal; para salvar esta cuestión, definimos

seguidamente la relación de orden lexicográfico que veremos que es un orden total.

Un vector $X \in R^n$ diremos que es lexicográficamente positivo o 1-positivo si su primera componente no nula es positiva; esto suele expresarse con $X > 0$.

Un vector $X \in R^n$ es lexicográficamente negativo si $-X$ es 1-positivo.

El vector cero será considerado indistintamente como 1-positivo y 1-negativo.

La relación de orden lexicográfica se define para cada pareja de elementos $X, Y \in R^n$ por:

$$X < Y \iff Y-X \text{ es 1-positivo}$$

la relación de orden lexicográfica así definida es un orden total.

Este orden total nos permitirá situar los elementos de χ en orden lexicográfico decreciente:

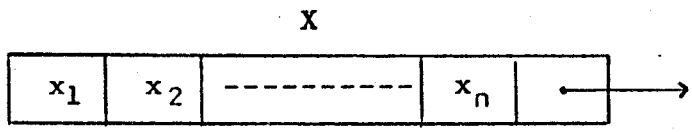
$$X_{i_1} > X_{i_2} > \text{-------} > X_{i_k} > \text{-------} > X_{i_s}$$

los vectores situados a la izquierda de X_{i_k} son lexicográficamente mayores, y los situados a su derecha, menores; así pues, todo vector dominado por X_{i_k} debe quedar a su derecha, y todo vector que domine a X_{i_k} debe quedar a su izquierda; obviamente y salvo que X_{i_k} sea la solución ideal, a la izquierda de X_{i_k} existen elementos que no dominan a X_{i_k}, o a su derecha existen elementos no dominados por él.

Veamos a continuación como hacer dinámica la lista anterior, para de esta forma, ir determinando los vectores eficientes manejando una parte reducida de χ.

2.1.2.- Dinamización de una lista multidimensional

Para hacer dinámica una lista de vectores ordenada lexicográficamente decreciente, almacenaremos cada vector $X(x_1, x_2, \ldots, x_n)$ en un registro donde serán colocadas sus n componentes, junto con la dirección en la que se encuentra el vector siguiente:

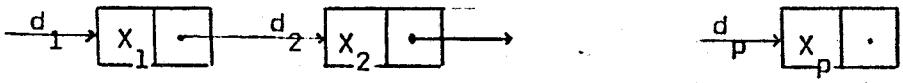


este será el tipo base que servirá para almacenar los elementos de la lista.

Una lista

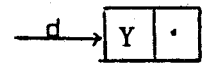
$$X_1 > X_2 > \text{----------} > X_p$$

quedará en la forma:



hemos designado por d_i la dirección donde se encuentra almacenado el elemento X_i.

El recorrido de esta lista se hace a partir de la dirección d_1 del primer elemento; este primer elemento, nos facilita la dirección del segundo, que a su vez, nos dará la dirección del tercero, y así sucesivamnte hasta llegar al último, que al no guardar dirección alguna, nos indica que el recorrido ha terminado.

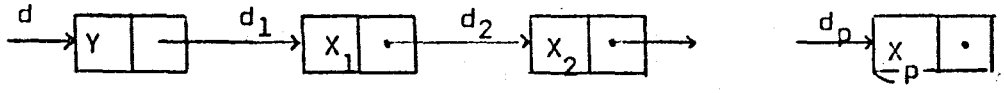
Veamos ahora como podemos insertar un nuevo elemento en la lista; sea Y éste nuevo elemento que lo guardaremos en un registro 

y deseamos insertarlo lexicográficamente en la lista anterior; para ello haremos el recorrido de la lista

comparando cada vector X_i con Y y avanzando cada vez que el vector X_i sea 1-mayor que Y; siguiendo este proceso, sea X_k el primer vector de la lista tal que $Y \not< X_k$.

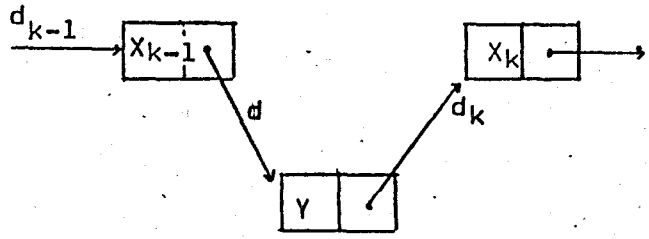
puede ocurrir:

a.- k=1 , con lo cual estamos al principio de la lista donde debemos insertar Y; para ello guardamos d_1 en Y y d será ahora el comienzo de la lista, que quedaría:

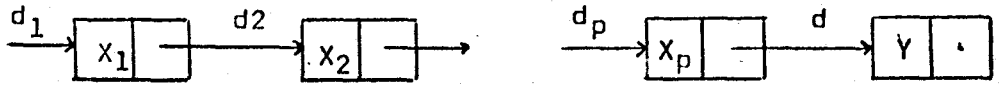


tomamos d como la dirección del comienzo de la lista indicándonos dónde se encuentra Y, y de Y colgaremos el resto de la lista.

b.- $1 < k < p$, es decir, se tiene que $Y < X_{k-1}$ pero $Y \not< X_k$, con lo que Y debe ser colocado entre ambos; esto se consigue con un simple cambio de direcciones.



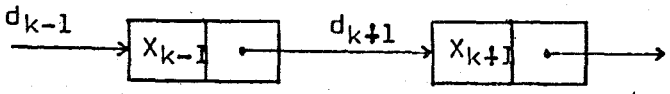
c.- Si para todo valor de i se verifica que $Y < X_i$, debemos insertar Y al final de la lista, quedando



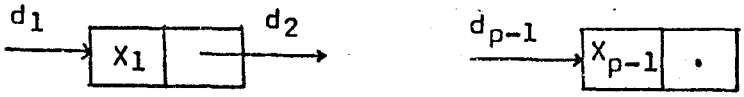
Veamos a continuación como podemos borrar un elemento X_j de la lista; lo primero que debemos hacer es recorrer la lista, como hemos indicado anteriormente, localizarlo y llevar a cabo el correspondiente cambio de direcciones; también aquí pueden ocurrir tres casos, según que el punto esté situado en los extremos de la lista o en un lugar intermedio.

a.- Borrar X_1: se reduce a considerar d_2 como la dirección de comienzo de la lista:

b.- Borrar X_k con $1 < k < p$: en este caso colocaremos en X_{k-1} la dirección de X_{k+1}:



c.- Borrar X_p: borrar el último elemento de la lista equivale a indicar el fin de la lista en X_{p-1} :



2.1.3.- Algoritmo eficiente en lista lineal

El algoritmo que seguiremos para obtener los vectores eficientes del conjunto X, consistirá en ir recorriendo dicho conjunto y mantener una lista con el conjunto de todos los vectores eficientes correspondientes al subconjunto X' de vectores recorridos hasta el momento. Para ello sea:

$$X_1 > X_2 > \text{----------} > X_k > X_{k+1} > \text{-------} > X_p$$

los vectores eficientes del subconjunto $X' \subset X$ y sea Y un elemento de $X-X'$; recorreremos la lista anterior buscando el punto donde debe ser insertado el elemento Y, y supongamos que el lugar apropiado fuese entre X_k y

X_{k+1} . Puede ocurrir:

a.- Que en el recorrido hasta X_k, hubiese un vector X_i $1 \leqslant i \leqslant k$ que eliminase al vector Y, en cuyo caso Y no puede ser eficiente de X y podríamos abandonarlo y pasar a otro elemento de $X-\{X' \cup \{Y\}\}$.

b.- Si Y no ha sido dominado, lo insertaremos entre X_k y X_{k+1}, como hemos visto anteriormente, pero habremos de recorrer la lista desde X_{k+1} hasta X_p y eliminar de ella todos los vectores dominados por Y.

Evidentemente, la nueva lista la formarán los vectores eficientes del conjunto $X' \cup \{Y\}$.

Obsérvese que Y no puede dominar a ningún vector X_i $1 \leqslant i \leqslant k$ ya que $Y \prec X$ y tampoco puede ser dominado por ningún vector X_j con $k+1 \leqslant j \leqslant p$ dado que $Y \succ X$.

2.1.4.- Experiencias computacionales

El programa para la búsqueda de vectores eficientes en lista lineal ha sido realizado en lenguaje pascal y lo hemos titulado efflista.

La constante n representará el número de componentes de los vectores a estudiar.

Vector es un tipo de array que guardará las componentes de un vector dado.

Pnodo es una variable que guardará direcciones de elementos de tipo nodo.

Nodo es un tipo registro que guardará un vector y la dirección del siguiente nodo de la lista; este tipo será el

componente básico de la lista.

A, representará el vector a insertar en cada momento.

Raiz guardará siempre la dirección que apuntará a la cabecera de la lista, mientras que cent será un registro de fin de lista que actuará a modo de centinela para avisarnos de que hemos llegado al final.

Nv, co y ti nos medirán respectivamente el número de vectores insertados, el de comparaciones vectoriales efectuadas y el tiempo empleado.

La función "igual" nos dice si los vectores v y w son iguales; si lo son vale true y si no, false. Analogamente se comportan las funciones "dom" para ver si el vector v domina al w, y la función "lmenor" que nos dice si el vector v es lexicográficamente menor que w.

La rutina "imprimir" nos da en salida los vectores eficientes de X.

La rutina "insertar" es el núcleo del programa y es la encargada de construir la lista en orden lexicogrfico decreciente, entre raiz y cent; para insertar un vector en la lista actual, buscaremos la posición donde debe ser insertado y para esta localización usamos los punteros p1 y p2, adelantando el p1 una posición sobre p2; el p3 lo usaremos durante la inserción en el intercambio de punteros.

Como vemos en el programa, p2 se inicializa en raiz y p1 una posición adelantada respecto a p2; esto obliga a que el vector que contiene la dirección de raiz sea vacío; en la dirección cent, guardaremos el vector a insertar antes de iniciar el recorrido de la lista; ello nos indicará el final de la lista si la búsqueda no se hubiese detenido antes.

La sentencia while recorre la lista mientras el vector a, sea l-menor que el vector sobre el que nos encontramos y no sea dominado por éste; así, como máximo llegaremos hasta la posición de cent. Si a no ha sido dominado, lo insetamos y hacemos el recorrido de la lista hasta el final, borrando los vectores dominados por a.

El programa principal inicializa la lista vacía y las variables que nos medirán los vectores insertados, el número de comparaciones vectoriales efectuadas y el tiempo; después, el programa va leyendo e insertando los vectores almacenados en un fichero, hasta llegar al final de éste.

El cuadro 1 muestra los datos obtenidos para vectores de 3,5,7 y 9 componentes, generados aleatoriamente. La columna izquierda indica el número de componentes, y la superior el número de vectores insertados, que varía desde 1000 hasta 10000; para cada una de las combinaciones está expresado, de arriba abajo, el número de vectores eficientes, el tiempo en milisegundos y el número de comparaciones vectoriales efectuadas. Las gráficas de las figuras 1,2 y 3, nos muestran la evolución de estas variables, en relación con el número de vectores insertados.

Programa efflista

```
Program efflista(input,output);
const n=5;
type vector=array[1..n] of integer;
     pnodo=^nodo;
     nodo=record x:vector;
                 sig:pnodo
          end;
var a:vector;
    raiz,cent:pnodo;
    i,co,nv,ti:integer;

function igual(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<>w[i]) then t:=false
  end;
  igual:=t;
end;

function dom(v,w:vector):boolean;
var t:boolean;
begin
  t:=true;i:=0;co:=co+1;
  while (t and (i<n)) do
  begin i:=i+1;
    if (v[i]<w[i]) then t:=false
  end;
  if t then if (igual(v,w)) then t:=false;
  dom:=t;
end;


function lmenor(v,w:vector):boolean;
begin
  i:=1;co:=co+1;
  while ((v[i]=w[i]) and (i<n)) do i:=i+1;
  if (v[i]<w[i]) then lmenor:=true else lmenor:=false;
end;

procedure imprimir;
var p:pnodo;
    J,k,l:integer;
begin
  p:=raiz^.sig;
  l:=24 div n;J:=0;
  while (p<>cent) do
  begin for k:=1 to l do
    begin if(p<>cent) then
      begin for i:=1 to n do write(p^.x[i]:3);
        write(' ');J:=J+1;
        p:=p^.sig
      end;
    end;
    writeln
```

```
  end;
  writeln;writeln('      numero de vectores eficientes=',j);
end;

procedure insertar(a:vector);
var p1,p2,p3:pnodo;
    t:boolean;

begin
  p2:=raiz;p1:=p2^.sig;cent^.x:=a;
  t:=true;
while (t and lmenor(a,p1^.x)) do
    if dom(p1^.x,a) then t:=false else
    begin p2:=p1;p1:=p2^.sig
    end;
  if t then
  begin
    if ((p1=cent) or not(igual(p1^.x,a))) then
    begin new(p3);p3^.x:=a;p3^.sig:=p1;p2^.sig:=p3;p2:=p3
    end;
    while (p1<>cent) do
      if dom(a,p1^.x) then
      begin
        p2^.sig:=p1^.sig;
        dispose(p1);
        p1:=p2^.sig;
      end else
      begin
        p2:=p1;
        p1:=p2^.sig
      end;
  end;
end;

begin
  new(raiz);new(cent);raiz^.sig:=cent;
  nv:=0;co:=0;ti:=clock;
  while not(eof(input)) do
  begin
    for i:=1 to n do read (a[i]);nv:=nv+1;
    if eoln(input) then readln;
    insertar(a);
  end;
  ti:=clock-ti;
  writeln;
  writeln(nv,ti,co);
  imprimir
end.
```

	1000	2000	3000	4000	5000
	37	30	37	48	40
3	1140	2230	3030	4020	4840
	12239	23605	31670	41497	49632
	184	249	292	326	371
5	7220	17090	26550	37200	48050
	78090	186532	292727	412900	533685
	417	667	850	1056	1097
7	23660	65600	114190	171100	230790
	245774	681905	1187088	1779108	2405630
	695	1201	1624	2082	2478
9	50200	160950	314800	504970	721820
	485391	1546990	3018665	4838947	6886793

	6000	7000	8000	9000	10000
	42	42	44	48	48
3	5640	6390	7150	8170	9170
	57139	64409	72249	82300	92308
	421	437	469	493	516
5	59760	71820	83160	94690	107380
	666043	802261	931459	1061620	1200474
	1245	1330	1365	1485	1503
7	285870	343800	399440	450800	507190
	2972561	3570342	4145722	4679751	5262962
	2820	3117	3471	3769	4038
9	975300	1247460	1550830	1882860	2218010
	9259359	11807341	14678805	17810153	21013703

Cuadro 1

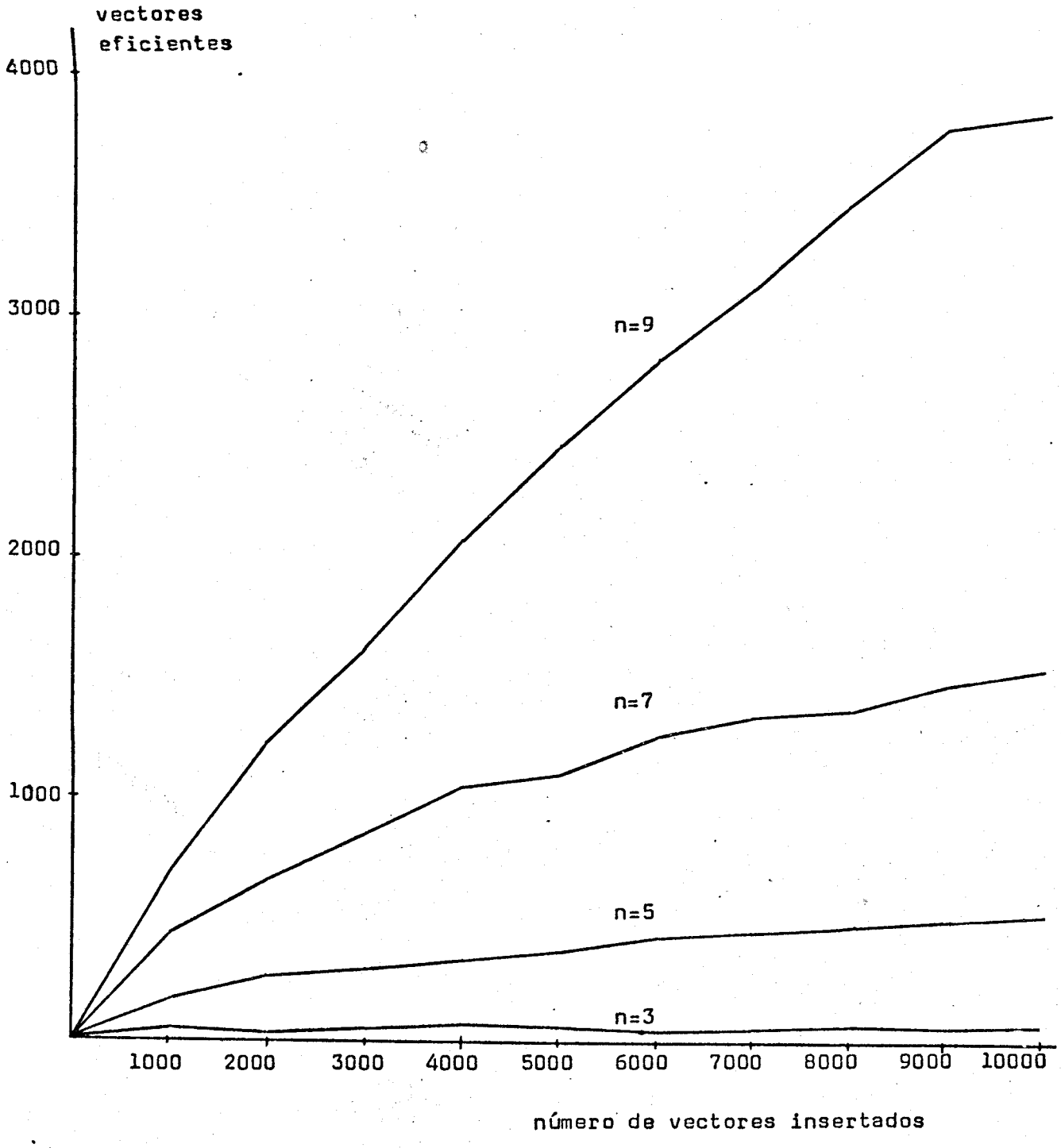

vectores
eficientes
4000
3000
2000
1000
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
número de vectores insertados

Figura 1

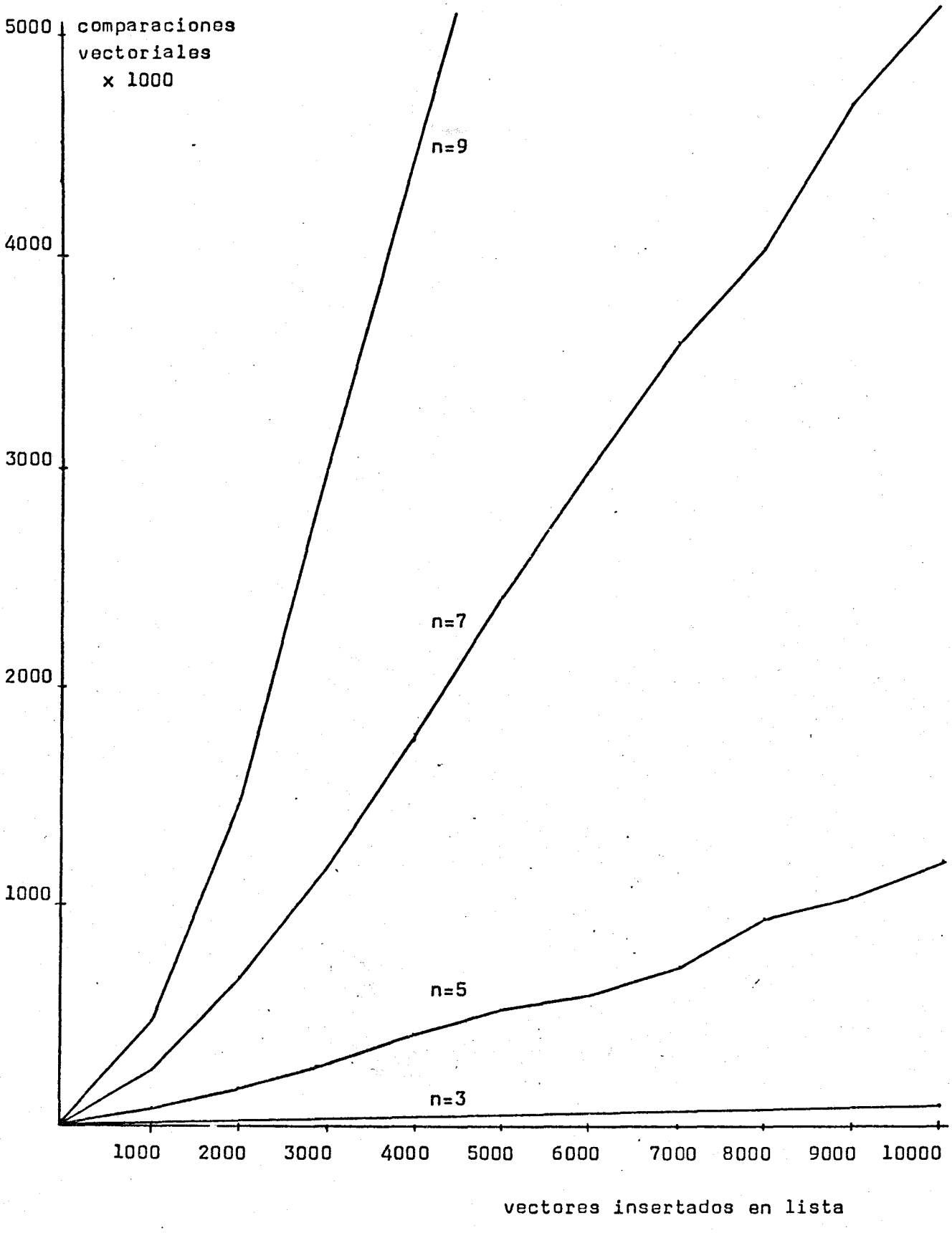

5000
comparaciones
vectoriales
x 1000
4000
3000
2000
1000
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en lista

Figura 2

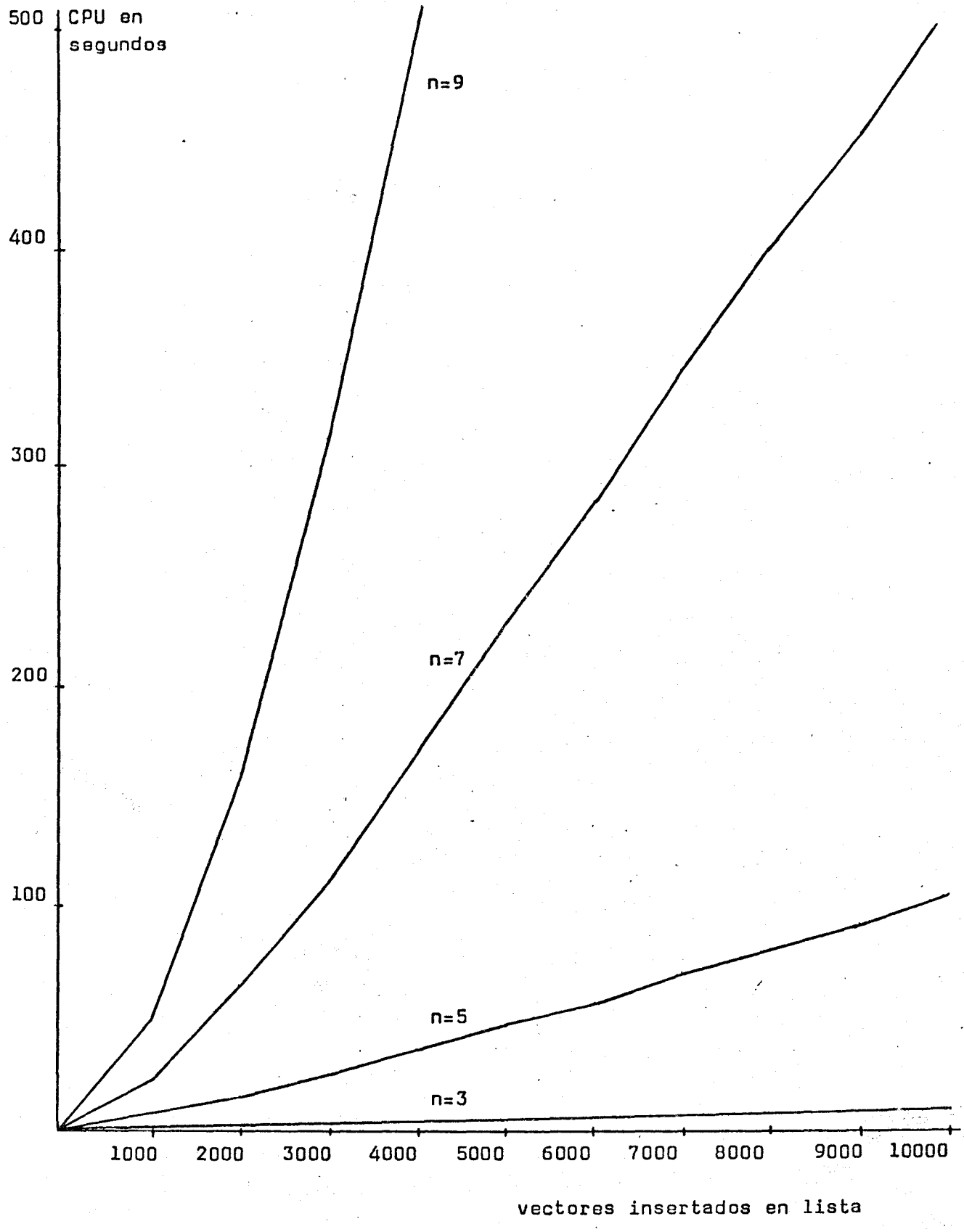

500
CPU en
segundos
400
300
200
100
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en lista

Figura 3

2.2.- Vectores eficientes sobre árboles ordenados

La relación de orden lexicográfica permite usar como soporte un árbol binario ordenado y si se quiere, equilibrado tipo AVL.

Recordemos que en esta estructura, el subárbol izquierdo de cada nodo X, lo forman nodos que son 1-menores que X y el subárbol derecho lo forman nodos que son 1-mayores; vimos cómo efectuar la búsqueda de un elemento del árbol y su actualización, bien se trate de inserción o de borrado en un tiempo de orden logarítmico del número de nodos actuales, mientras que en las listas lineales, el orden es del número de nodos actuales en lista.

Ahora bien, ésta eficiencia superior de los árboles binarios ordenados sobre las listas, desaparece en el problema que estamos resolviendo; en efecto, aquí no se trata de localizar un elemento en el árbol, o de insertar o borrar otro, sino que, lo que hemos de hacer es que al introducir un elemento en el árbol ordenado, que suponemos libre de dominancia, nos resulte otro árbol que también esté libre de dominancia, es decir, introducir un elemento Y, y ver si es dominado por alguno del árbol, y si no lo es, estudiar si domina algún nodo ya existente.

Para comprobar esto, tendremos que comparar Y con los que son 1-mayores que él, y si alguno lo domina, hemos terminado el proceso; si Y no fuese dominado por algún vector del árbol, tendremos que visitar todos los nodos que son 1-menores que Y, para ver si Y los domina o no; los que sean dominados por Y deberán ser borrados del árbol, y los que no lo sean deben

permanecer en la estructura; el resultado es un árbol libre de dominancia.

Vemos que en este algoritmo, el número de nodos visitados es del mismo orden que en las listas lineales, ya que si Y no es dominado por ningún nodo ya existente en el árbol, hemos tenido que visitar, para hacer esta comprobación, todos los nodos que son 1-mayores que Y; a su vez, para eliminar del árbol todos los dominados por Y, tendremos que visitar todos los nodos menores que Y; luego hemos recorrido todos los nodos del árbol, tal como ocurría en las listas lineales. Si Y es dominado, también el número de comparaciones es del mismo orden.

Tenemos así, que la estructura árbol 1-ordenado no aporta ningún beneficio sobre las listas en el proceso de búsqueda de los vectores eficientes; esto es debido a que el orden lexicográfico no da ninduna información sobre la dominancia, salvo el hecho de que los vectores que dominen a Y, han de ser 1-mayores que Y, y los dominados han de ser 1-menores, pero no existen otros criterios que nos eviten comparaciones innecesarias sobre parte de los vectores.

En las siguientes estructuras multidimensionales existirán criterios que nos permitan descartar ramas completas, con la seguridad de que no existe relación de dominancia entre el vector Y y los vectores de dicha rama.

Naturalmente, esto habrá de conseguirse a base de hacer la estructura arbórea más compleja que la típica del árbol binario ya estudiado.

2.3.- Vectores eficientes sobre q-árboles

2.3.1.- Construcción del q-árbol

Sean $X,Y \in R^n$ y veamos como identificar el cuadrante en el que se encuentra uno respecto del otro; para ello debemos establecer una identificación entre los cuadrantes originados por los planos coordenados; cada cuadrante está caracterizado por el hecho de que las coordenadas de los puntos que pertenecen a él tienen signo constante; además, los puntos de los hiperplanos que dividen el espacio los consideraremos pertenecientes al semiespacio de coordenada positiva, es decir, si $X(x_1,x_2,\ldots,x_n)$ es un elemento de R^n, el cuadrante al que pertenece lo identificaremos por:

$$(\delta(x_1),\delta(x_2),\ldots,\delta(x_n))$$

siendo:

$$\delta(x_i) = 1 \text{ si } x_i \geq 0$$
$$\delta(x_i) = 0 \text{ si } x_i < 0$$

como éste vector está formado por ceros y unos, podemos representarlo por un número decimal cuya codificación en binario sea exactamente dicho vector; estableceremos el orden de izquierda a derecha, de forma que $\delta(x_1)$ son las unidades de primer orden en base dos, $\delta(x_2)$ las de segundo orden, y así sucesivamente hasta $\delta(x_n)$; de manera que dicho cuadrante queda unívocamente definido por:

$$k = \sum_{i}^{n} \delta(x_i)\ 2^{i-1}$$

Dados dos vectores $X,Y \in R^n$, el cuadrante que ocupa Y respecto de X viene dado por:

$$k_{xy} = \sum_{i}^{n} \delta(y_i - x_i) \cdot 2^{i-1}$$

evidentemente el cuadrante que ocupa X respecto de Y es:

$$k_{yx} = \sum_{i}^{n} \delta(x_i - y_i) \cdot 2^{i-1} = 2^n - 1 - k_{yx}$$

es decir, los cuadrantes opuestos son complementarios respecto del número 2^n-1.

Por ejemplo, en R^n el punto X(3,5,4) origina 8 cuadrantes, y los puntos Y indicados son i-sucesores de X, como vemos en la figura 4.

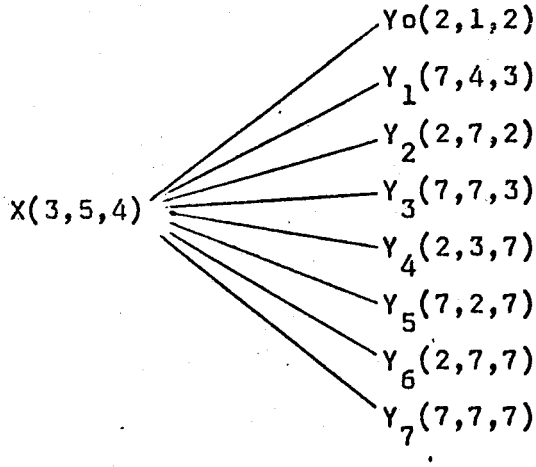


Figura 4

Definición: Sean X,Y dos nodos de un q-árbol; Y es llamado k-sucesor de X si:

$$k = \sum_{i}^{n} \delta(y_i - x_i)\ 2^{i-1}$$

Además, Y es llamado un k-hijo de X si:

a.- Y es un k-sucesor de X

b.- X es un predecesor directo de Y

Con esta notación y enumeración de cuadrantes, el algoritmo de inserción de un elemento Y en un q-árbol es el siguiente:

a.- Si el q-árbol está vacío, colocaremos Y como raiz y terminamos.

b.- Sea X la raiz del q-árbol.

c.- Hallar $k = \sum_{1}^{n} \delta(y_i - x_i)\ 2^{i-1}$; Y es un k-sucesor de X.

d.- Si X no tiene ningún k-hijo, colocar ahí el elemento Y y hemos terminado.

e.- Si X tiene un k-hijo, llamar nuevamente X al k-hijo de X y volver a c.

Este algoritmo construye el q-árbol para un conjunto dado de vectores a base de inserciones, sin que permita borrar ningún elemento ni equilibrar el árbol; veamos cómo podemos usar ésta estructura en la búsqueda de vectores eficientes.

2.3.2.- Identificación de vectores eficientes sobre q-árboles

Diremos que un q-árbol está libre de dominancias si no existe un par de nodos que domine uno al otro; lo que intentaremos conseguir es insertar nodos en el q-árbol manteniéndolo libre de dominancia; para ello, si tenemos un q-árbol libre de dominancia y deseamos insertar un nuevo vector Y, debemos plantearnos dos cuestiones:

a.- Si existe algún vector en el q-árbol que domine a Y, desechar el vector Y; si Y no es dominado por ningún vector del q-árbol, lo insertaremos tal como dijimos en el apartado anterior.

Hecha la inserción de Y, debemos eliminar del q-árbol todos los vectores dominados por Y.

Por la transitividad de la dominancia, un vector que se rechace en el punto a, no puede dominar a ninguno de los que ya están en el q-árbol, por lo que el punto b, no es necesario comprobarlo en este caso.

En cualquier caso, es evidente que el q-árbol resultante está libre de dominancias.

Ahora establecemos los criterios que nos permitirán ejecutar los puntos a y b, sin tener que visitar todos los nodos existentes; con el fin de enumerar los sucesores de un nodo X, designemos por:

$$C_{xy} = (\delta(y_1-x_1),\delta(y_2-x_2),\ldots,\delta(y_n-x_n))$$

$$k = \sum \delta(y_i-x_i)\cdot 2^{i-1} \quad Y \text{ es k-sucesor de } X$$

$$C_1(k) = \{ i: \delta(y_i-x_i) = 1\}$$
$$C_0(k) = \{ i: \delta(y_i-x_i) = 0\}$$

evidentemente se tiene:

$$C_0(k) \cup C_1(k) = \{1,2,\ldots,n\}$$

Según esto, Y es un k-sucesor de X si y solo si se verifica que:

a.- $i \in C_1(k) \iff y_i > x_i$

b.- $i \in C_0(k) \iff y_i < x_i$

Los siguientes resultados ayudarán a que la inserción en el q-árbol manteniéndolo libre de dominancias, evite el recorrido de amplias partes del q-árbol.

Teorema 1.- Sean dos vectores X,Y distintos,y supongamos que Y es un k-sucesor de X; se verifica:

a.- Y domina a X si y solo si $k=2^n-1$

b.- X domina a Y si y solo si $y_i=x_i \ \forall i \ C_1(k)$

Demostracion: Este teorema es una consecuencia directa de la definición de k-sucesores; si $k=2^n-1$ el conjunto $C_1(k)=\{1,2,\ldots,n\}$ ya que $C_{xy}=\{1,1,\ldots,1\}$ por lo que $y_i \geq x_i$ para cualquier i, y por lo tanto Y domina a X; así pues, todo $k=2^n-1$ sucesor de X domina al vector X.

El punto b, también es evidente, ya que si $i \in C_0(k)$ es $y_i < x_i$ y si $i \in C_1(k)$ es $y_i = x_i$, luego como X e Y son distintos, se verifica que X domina a Y.

c.q.d.

Luego en un q-árbol libre de dominancias cualquier nodo tiene a lo sumo 2^n-2 descendientes directos, ya que el 0 y el 2^n-1 no pueden existir; de aquí que para n=2 se tenga un árbol binario.

El siguiente teorema nos ayudará a identificar las ramas del q-árbol que no tengamos que recorrer, manteniendo la garantía de la libre dominancia.

Teorema 2.- Sean X,Y,Z tres vectores de R^n, y supongamos que Y es un k_1-sucesor de X y Z un k_2-sucesor de X; se verifica:

a.- Si Z domina a Y entonces $C_1(k_1) \subset C_1(k_2)$ o lo que es

equivalente que $C_0(k_1) \supset C_0(k_2)$.

b.- Si Y domina a Z entonces $C_0(k_1) \subset C_0(k_2)$ o lo que es equivalente $C_1(k_1) \supset C_1(k_2)$.

Demostración: La demostración también es inmediata ya que si existe un $i \in C_1(k_1)$ tal que $i \notin C_1(k_2)$, sería $y_i \geq x_i > z_i$ e Y no sería dominado por Z.

Además, si suponemos que existe $i \in C_0(k_1)$ tal que $i \notin C_0(k_2)$ sería $y_i < x_i \leq z_i$ por lo que Y no dominaría a Z.

c.q.d.

Por parte a, del teorema, se puede restringir la búsqueda de vectores dominantes de Y sobre aquellos k_2-sucesores de X tales que la reprsentación binaria de k_2 tiene un uno donde k_1 lo tenga.

Por la parte b, los vectores dominados por Y están entre aquellos k_2-sucesores, tales que la representación binaria de k_2 tiene un cero donde la de k_1 lo tenga.

Una consecuencia inmediata de este teorema es:

a.- Si Z domina a Y ha de ser $k_1 \leq k_2$

b.- Si Y domina a Z ha de ser $k_2 \leq k_1$

Supongamos que Y es un k-sucesor de X y que C_{xy} tiene p unos y q ceros; estudiemos las ramas que tendríamos que recorrer para ver si Y es dominado, o si Y domina algún vector descendiente de X.

Los vectores que dominen a Y deben tener igual o mayor número de unos que C_{xy}, luego el total de ramas a examinar viene dado por:

$$\binom{q}{0} + \binom{q}{1} + \text{----------} \binom{q}{q} = 2^q$$

En cambio, los posibles vectores dominados por Y deben tener igual o mayor número de ceros que la representación binaria de k, por lo que en total son:

$$\binom{p}{0} + \binom{p}{1} + \text{----------} \binom{p}{p} = 2^p$$

como tanto la rama correspondiente a

$$\binom{p}{p} \text{ (todos unos) como la } \binom{q}{q} \text{ (todos ceros)}$$

no existen en el q-árbol libre de dominancias, tendremos que revisar:

$2^q - 1$ ramas para ver si existe alguno que domine a Y.

$2^p - 1$ para ver si existe alguno dominado por Y.

Lógicamente, en el peor de los casos se examinarán 2^p+2^q-2 ramas, mientras que, sin tener en cuenta el teorema anterior, habría que examinar $2^{p+q}-2 = 2^n-2$ ramas, y evidentemente:

$$2^p+2^q-2 \leqslant 2^{p+q}-2$$

y en algunos casos, mucho menor, cuando $p = \lfloor n/2 \rfloor$ y $q = \lceil n/2 \rceil$.

Realmente:

$$2^p+2^q-2 \leqslant 2^1+2^{n-1}-2 = 2^{n-1}$$

lo cual quiere decir, que en el peor de los casos tendremos que examinar la mitad de los nodos del q-árbol que parten de X.

2.3.3.- Algoritmo eficiente

Veamos el algoritmo de inserción de un elemento Y en un q-árbol preservando la relación de dominancia libre, es decir, supondremos un q-árbol en el que no existe una pareja de elementos dominado uno por el otro, y además, lo supondremos no vacío, ya que si estuviese vacío colocaríamos a Y como raiz del q-árbol.

a.- Sea X la raiz del q-árbol

b.- Calcular $k = \sum_{I}^{n} \delta (y_i - x_i) \cdot 2^{i-1}$

Si $k = 2^n - 1$ entonces Y domina a X; borrar X

Si X domina a Y, es decir, si $x_i = y_i \ \forall i \in C_1(k)$ Fin.

c.- Para todo vector Z tal que Z es un k'-sucesor directo de X con $k < k'$ y $C_1(k) \subset C_1(k')$ ejecutar el test de dominancia Test1(Y,Z).

d.- Para todo vector Z tal que Z es un k'-sucesor directo de X con $k' < k$ y $C_0(k) \subset C_0(k')$ ejecutar el test de dominancia Test2(Y,Z).

e.- Si X ya tiene un k-hijo, volver a b, llamando nuevamente X al k-hijo de X; en otro caso, Y ocupará la posición de k-hijo de X y habríamos terminado.

Los procedimientos Test1 y Test2 son definidos recursivamente aprovechando la propiedad de estas estructuras de que cualquier subárbol de un q-árbol es también un q-árbol. Vienen definido por:

Test1(Y,Z)

a.- Determinar $k = \sum_{I}^{n} \delta(y_i - z_i)\ 2^{i-1}$ de forma que Y sería un k-sucesor de Z. Si $x_i = y_i\ \forall\ i \in C_1(k)$ tendríamos que Y está dominado por Z, y terminaríamos.

b.- Para todo vector T tal que sea un k'-hijo de Z con $k' < k$ y $C_0(k) \subset C_0(k')$ ejecutar Test1(Y,T).

Test2(Y,Z)

a.- Determinar $k = \sum \delta(y_i - z_i) \cdot 2^{i-1}$ de forma que Y sería un k-sucesor de Z. Si $k = 2^n - 1$ borraríamos Z ya que es dominado por Y.

b.- Para todo vector T tal que sea un k'-hijo de Z con $k' < k$ y $C_0(k) \subset C_0(k')$ ejecutar Test2(Y,T).

En el siguiente apartado estudiaremos el comportamiento computacional del algoritmo y algunas dificultades que presenta su implementación.

2.3.4.- Experiencias computacionales

El programa, hecho en lenguaje pascal, para la búsqueda de vectores eficientes sobre q-árboles lo hemos titulado qárbol.

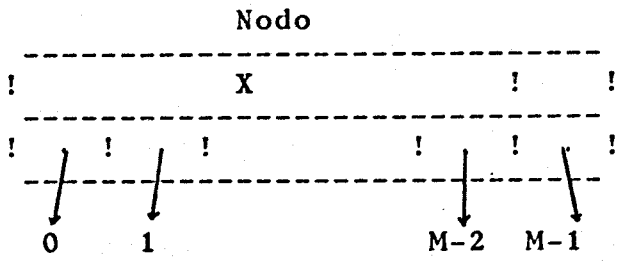
La constante n representará el número de componentes de los vectores a tratar, y la constante $m = 2^n - 1$ indicará las ramas que cuelgan de cada nodo, desde 0 hasta m, aunque como hemos dicho, ambos extremos estarán siempre vacíos.

Vector es el tipo que almacenará las componentes de un vector dado.

Pnodo es un tipo puntero que guardará direcciones de tipo

nodo.

Nodo es el tipo base de la estructura que guardará, un vector X, el número d de descendientes y 2^n punteros hacia elementos situados en los cuadrantes originados por X:



A representará el vector a insertar en cada momento y el el vector s hará el papel de C estudiado anteriormente.

Raiz será en cada momento la dirección que apuntará a la raiz del q-árbol.

Las variables nv, co y ti asi como las funciones "igual" y "dom" tendrán el mismo significado que en efflista.

La función "delta" está definida por:

$$delta(v,w) = (\delta(v_1 - w_1), \delta(v_2 - w_2), \ldots, \delta(v_n - w_n))$$

el resultado es un vector que será almacenado usualmente en s.

Las rutinas imprimir1, imprimir2, e imprimir3 nos escriben los vectores eficientes en distintas configuraciones; las dos primeras mantienen en impresión la forma de un q-árbol y la variable e, que es usada como parámetro indica el nivel del q-árbol en cada momento; estas dos formas de impresión son usadas como ilustración cuando tenemos pocos elementos. Con imprimir3, se escriben los vectores en varias columnas dependiendo del número de componentes.

La función test1 es cierta si el vector a no es dominado por algún vector bajo el puntero p, y falso en otro caso; el parámetro k indica la rama que está siendo sondeada.

E indica la rama a partir de la cual hay que aplicar test1; cuando se llama test1 desde el programa principal, hay que probar desde la rama k+1 hasta la m-1, pero cuando se llama a sí misma recursivamente hay que incluir la rama k, de ahí que se use la variable universal e para que se distinga una llamada de otra; usamos la función del sistema uand, que ejecuta la intersección binaria de dos números enteros, así cuando k=uand(j,k) significa que todo uno que figure en la descomposición binaria de k, aparece también en la descomposición binaria de j, y sabemos que esto es una condición necesaria para que un vector de la rama j domine al vector a.

La rutina insertar2 construye un q-árbol sabiendo de antemano que cada vector que entra, ni domina ni es dominado por ninguno de los que ya existen en dicho q-árbol; se usa en las reconstrucciones parciales de ramas por desaparición de algún nodo interior.

La rutina borrar, elimina del q-árbol con raiz p, todos los vectores dominados por a; el número de vectores eliminados lo indicará el parámetro variable q; dentro de borrar está la rutina recorrer, que es la encargada de visitar todos los nodos de la rama a reconstruir y reinsertarlos en el subárbol que construye borrar.

La rutina test2, localiza en el árbol de raiz p, los vectores dominados por el vector a; si se encuentra un vector dominado por a, tendrá que llamar a borrar y reconstruir el subárbol correspondiente.

La rutina insertar es el núcleo del programa; su objetivo es insertar el vector a, en el árbol de raiz p; esta rutina determina la rama que corresponde seguir al vector a; si la rama

está vacía, inserta a, y si no lo está, ejecuta test1 en las ramas superiores y test2 en las inferiores; luego continúa avanzando, si procede, a un nuevo nodo.

El programa principal inicializa el q-árbol vacío y las variables nv,co y ti; luego, el programa va leyendo e insertando los vectores almacenados en un fichero hasta llegar al final de éste.

El cuadro 2 muestra, el número de vectores eficientes, el tiempo y el número de comparaciones vectoriales, correspondientes a las mismas series de vectores generados para listas lineales. Las gráficas de las figuras 5 y 6 nos muestran la evolución del tiempo y de las comparaciones.

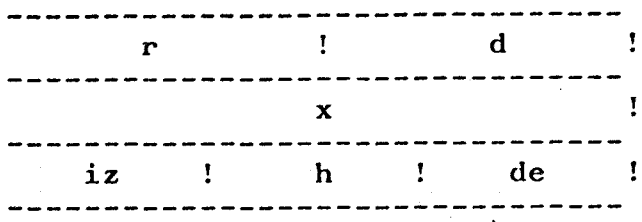
Observando estos datos, aparecen ciertas irregularidades como son el hecho de que no se tengan datos a partir de 5000 vectores de 9 componentes y que el tiempo empleado sea superior a las listas lineales; lo único que se ha comportado como era de esperar, es el número de comparaciones vectoriales, que ha disminuido notablemente.

La razón de todas las irregularidades es única, la cantidad de nodos vacíos que aparecen en una estructura q-árbol; pensemos que cada nodo hoja tiene 2 nodos vacíos que, por una parte, ocupan memoria dinámica y por otra, el hecho de que estando o no vacíos, tenemos que comprobar su estado.

La primera razón nos ha llevado a agotar la memoria dinámica asignada al proceso a partir de 5000 vectores de 9 componentes; la segunda es la causante de que el tiempo haya aumentado.

Esto puede ser salvado reformando el tipo de nodo base de la estructura q-árbol, de forma que cada nodo no tenga necesariamente 2 nodos hijos; lo que haremos es organizar todos

los hijos de un nodo en un árbol ternario, y el tipo base será:



R indica la rama paterna, de 0 a m-1 d el número de descndientes, x el vector, iz es un apuntador a hermanos izquierdo y de a derechos mientras que h apunta a los descendientes del nodo x.

De esta forma, cada nodo tiene tres direcciones en lugar de 2 ; esto obliga a una reorganización de los datos y el programa lo hemos llamado superq; los resultados, en este caso, están expresados en el cuadro 3, y en las gráficas 7 y 8.

Podemos observar que hemos mejorado los resultados de la lista lineal tanto en tiempo como en número de comparaciones; no obstante, la complejidad de la estructura, hace que los resultados no sean mejores, así que pasamos directamente a la estructura con la que hemos obtenido mejores resultados.

Programa a-arbol

```
Program aarbol(input,output);
const n=5;
      m=31;
type vector=array[1..n] of integer;
     pnodo=^nodo;
     nodo=record x:vector;
          d:integer;
          sig:array[0..m] of pnodo
     end;
var a,s:vector;
    raiz:pnodo;
    i,a,e,nv,co,ti:integer;

function igual(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<>w[i]) then t:=false
  end;
  igual:=t;
end;

function delta(v,w:vector):vector;
var b:vector;
begin co:=co+1;
  for j:=1 to n do if (v[i]>w[i]) then b[i]:=1 else b[i]:=0;
  delta:=b
end;

function dom(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<w[i]) then t:=false
  end;
  if t then if igual(v,w) then t:=false;
  dom:=t
end;

procedure imprimir(p:pnodo;e:integer);
var l,j:integer;
begin
  if (p<>nil) then
  begin l:=(m+1) div 2;
    for j:=m-1 downto l do  imprimir(p^.sig[j],e+1);
    for i:=1 to e do write('                ');
    for j:=1 to n do write(p^.x[j]:3);writeln(p^.d:4);
    for j:=l-1 downto 1 do imprimir(p^.sig[j],e+1);
    writeln;
  end else
  begin for i:=1 to e do write('                ');
     writeln('-->nil')
  end;
end;
```

```
procedure imprimir2(p:pnodo;e:integer);
var l,j:integer;
begin
  if (p<>nil) then
  begin l:=(m+1) div 2;
    for j:=m-1 downto l do  imprimir2(p^.sig[j],e+1);
    for i:=1 to e do write('  ');
    for i:=1 to n do write(p^.x[i]:3);writeln(p^.d:4);
    for j:=l-1 downto 1 do imprimir2(p^.sig[j],e+1)
  end;
end;

procedure imprimir3(p:pnodo);
var l,j:integer;
begin
  if (p<>nil) then
  begin l:=(m+1) div 2;
    for j:=m-1 downto l do  imprimir3(p^.sig[j]);
    for i:=1 to n do write(p^.x[i]:3);write(p^.d:4);
    e:=e+1;if (e=(24 div (n+1))) then
    begin e:=0;writeln
    end;
    for j:=l-1 downto 1 do imprimir3(p^.sig[j])
  end;
end;

function test1(a:vector;p:pnodo;k:integer):boolean;
var j,k1:integer;
    t:boolean;
begin j:=e;t:=true;
  while((j<m-1) and t) do
  begin j:=j+1;with p^ do
    begin if (sig[j]<>nil) then if (k=uand(j,k)) then
      begin k1:=0;s:=delta(a,sig[j]^.x);
        for i:=n downto 1 do k1:=s[i]+2*k1;e:=k1-1;
        if (k1=0) then t:=false else t:=test1(a,sig[j],k1)
      end;
    end;
  end;
  test1:=t
end;

procedure insertar2(a:vector;var p:pnodo);
var k:integer;
begin if (p=nil) then
  begin new(p);p^.x:=a;p^.d:=1;
    for i:=0 to m do p^.sig[i]:=nil;
  end else
  begin s:=delta(a,p^.x);k:=0;
    for i:=n downto 1 do k:=s[i]+2*k;
    insertar2(a,p^.sig[k]);
    p^.d:=p^.d+1
  end;
end;

procedure borrar (a:vector;var p:pnodo;var e:integer);
var raiz2:pnodo;

procedure recorrer(p:pnodo);
```

```
var j:integer;
begin if (p<>nil) then
  begin for j:=m-1 downto 1 do recorrer(p^.sig[j]);
    if dom(a,p^,x) then e:=e-1 else insertar2(p^,x,raiz2);
    dispose(p);
  end;
end;

begin
  raiz2:=nil;
  recorrer(p);
  p:=raiz2;
end;

procedure test2(a:vector;p:pnodo;k:integer;var e:integer);
var j,l,k1,k2:integer;
begin k2:=e-1;
  for j:=1 to k2 do
  begin with p^ do if (sig[j]<>nil) then
    if(j=uand(k,j)) then if dom(a,sig[j]^,x) then
    borrar(a,sig[j],e) else
    begin k1:=0;s:=delta(a,sig[j]^,x);
      for i:=m downto 1 do k1:=s[i]+2*k1;e:=k1+1;
      l:=e;e:=0;test2(a,sig[j],k1,e);
      sig[j]^.d:=sig[j]^.d+e;e:=e+1
    end;
  end;
end;

procedure insertar (a:vector;var p:pnodo;var e:integer);
var k,l:integer;
begin
  if (p=nil) then
  begin new(p);p^.x:=a;p^.d:=1;e:=1;
    for i:=0 to m do p^.sig[i]:=nil
  end else
  begin if dom(a,p^,x) then
    begin p^.x:=a;borrar(a,p,e)
    end else
    begin k:=0;s:=delta(a,p^,x);for i:=m downto 1 do k:=s[i]+2*k;
      if (k>0) then
      begin e:=k;if test1(a,p,k) then
        begin e:=k;test2(a,p,k,e);l:=e;e:=0;
          insertar(a,p^.sig[k],e);
          e:=e+l;p^.d:=p^.d+e
        end;
      end;
    end;
  end;
end;

begin
  raiz:=nil;nv:=0;ti:=clock;co:=0;
  while not(eof(input)) do
  begin a:=0;
    for i:=1 to n  do read(a[i]);nv:=nv+1;
    if eoln(input) then readln;
    insertar(a,raiz,e);
  end;
  ti:=clock-ti;
end.
```

	1000	2000	3000	4000	5000
	37	30	37	48	40
3	1000	1990	2850	3770	4640
	3839	7690	11230	14978	18540
	184	249	292	326	371
5	6910	13850	20040	26420	32580
	15354	30156	43665	57266	70237
	417	667	850	1056	1097
7	39290	91430	147140	204070	266550
	34602	79624	127398	176664	228155
	695	1201	1624	2082	
9	184710	527220	950060	1431130	
	54475	145523	250914	366062	

	6000	7000	8000	9000	10000
	42	42	44	48	48
3	5850	7300	8660	10000	11230
	24048	29435	34751	40028	45276
	421	437	469	493	516
5	39040	45410	50930	56590	63080
	83795	97172	108860	120749	133924
	1245	1330	1365	1485	1503
7	318440	375610	443860	499770	564440
	270176	317166	368186	411495	461317
9					

Cuadro 2

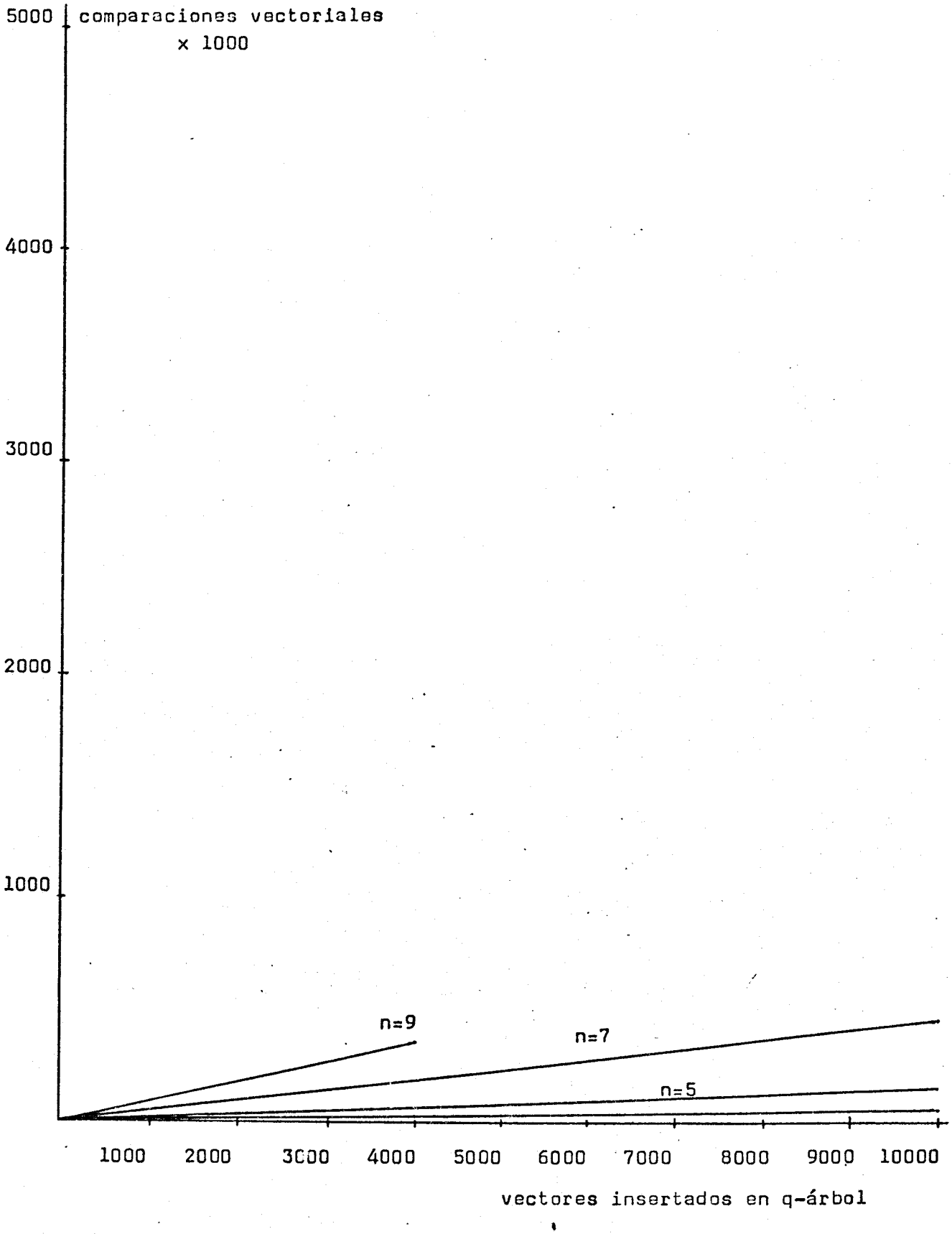

5000
comparaciones vectoriales
x 1000
4000
3000
2000
1000
n=9
n=7
n=5
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en q-árbol

Figura 5

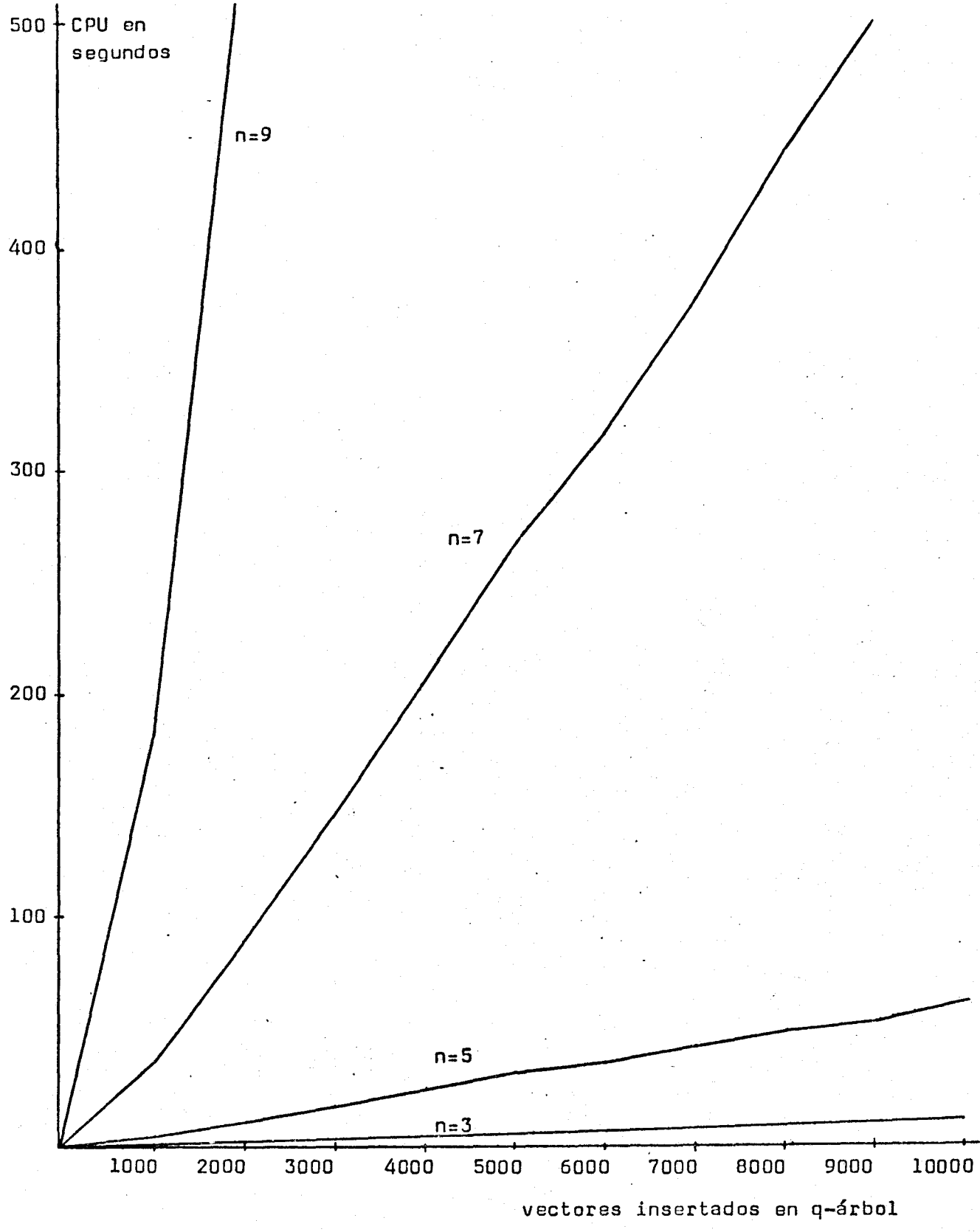

500
CPU en
segundos
400
300
200
100
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en q-árbol

Figura 6

Programa supero

```
Program supera(input,output);
const n=5;
      m=31;
type vector=array[1..n] of integer;
     pnodo=^nodo;
     nodo=record r,d:integer;
           x:vector;
           iz,h,de:pnodo
     end;
var a,s:vector;
    raiz:pnodo;
    i,k,e,nv,co,ti:integer;

function igual(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<>w[i]) then t:=false
  end;
  igual:=t;
end;

function delta(v,w:vector):vector;
var b:vector;
begin co:=co+1;
  for i:=1 to n do if (v[i]>w[i]) then b[i]:=1 else b[i]:=0;
  delta:=b
end;

function dom(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<w[i]) then t:=false
  end;
  if t then if igual(v,w) then t:=false;
  dom:=t
end;

procedure imprimir(p:pnodo;e:integer);
var i:integer;
begin
  if (p<>nil) then
  begin
    imprimir(p^.de,e+1);
    imprimir(p^.h,e+2);
    for i:=1 to e do write('    ');write(p^.r:3,' ');
    for i:=1 to n do write(p^.x[i]:3);writeln(p^.d:4);
    imprimir(p^.iz,e+1)
  end;
end;

function test1(a:vector;p:pnodo;k:integer):boolean;
var t:boolean;
```

```
begin t:=true;
  if (p<>nil) then
  begin t:=test1(a,p^.de,k);if (t and (k<p^.r)) then
    t:=test1(a,p^.iz,k);
    if ((k=uand(k,p^.r)) and (t)) then
    if dom(p^.x,a) then t:=false else
    begin k:=0;s:=delta(a,p^.x);
      for i:=n downto 1 do k:=s[i]+2*k;
      t:=test1(a,p^.h,k);
    end;
  end;
  test1:=t;
end;

procedure insertar2(a:vector;var p:pnodo;k:integer);
begin
  if (p=nil) then
  begin new(p);p^.x:=a;p^.iz:=nil;p^.de:=nil;
    p^.h:=nil;p^.d:=1;p^.r:=k
  end else
  begin if k<p^.r then insertar2(a,p^.iz,k) else
    if k>p^.r then insertar2(a,p^.de,k) else
    begin k:=0;s:=delta(a,p^.x);
      for i:=n downto 1 do k:=s[i]+2*k;
      insertar2(a,p^.h,k);p^.d:=p^.d+1;
    end;
  end;
end;

procedure borrar (a:vector;var p:pnodo;var e:integer);
var raiz2,p1,p2:pnodo;
    l:integer;

procedure bus(var pd:pnodo);
begin
  if pd^.iz<>nil then bus(pd^.iz) else
  begin p:=pd;pd:=pd^.de
  end;
end;

procedure recorrer(p:pnodo);
var k:integer;
begin if (p<>nil) then
  begin recorrer(p^.de);
    recorrer(p^.h);
    recorrer(p^.iz);k:=0;
    if dom(a,p^.x) then e:=e-1 else insertar2(p^.x,raiz2,k);
    dispose(p);
  end;
end;

begin raiz2:=nil;
  p1:=p^.iz;p2:=p^.de;p^.iz:=nil;p^.de:=nil;l:=p^.r;
  recorrer(p);
  p:=raiz2;if p=nil then
  begin if p2=nil then p:=p1 else
    if p1=nil then p:=p2 else
    begin bus(p2);p^.iz:=p1;p^.de:=p2
    end;
  end else
```

```
  begin p^.r:=1;p^.iz:=p1;p^.de:=p2
  end;
end;

procedure test2(a:vector;var p:pnodo;k:integer;var q:integer);
var l:integer;
begin
  if (p<>nil) then
  begin test2(a,p^.iz,k,q);if (k>p^.r) then test2(a,p^.de,k,q);
    if (p^.r=uand(k,p^.r)) then if dom(a,p^.x) then
    borrar(a,p,q) else
    begin
      k:=0;s:=delta(a,p^.x);
      for i:=n downto 1 do k:=s[i]+2*k;
      l:=q;q:=0;test2(a,p^.h,k,q);p^.d:=p^.d+q;q:=q+1
    end;
  end;
end;

procedure insertar (a:vector;var p:pnodo;k:integer;var q:integer);
var l,k1:integer;
    t:boolean;
begin t:=true;
  while((p<>nil) and t and (k<>p^.r))do
    if dom(a,p^.x) then borrar(a,p,q) else t:=false;
  if (p=nil) then
  begin new(p);p^.x:=a;p^.iz:=nil;p^.de:=nil;
    p^.h:=nil;p^.d:=1;p^.r:=k;q:=q+1
  end else
  begin k1:=0;s:=delta(a,p^.x);for i:=n downto 1 do k1:=s[i]+2*k1;
    if (k1>0) then if k<p^.r then
    begin if (test1(a,p^.de,k) and test1(a,p^.h,k1)) then
      insertar(a,p^.iz,k,q)
    end else if k>p^.r then
    begin test2(a,p^.iz,k,q);l:=q;q:=0;
      test2(a,p^.h,k1,q);p^.d:=p^.d+q;q:=q+1;insertar(a,p^.de,k,q)
    end else
    begin if test1(a,p^.de,k) then
      begin test2(a,p^.iz,k,q);k:=k1;
        if dom(a,p^.x) then
        begin p^.x:=a;borrar(a,p,q);
        end else if k>0 then
        begin l:=q;q:=0;insertar(a,p^.h,k,q);p^.d:=p^.d+q;q:=q+1
        end;
      end;
    end;
  end;
end;

begin
  ti:=clock;nv:=0;co:=0;
  raiz:=nil;
  while not(eof(input)) do
  begin k:=0;q:=0;
    for i:=1 to n  do read(a[i]);nv:=nv+1;
    if eoln(input) then readln;
    insertar(a,raiz,k,q);
  end;
  ti:=clock-ti;
  writeln(nv,ti,co);
end.
```

	1000	2000	3000	4000	5000
.	37	30	37	48	40
3	1530	3530	5900	8350	10570
	6333	14467	24089	34319	44052
	184	249	292	326	371
5	9070	18330	27110	36150	44660
	26751	53304	77902	102290	125227
	417	667	850	1056	1097
7	24210	59850	98700	141680	185730
	57856	138578	224624	318903	415691
	695	1201	1624	2082	2478
9	44620	119340	208890	313700	423090
	87101	229511	393540	582443	780729

	6000	7000	8000	9000	10000
	42	42	44	48	48
3	13080	15410	17430	19170	20720
	54118	63684	72557	79002	84920
	421	437	469	493	516
5	53630	62050	69940	78220	864600
	150006	173996	196438	220630	245846
	1245	1330	1365	1485	1503
7	225460	268400	311750	350640	390940
	502212	595397	694550	787408	883399
	2820	3117	3471	3769	4038
9	543600	666810	802360	936580	1073150
	998932	1222221	1468548	1707307	1950654

Cuadro 3

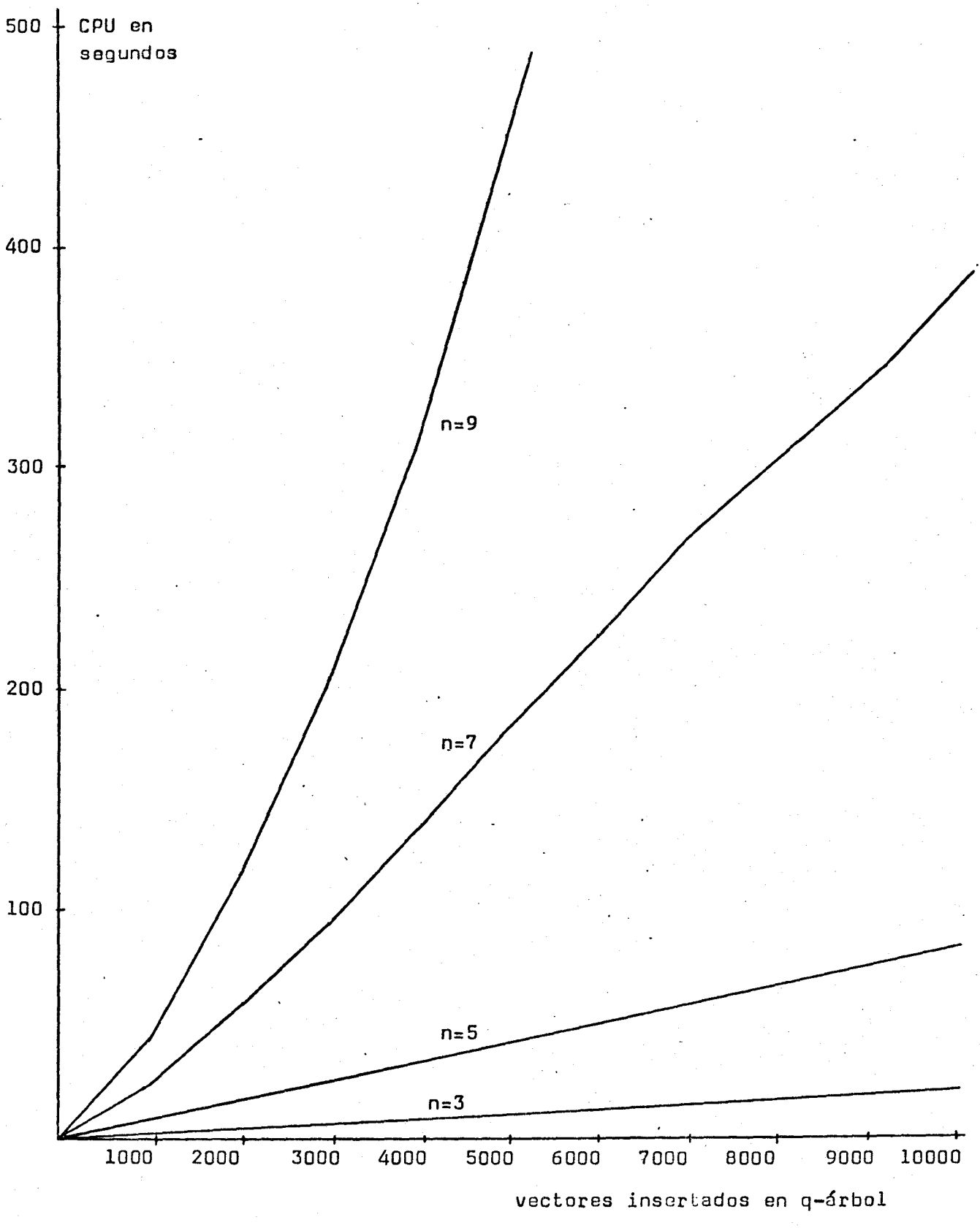

500
CPU en
segundos
400
300
200
100
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en q-árbol

Figura 7

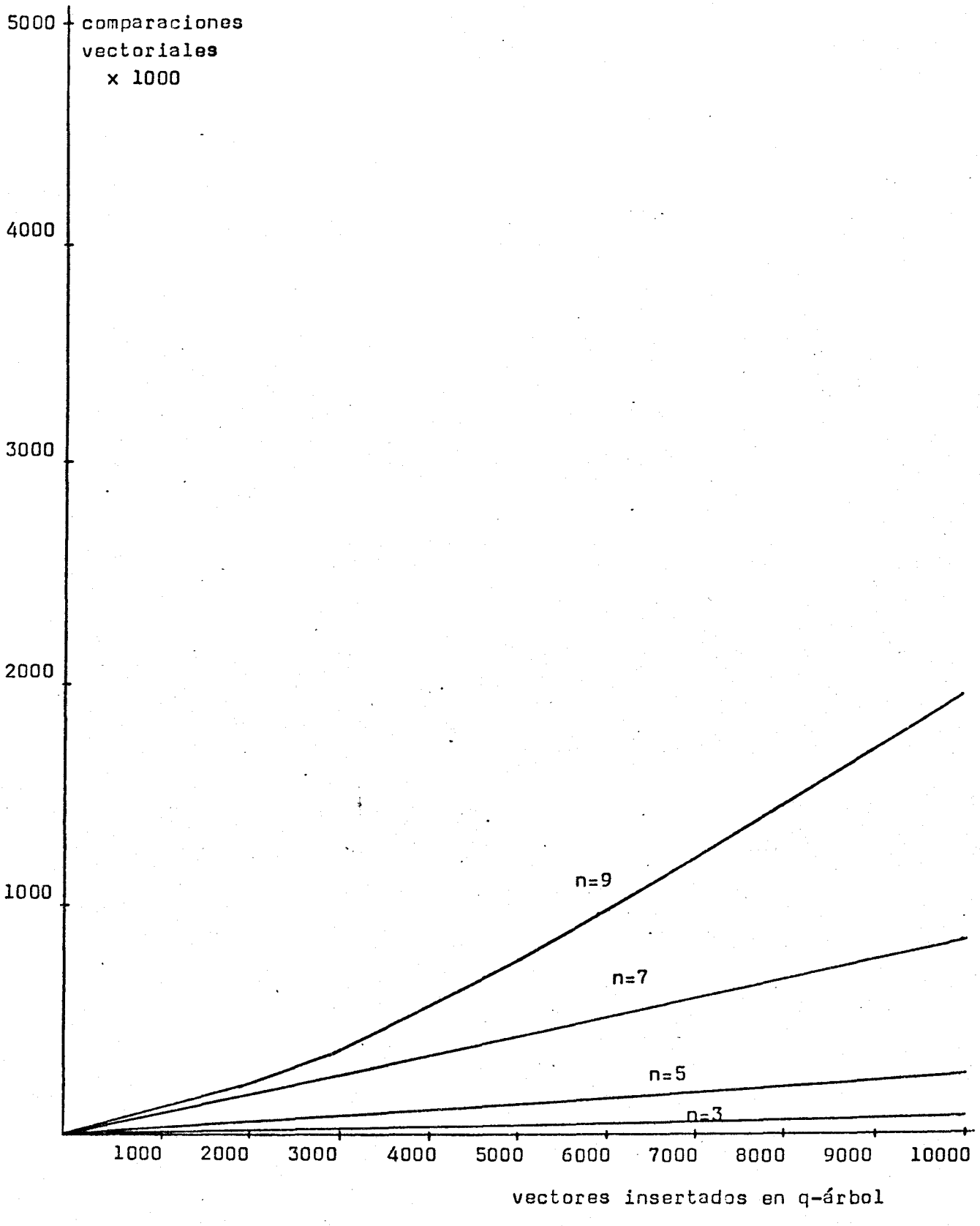

5000
comparaciones
vectoriales
x 1000
4000
3000
2000
1000
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en q-árbol

Figura 8

2.4.- Vectores eficientes sobre kd-árboles

2.4.1.- Construcción del kd-árbol

Sea X $(x_1, x_2, \ldots, x_n)$ un elemento de R^n; con respecto a los hiperplanos coordenados, los semiespacios a los que pertenece vienen identificados por:

$$\delta(x_1), \delta(x_2), \ldots, \delta(x_n)$$

siendo $\delta(x_i) = 1$ si $x_i \geq 0$, lo cual significa que el punto pertenece a este semiespacio $(x_i \geq 0)$, y $\delta(x_i) = 0$ si $x_i < 0$, lo cual significa que X está en el semiespacio $x_i < 0$.

Definamos la función

$$h(X) = (\delta(x_1), \delta(x_2), \ldots, \delta(x_n)) \text{ y } h_i(X) = \delta(x_i)$$

Dados dos elementos $X, Y \in R^n$, los semiespacios que ocupa Y respecto de los hiperplanos que pasan por X paralelos a los coordenados, vienen definidos por:

$$h_i(Y-X) = \delta(y_i - x_i)$$

En un kd-árbol el punto que ocupa la raiz divide el espacio en dos según su primera coordenada; asignando a la raiz el nivel 1 del kd-árbol tenemos que la coordenada que origina la división del espacio es la correspondiente al nivel que ocupa; en el nivel 2, que ocupan los hijos de la raiz, la división se hace según la segunda coordenada de dichos elementos; así sucesivamente, hasta llegar al nivel n en el que la división se efectúa según la coordenada enésima. En el nivel n+1 la división vuelve a

efectuarse según la primera coordenada.

En general, se tiene que en un nivel p, la coordenada que origina la división es la p mod n, es decir, el resto de la división de p entre n.

Si un punto X se encuentra situado en el nivel p, todos los elementos Y situados en su subárbol derecho verificarán que $y_i \geq x_i$ siendo i=p mod n, mientras que los elementos de su subárbol izquierdo verificarán que $y_i < x_i$.

Vemos así, que la información aportada por un nodo depende, no solo del valor de sus coordenadas sino del nivel en el que se encuentre.

Definición: Sean X,Y dos nodos del kd-árbol; Y es llamado un k-hijo de X ($k \leq n$) si:

a.- Y es un sucesor de X y X está en el nivel k+n

b.- X es un predecesor directo de Y

El algoritmo de construcción del kd-árbol, efectuando unicamente inserciones, viene dado como sigue; sea Y el elemento a insertar:

a.- Si el kd-árbol está vacío, colocaremos Y como raiz, y terminaríamos.

b.- Sea X la raiz del kd-árbol

c.- Sea k el nivel de X

Si $y_k \geq x_k$ Y es un sucesor derecho de X

Si $y_k < x_k$ Y es un sucesor izquierdo de X

d.- Si dicha posición está vacía, colocar el vector Y y finalizar.

e.- Si está ocupada, llamar X al elemento que la ocupa y volver a c.

Este algoritmo que construye el kd-árbol es más simple que el correspondiente al q-árbol, fundamentalmente porque las comparaciones que permiten avanzar en el kd-árbol, son comparaciones entre números reales y no vectoriales como ocurría en el q-árbol. Además, en la mayoría de los casos, un gran porcentaje de ramas del q-árbol estaban vacías, debido a su orden de multiplicidad por niveles, mientras que el kd-árbol optimiza en este sentido el aprovechammiento del espacio.

2.4.2.- Identificación de vectores eficientes sobre kd-árboles

Un kd-árbol es de dominancia libre, si no existen en él una pareja de nodos que domine uno al otro; construiremos un kd-árbol manteniéndolo libre de dominancias, planteando para ello dos cuestiones; sea Y el vector a insertar sobre el kd-árbol:

a.- Determinar si existe algún vector X sobre el kd-árbol que domine a Y; si existiese un X que dominase a Y, el vector Y sería rechazado. Esto nos obligará a un recorrido parcial del kd-árbol a la búsqueda de posibles dominadores de Y.

b.- Hallar los vectores del kd-árbol que sean dominados por Y; todos los vectores que verifiquen dicha condición deben ser eliminados del kd-árbol.

Logicamente, si un vector Y elimina a algún otro de los que ya están en el kd-árbol, Y no puede ser dominado por ningún vector del kd-árbol; análogamente, por la transitividad de la dominancia, si un vector Y es dominado por alguno del kd-árbol, entonces Y no puede dominar a ningún otro del kd-árbol.

Dado que la condición para que un vector X domine a otro vector Y es que $y_i \leqslant x_i$ para cualquier valor de i, para localizar si existe algún vector X que domine a Y seguiremos siempre las ramas que verifiquen $y_i \leqslant x_i$; más exactamente, seguiremos todas las ramas para las que se verifique $y_i \not> x_i$, ya que en cada nodo se divide el espacio, y salvo que $y_i > x_i$ puede ocurrir que existan vectores a izquierda y derecha que dominen a X, es decir:

nivel k X

Si $y_k \geqslant x_k$ sus posibles sominadores estan en el subarbol derecho.

$y_k < x_k$ $y_k \geqslant x_k$

Si $y_k < x_k$ sus posibles dominadores estan en el subarbol derecho o en el izquierdo.

En cualquier caso, vemos que con estos criterios quedan eliminados gran cantidad de subárboles tanto en la búsqueda de algún dominador de Y como de vectores dominados por Y, debido a que, aunque tenemos la estructura de un árbol binario, la carga de información que soporta cada nivel, hace que su rendimiento sea mucho más alto que el de aquellos.

2.4.3.- Algoritmo eficiente

Desarrollemos el algoritmo de inserción de un elemento Y en un kd-árbol preservando la relación de dominancia libre; si el kd-árbol estuviese vacío, colocaríamos Y en la raiz, así que supondremos que no es vacío.

a.- Sea X la raiz del kd-árbol

b.- Sea k el nivel en que nos encontramos

c.- Si $y_k < x_k$ ejecutar Test1(Y) en el subárbol derecho de X

c1.- Si Test1(Y) resulta positivo (Y ha sido dominado), hemos terminado.

c2.- Si el subárbol izquierdo de X está vacío, insertar Y, y si no llamar X al hijo izquierdo de X y volver a b.

d.- Si $y_k \geq x_k$ ejecutar Test2(Y) en el subárbol izquierdo de X

e.- Si el subárbol derecho de X está vacío, insertar Y, y si no, llamar X al hijo derecho de X y volver a b.

Test1(Y)

a.- Sea k el nivel en el que nos encontramos; si está vacío, Y no es dominado.

b.- Si el nodo X sobre el que estamos, domina a Y, hemos terminado con test1 positivo.

c.- Si $y_k \geq x_k$ ejecutar Test1(Y) en el subárbol derecho de X

d.- Si $y_k < x_k$ ejecutar Test1(Y) en el subárbol derecho y en el izquierdo de X.

Test2(Y)

a.- Sea k el nivel en el que nos encontramos; si está vacío, hemos terminado.

b.- Si Y domina al nodo X sobre el que nos encontramos, borrar X.

c.- Si $y_k < x_k$ ejecutar Test2(Y) en el subárbol izquierdo de X

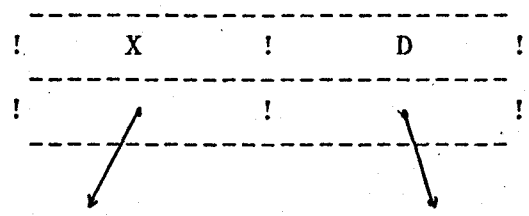
d.- Si y $\geq$ x ejecutar Test2(Y) en el subárbol derecho e izquierdo de X.

2.4.4.- Experiencias computacionales

El programa para la búsqueda de vectores eficientes sobre kd-árboles lo hemos llamado kd-árbol.

Las variables n,nv,co y ti, asi como las funciones igual y dom, tienen un funcionamiento análogo al que tenían en los q-árboles.

El tipo de variable nodo es el tipo base de la estructura; se trata de un registro que almacenará, un vector X, el número de descendientes d y dos punteros iz y de, que guardarán las direcciones de los subárboles izquierdo y derecho de X respectivamente.



el dato d, del número de sucesores podría ser eliminado sin que ello afecte al funcionamiento de la estructura, aunque es un dato interesante si se pretende cierto efecto de equilibrio.

La función test1, comprueba si el vector a es dominado por algún nodo perteneciente al árbol de raiz p, la cual se encuentra en el nivel k; si a no es dominado test1 es cierto y si no, es falso.

La rutina insertar2, inserta el vector a en el kd-árbol de raiz p, la cual se encuentra en el nivel k, cuando sabemos de

antemano que el vector a no domina, ni es dominado por ningún vector del árbol de raiz p. Esta rutina es usada en la reconstrucción de ramas del kd-árbol en las que hemos tenido que borrar ciertos elementos.

La rutina borrar, es la encargada de eliminar del kd-árbol de raiz p, todos los vectores dominados por a, cuando ya sabemos que a domina al vector indicado por p, lo que nos obliga a la reconstrucción; el número de nodos borrados lo almacena la variable c.

Cuando debemos hacer una reconstrucción, usamos un heurístico que da buenos resultados en cuanto a que el subárbol resultante colabore en un posible equilibrio del kd-árbol; para ello la rutina dividir, recorre el camino que va desde la raiz hasta un nodo, de forma tal que a la izquierda de dicho camino queda un número de nodos igual que a la derecha; luego, de vuelta hacia la raiz, aplica la rutina recorrer, que ejecuta un recorrido recursivo del kd-árbol, en el sentido subárbol derecho-raiz-subárbol izquierdo, eliminando los vectores dominados por a, y los restantes los envía a insertar2; esto provoca, normalmente, un kd-árbol tal que la rama izquierda es superior a la rama derecha, lo cual es conveniente, dado que en la mayoría de las inserciones el kd-árbol tiende a perder nodos del subárbol izquierdo y a aumentar por el derecho.

La rutina test2, elimina los vectores dominados por a pertenecientes al kd-árbol de raiz p; cuando procede borrar, hace uso de la rutina correspondiente.

La rutina insertar es el núcleo del programa y es la que lleva el control de la inserción de un elemento a en un kd-árbol con raiz p y de dominancia libre; sigue el algoritmo eficiente ya

estudiado y hace uso de todas las demás funciones y rutinas.

El programa principal inicializa algunas variables y a continuación lee los vectores del fichero de entrada; finalmente, llama a la rutina imprimir que nos muestra los resultados, es decir, los vectores eficientes delconjunto de vectores leidos.

En el cuadro 4 figuran los resultados obtenidos para los kd-árboles, para los mismos conjuntos de vectores que tratamos en las listas y en los q-árboles; puede verse en dicho cuadro y en las figuras 9 y 10, que los resultados obtenidos son mejores que en los otros casos.

La figura 11 muestra una comparación del tiempo en las tres estructuras y podemos confirmar las ventajas de la estructura kd-árbol en la búsqueda de vectores eficientes.

El presente estudio puede generalizarse facilmente a los pseudo kd-árboles, pero dado que el análisis de la eficiencia que estamos haciendo,es fácil ver que para el caso de un vector no dominado, el número de comparaciones es el mismo que para el kd-árbol, es decir, si las coordenadas de división son idénticas en ambos casos; esto se debe a que el pseudo kd-árbol es muy eficiente para insertar o borrar un elemento, pero no supera al kd-árbol para ver si un vector Y es dominado, o borrar todos los elementos dominados por Y.

Si el vector es dominado, el comportamiento también es análogo, aunque mientras que en el kd-árbol la dominancia puede darse en niveles intermedios, en los pseudo kd-árboles, hay que llegar a las hojas antes de comprobar cualquier dominancia.

Programa kd-arbol

```
Program kdarbol(input,output);
const n=5;
type vector=array[1..n] of integer;
     pnodo=^nodo;
     nodo=record x:vector;
           d:integer;
           iz,de:pnodo
     end;
var a:vector;
    raiz:pnodo;
    i,k,a,nv,co,ti:integer;

function igual(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<>w[i]) then t:=false
  end;
  igual:=t;
end;

function dom(v,w:vector):boolean;
var t:boolean;
begin t:=true;i:=0;co:=co+1;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<w[i]) then t:=false
  end;
  if t then if igual(v,w) then t:=false;
  dom:=t
end;

procedure imprimir(p:pnodo;e:integer);
var i:integer;
begin
  if (p<>nil) then
  begin
    imprimir(p^.de,e+1);
    for i:=1 to e do write('   ');
    for i:=1 to n do write(p^.x[i]:3);writeln(p^.d:4);
    imprimir(p^.iz,e+1)
  end;
end;

function test1(a:vector;p:pnodo;k:integer):boolean;
var l:integer;
    t:boolean;
begin t:=true;l:=(k mod n)+1;
  if (p<>nil) then if dom(p^.x,a) then t:=false else
  begin t:=test1(a,p^.de,l);
    if (t and (a[k]<=p^.x[k])) then t:=test1(a,p^.iz,l);
  end;
  test1:=t;
end;
```

```
procedure insertar2(a:vector;var p:pnodo;k:integer);
var l:integer;
begin l:=(k mod n)+1;
  if (p=nil) then
  begin new(p);p^.x:=a;p^.iz:=nil;p^.de:=nil;p^.d:=1
  end else
  begin if a[k]<=p^.x[k] then insertar2(a,p^.iz,l)
    else insertar2(a,p^.de,l);
    p^.d:=p^.d+1
  end;
end;

procedure borrar(a:vector;var p:pnodo;k:integer;var c:integer);
var raiz2:pnodo;
    z:integer;

procedure recorrer(p:pnodo);
begin if (p<>nil) then
  begin recorrer(p^.de);
    recorrer(p^.iz);
    if dom(a,p^.x) then c:=c-1 else insertar2(p^.x,raiz2,k);
    dispose(p);
  end;
end;

procedure dividir(var p:nodo);
begin if ((p<>nil) and (z>0)) then
  begin if (z-p^.d)>=0 then z:=z-p^.d else
    begin z:=z-1;if (z>0) then dividir(p^.de);
      if (z>0) then dividir(p^.iz)
    end;
  end;
  if (z=0) then recorrer(p);p:=nil
end;

begin raiz2:=nil;
  z:=(p^.d+1) div 2;
  dividir(p);
  p:=raiz2;
end;

procedure test2(a:vector;var p:pnodo;k:integer;var e:integer);
var l,j:integer;
begin l:=(k mod n)+1;
  if (p<>nil) then if dom(a,p^.x) then borrar(a,p,k,e) else
  begin j:=e;e:=0;test2(a,p^.iz,l,e);
    if a[k]>p^.x[k] then test2(a,p^.de,l,e);
    p^.d:=p^.d+e;e:=e+j
  end;
end;

procedure insertar(a:vector;var p:pnodo;k:integer;var c:integer);
var j,l:integer;
begin l:=(k mod n)+1;
  if (p=nil) then
  begin new(p);p^.x:=a;p^.iz:=nil;p^.de:=nil;p^.d:=1;c:=1
  end else if not(dom(p^.x,a)) then if dom(a,p^.x) then
  begin p^.x:=a;borrar(a,p,k,c)
  end else
  begin if a[k]<=p^.x[k] then
```

```
    begin if test1(a,p^.de,l) then insertar(a,p^.iz,l,o)
    end else
    begin test2(a,p^.iz,l,o);j:=o;o:=0;insertar(a,p^.de,l,o);o:=o+j
    end;
    p^.d:=p^.d+o
  end;
end;

begin
  ti:=clock;nv:=0;co:=0;
  raiz:=nil;
  while not(eof(input)) do
  begin k:=1;a:=0;
    for i:=1 to n  do read(a[i]);nv:=nv+1;
    if eoln(input) then readln;
    insertar(a,raiz,k,a);
  end;
  ti:=clock-ti;
  writeln(nv,ti,co);
  imprimir(raiz,0)
end.
```

	1000	2000	3000	4000	5000
	37	30	37	48	40
3	1030	2100	2630	3450	42800
	5651	11325	15816	20534	25016
	184	249	292	326	371
5	5990	11900	17910	24130	30370
	29568	58188	89048	120495	151836
	417	667	850	1056	1097
7	14520	35090	56620	80750	104380
	69866	166007	268507	382934	493918
	695	1201	1624	2082	2478
9	27010	69500	120620	175800	236270
	121179	309338	533994	777500	1028159

	6000	7000	8000	9000	10000
	42	42	44	48	48
3	5020	5690	6380	7000	7640
	29023	32782	36508	40136	43750
	421	437	469	493	516
5	37320	43990	49810	55580	61800
	185812	219193	248991	278610	309763
	1245	1330	1365	1485	1503
7	124800	146630	168060	185780	204720
	591358	694406	784839	868121	954205
	2820	3117	3471	3769	4038
9	297780	357620	421780	495770	567660
	1297734	1559458	1841767	2139044	2451621

Cuadro 4

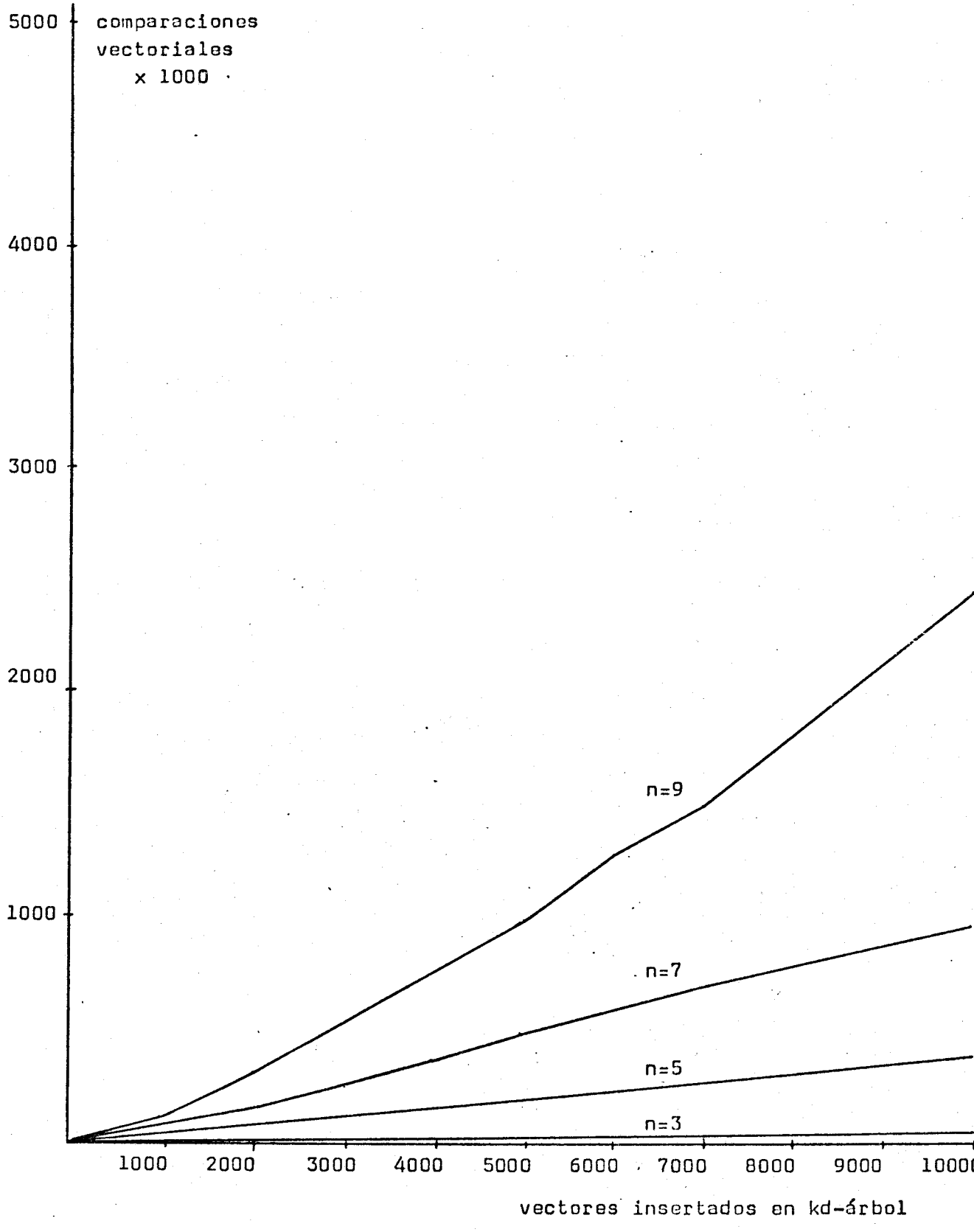

5000
comparaciones
vectoriales
x 1000
4000
3000
2000
1000
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en kd-árbol

Figura 9

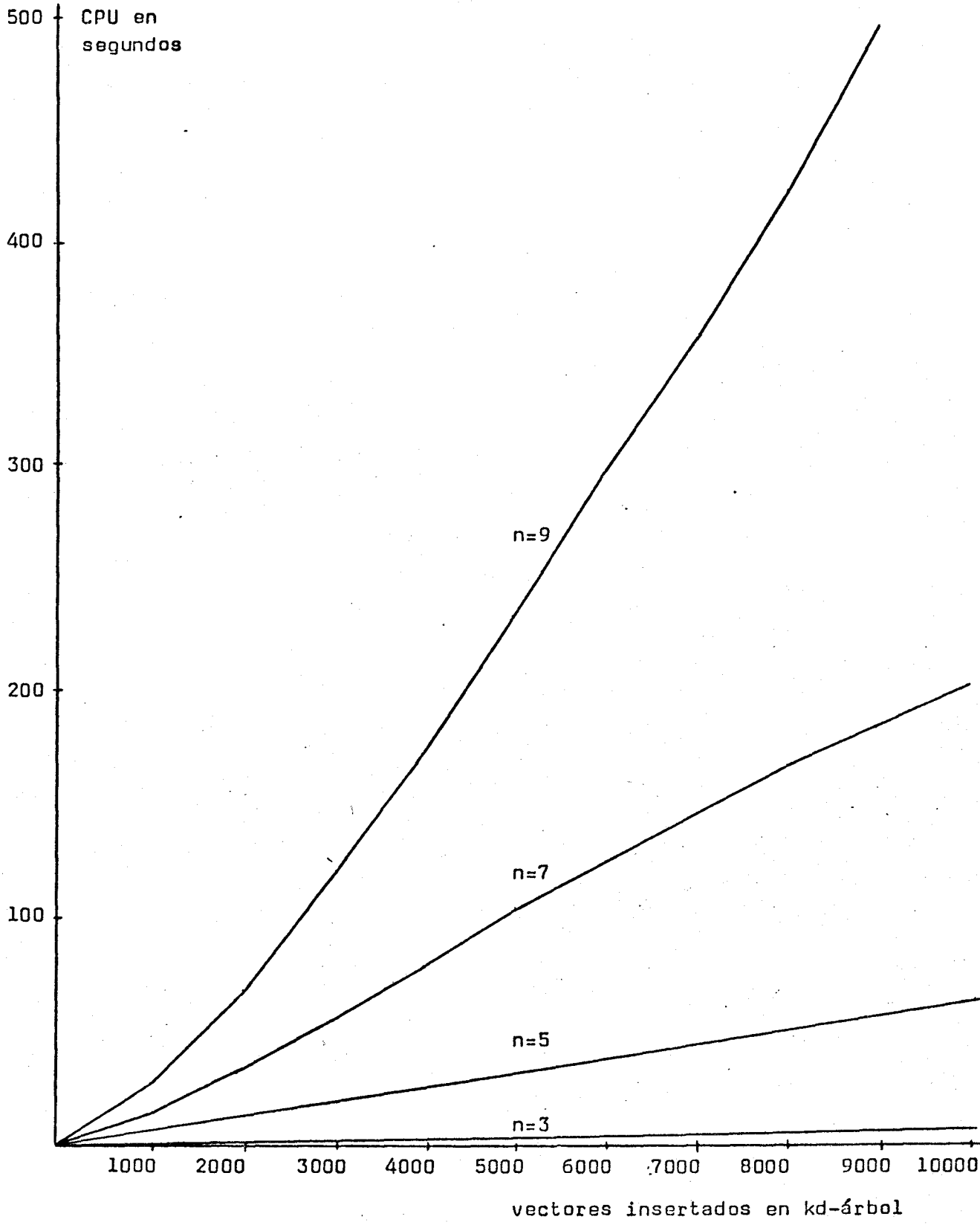

500
CPU en
segundos
400
300
200
100
n=9
n=7
n=5
n=3
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados en kd-árbol

Figura 10

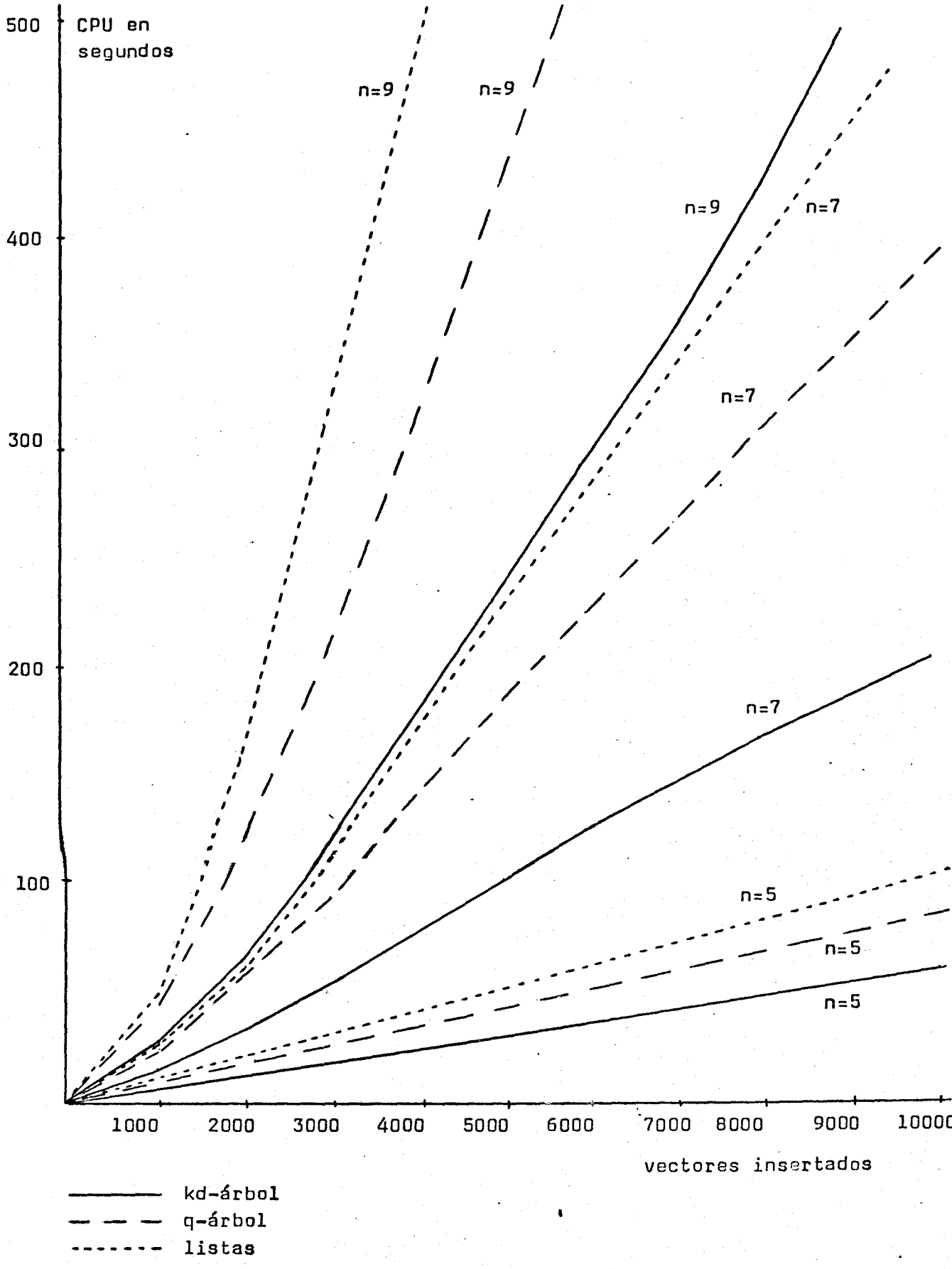

500
CPU en
segundos
400
300
200
100
n=9
n=9
n=9
n=7
n=7
n=7
n=5
n=5
n=5
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
vectores insertados
kd-árbol
q-árbol
listas

Figura 11

3.- Puntos eficientes enteros en poliedros

3.1.- Introducción

Consideremos el poliedro P definido por

$$\begin{aligned} \sum_{j=1}^{n} a_{ij}x_j &\leqslant b_i \quad i=1,2,\ldots,m \\ x_j &\geqslant 0 \quad j=1,2,\ldots,n \\ a_{ij}, b_i &\ R \quad \forall i,j \end{aligned} \qquad (1)$$

que supondremos acotado; nuestro propósito es localizar los puntos enteros eficientes según Pareto en P.

Veamos a continuación la notación y algunos conceptos que utilizaremos en los siguientes apartados.

El conjunto factible lo designaremos por χ y representará el conjunto de puntos de R^n, cuyas coordenadas verifican las inecuaciones (1).

Llamemos H_i al hiperplano determinado por la i-ésima ecuación de (1), es decir:

$$H : \sum_{j=1}^{n} a_{ij}\, x_j = b_i$$

la superficie poliédrica S la integran los puntos de χ que pertenecen además, a algún hiperplano H ; se tiene:

$$S = \bigcup_{i=1}^{m} \{\chi \cap H_i\}$$

El subconjunto de puntos de S que son óptimos para alguna dirección $z = \alpha_1 x_1 + \alpha_2 x_2 + \text{----} + \alpha_n x_n$ con al menos un coeficiente $\alpha_i > 0$, para el problema de maximización de z, lo indicaremos por S_o.

Llamemos S_o^o al subconjunto de S_o que está formado por los puntos que no son óptimos para alguna dirección

$z = \alpha_1 x_1 - \alpha_2 x_2 - \cdots - \alpha_n x_n$ con $\alpha_i \leq 0$ para cualquier i.

La frontera eficiente F_e representará los puntos de S que son óptimos para alguna dirección $z = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n$ con $\alpha_i > 0$ para el problema de maximización de z; F_e está formada por todos los puntos eficientes Pareto de P.

Llamemos F_e^o al subconjunto de F_e que está formado por los puntos de F_e que no son óptimos para alguna dirección $z = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n$ con $\alpha_i \geq 0$ para cualquier i, y al menos un $\alpha_j = 0$.

Esto implica que el conjunto de direcciones para las que un punto de F_e^o es óptimo no contiene direcciones paralelas a los ejes.

Designemos por χ^* el conjunto de puntos de χ de coordenadas enteras, y por χ_e^* el conjunto de puntos eficientes enteros, es decir, los puntos de χ^* que no son dominados según Pareto, por ningún otro punto de χ^*.

Una condición necesaria para que un punto de χ^* pertenezca a χ_e^*, es que no pueda incrementarse en una unidad ninguna de sus coordenadas, manteniendo la factibilidad; tal condición no es suficiente.

3.2.- Puntos enteros en un entorno de la superficie poliédrica

El entorno de radio k según la norma 1 de un punto X viene definido por:

$$B(X,k) = \left\{ Y:\ y_i = x_i + \varepsilon_i \ \sum_{i=1}^{n} |\varepsilon_i| < k \right\}$$

nos interesan los entornos para valores de k=0,1,2,.....

Es claro, que para todo punto X de χ existe un $Y \in F_e$ y un valor de k tal que $X \in B(Y,k)$ con $y_i = x_i + \varepsilon_i$ siendo $\varepsilon_i \geq 0$; en efecto, si $X \in F_e$ puedo tomar Y=X y cualquier valor de $k \in N$; mientras que si $X \notin F_e$, debe estar dominado por algún punto eficiente $Y \in F_e$ y entonces $X \in B(Y,k)$ para $k = \left\lceil \sum_{i=1}^{n} (y_i - x_i) \right\rceil$.

Dado un poliedro P, se define la atmósfera entera en el entorno de radio k de F_e por:

$$A(k,F_e) = \left\{ X:\ X \in \chi^* \text{ y } X \in B(Y,k) \quad \text{para algún} \quad Y \in F_e \right\}$$

k=0,1,....

$A(0,F_e)$ está formado por los puntos de $\chi_e^* \cap F_e$.

$A(1,F_e)$ está formado por los puntos de χ^* que pertenecen al entorno B(Y,1) de algún punto de F_e .

En general, A(k,Fe) es el conjunto de puntos de χ^* que pertenecen al entorno de radio k de algún punto de F_e.

Si $\chi^* = \emptyset$ todos los $A(k,F_e)$ son vacíos, pero si $\chi^* \neq \emptyset$ y dado que:

$$A(0,F_e) \subset A(1,F_e) \subset \text{-------} \subset A(k,F_e) \subset A(k+1,F_e) \subset \text{--}$$

pueden ser vacíos hasta un cierto valor de $k=k_0$, a partir del cual serán no vacíos; además, como el poliedro es acotado, a partir de un valor de $k=k_1$, se tendrá que los conjuntos $A(k,F_e)$ coinciden con χ^*.

Teorema 1.- Si $X \in A(k,F_\Theta)$ se verifica que al menos uno de los puntos $X_i = X + ke_i$ no es factible.

Demostración: sea $X \in A(k,F_\Theta)$ e Y un punto eficiente perteneciente a F_Θ que domine a X y tal que $X \in B(Y,k)$; este punto Y existe como podemos ver, llamando:

$$D = \{ Z:\ Z \in B(X,k) \ \text{ con } x_i \leq z_i \ \forall\ i \}$$

si no existiera tal Y, se tendría que $D \cap F_\Theta = \emptyset$ y $D \subset P$, lo cual es imposible por ser $X \in A(k,F_\Theta)$; en efecto, $B(X,k) \cap F_\Theta \neq \emptyset$ implica que existe un punto $Y' \in B(X,k) \cap F_\Theta$ y por ser $Y' \in F_\Theta$ $y'_i = x_i + \varepsilon_i$ con $\sum_{i=1}^{n} |\varepsilon_i| < k$, pero esto es una contradicción porque el punto Z: $z_i = x_i + |\varepsilon_i|$ $Z \in D \subset P$, Z es factible y domina a Y', luego $Y' \notin F_\Theta$, contra la hipótesis.

Así pues, $D \cap F_\Theta \neq \emptyset$ y existe $Y \in D \cap F_\Theta$, luego $X \in B(Y,k)$ $y_i = x_i + \varepsilon_i$ $\varepsilon_i \geq 0$ y $D \not\subset P$; como P es convexo y los puntos X_i son los extremos de D no puede ocurrir que $X_i \in P$ para todo i, ya que D no está contenido en P.

c.q.d.

En el poliedro P se define la atmósfera entera en el entorno de radio k de S por:

$$A(k,S) = \{ X:\ X \in \mathbb{X}^* \text{ y } X \in B(Y,k) \text{ para algún } Y \in S \}$$

A(0,S) está formado por los puntos enteros solbre la superficie poliédrica.

A(k,S) (para $k \in N$) está formado por los puntos con coordenadas enteras que pertenecen al entorno B(X,k) de algún punto X de S.

Se verifica que:

$$A(0,S) \subset A(1,S) \subset \text{-------} \subset A(k,S) \subset A(k+1,S) \subset \text{--}$$

aunque A(0,S) puede ser vacío, se tiene que si A(1,S) es vacío—

entonces $\chi^* = \phi$, como demuestra el siguiente teorema.

Teorema 2.- Si $X \in A(k,S)$ se verifica que al menos uno de los puntos $X \pm ke_i$ no es factible.

Demostración: Si $X \in A(k,S)$ entonces $B(X,k) \cap S \neq \phi$, sea $Y \in B(X,k) \cap S$ $y_i = x_i + \varepsilon_i$ $\sum |\varepsilon_i| < k$ y sea:

$$D = \{ Z: \quad Z \in B(X,k) \quad z_i = x_i + ksg(\varepsilon_i)$$

como $Y \in D \cap S$ se verifica que $D \notin P$.

Luego al menos uno de los vértices $X + sg(\varepsilon_i)ke_i$ no es factible.

c.q.d.

Corolario.- Si $A(1,S) = \phi$ entonces $\chi^* = \phi$.

Demostración: Si $A(k,S) = \phi$ para todo k, es evidente que $\chi^* = \phi$; supongamos que $A(1,S) = \phi$ y que $\chi^* \neq \phi$; debe existir un k tal que $A(k,S) \neq \phi$; sea $X \in A(k,S)$, es decir, $X \in B(Y,k)$ para algún $Y \in S$; según elteorema, alguno de los puntos $X \pm ke_i$ no es factible; supongamos que $X + ke_j \notin \chi$ y formemos la sucesión:

$$X, \; X+e_j, X+2e_j, \; \text{------}, \; X+ke_j$$

como $X \in \chi$ y $X + ke_j \notin \chi$, debe existir un k_o tal que $X + k_o e_j \in \chi$ y $X + (k_o+1)e_j \notin \chi$, por lo que $X + k_o e_j \in A(1,S)$ contra la hipótesis de que era vacío.

c.q.d.

Por lo tanto $A(k,S)$ está formado por los puntos de coordenadas enteras que, siendo factibles, distan de S menos de k unidades para al menos una de sus coordenadas.

De la misma forma se puede definir la atmósfera entera en el entorno de radio k de S por:

$$A(k,S_o) = \{ X: \quad X \in \chi^* \text{ y } X \in B(Y,k) \text{ para algún } Y \in S_o \}$$

Aquí también se verifica que $A(0,S_0)$ puede ser vacío, pero si $A(1,S_0)=\phi$ entonces debe ser $X^*=\phi$.

Puesto que $S_0 \subset S$ también se verifica el teorema que demostramos en S que nos decía, que si $X \pm ke_i \in S_0$ entonces al menos uno de los puntos $X \pm ke_i$ no es factible.

Estudiaremos fundamentalmente $A(2,E_e)$ y $A(2,S_0)$; usaremos $A(2,F_e)$ en los casos para los que $X_e^* \subset A(2,F_e)$, mientras que tendremos que recorrer una zona más amplia en otros casos.

Claramente, recorriendo $A(1,S_0)$ o $A(2,S_0)$ (si es posible) podemos ir separando los puntos eficientes, no obstante, estudiaremos la forma de recorrer la mínima parte de $A(1,S_0)$ o $A(2,S_0)$, que nos permita alcanzar todos los puntos eficientes.

La idea es, recorrer ésta mínima parte de manera que tengamos la garantía de que llegaremos a todos los puntos eficientes; mientras tanto, todos los puntos visitados que sean factibles, los enviaremos a una estructura eficiente de las estudiadas en el capítulo anterior y que, al final, nos dará todos los puntos eficientes del poliedro.

Teorema 3.- Si $X \in A(2,S_0)$, existe al menos un i y un valor de k tal que el punto $X_k(x_1,\ldots,x_i+k,\ldots,x_n) \in X$ mientras que el punto $X_{k+1}(x_1,\ldots,x_i+k+1,\ldots,x_n) \notin X$, siendo además $X_j(x_1,\ldots,x_i+j,\ldots,x_n) \in A(2,S_0)$ para $j=1,2,\ldots,k$.

Demostración: Si $X \in A(2,S_0)$ existe $Y \in S_0$ tal que $X \in B(Y,2)$; Y es óptimo con respecto a alguna dirección $z=\alpha_1 x_1+\alpha_2 x_2+ \cdots \alpha_n x_n$ con algún $\alpha_i > 0$; veremos que el punto $X_1(x_1,\ldots,x_i+1,\ldots,x_n)$ si es factible, pertenece a $A(2,S_0)$.

En efecto, si X_1 no es factible, ya está demostrado el teorema y es k=0; supongamos entonces que X_1 es factible; como el

punto Y es óptimo en la dirección z con $\alpha_i > 0$, Y pertenece a una cara determinada por un hiperplano $H_j : a_{j1}x_1 + \cdots + a_{ji}x_i + \cdots + a_{jn}x_n = b_j$ con $a_{ji} > 0$ y el punto $Y_1 (y_1, \ldots, y_i + 1, \ldots, y_n)$ no puede ser factible, porque Y no sería óptimo.

Las desigualdades que Y_1 no verifica han de ser necesariamente alguna con coeficiente de x_i positivo (ya que Y sí las verifica); así pues, el segmento que une Y_1 con X_1 solo cortará a ciertos hiperplanos con coeficiente de x_i positivo.

Llamando Z al punto de corte de dicho segmento con el poliedro se tiene que:

$$X \in B(Y,2) \quad \Longrightarrow \quad X_1 \in B(Y_1,2)$$

por ser:

$$\sum_{i=1}^{n} |y_i^1 - x_i^1| = |y_1 - x_1| + |y_2 - x_2| + \cdots + |y_i + 1 - (x_i + 1)| + \cdots + |y_n - x_n| =$$

$$\sum_{i=1}^{n} |y_i - x_i| < 2$$

y además como $Z = \lambda X_1 + (1-\lambda) Y_1 \qquad 0 \leq \lambda \leq 1$

$$\sum_{i=1}^{n} |z_i - x_i^1| = \sum |\lambda x_i^1 + (1-\lambda) y_i^1 - x_i^1| = \sum |(1-\lambda) y_i^1 - (1-\lambda) x_i^1| =$$

$$\sum (1-\lambda) |y_i^1 - x_i^1| = (1-\lambda) \sum |y_i^1 - x_i^1| \leq 2(1-\lambda) \leq 2$$

por lo tanto, $X_1 \in B(Z,2)$ y como Z pertenece a una cara con coeficiente de x positivo, es óptimo en la dirección de dicha cara, por lo que $Z \in S_o$ y $X_1 \in A(2, S_o)$.

Repitiendo el razonamiento, llegaríamos al punto $X_2(x_1, \ldots, x_i + 2, \ldots, x_n)$

y como el poliedro es acotado, la existencia de $X_k \in \mathcal{X}$ y de $X_{k+1} \notin \mathcal{X}$ con $X_j \in A(2,S_o)$ para $j=1,\ldots,k$ está garantizada.

Teorema 4.- Sea $X \in A(1,S)$, se verifica que todos los vértices del entorno $B(X,1)$ que sean factibles pertenecen a $A(2,S)$.

Demostración:

$$X \in A(1,S) \Longrightarrow X \in B(Y,1) \qquad Y \in S \Longrightarrow \sum_{i=1}^{n} |y_i - x_i| < 1$$

Sea $Z(z_1, z_2, \ldots, z_n)$ con $z_i = x_i$ salvo para un j $z_j = x_j \pm 1$

Se tiene:

$$\sum_{i=1}^{n} |z_i - y_i| = \sum_{\substack{i=1 \\ i \neq j}}^{n} |x_i - y_i| + |x_j \pm 1 - y_i| \leq \sum_{i=1}^{n} |y_i - x_i| + 1 < 2$$

luego $Z \in B(Y,2)$

Como el punto de la superficie es el mismo para X y Z se tiene:

Corolario: Si $X \in A(1,S_o)$, todos los vértices del entorno $B(X,1)$ que sean factibles, pertenecen a $A(2,S_o)$.

3.3.- Poliedro denso entero

Definición: Un poliedro P es denso entero, si para cada par de puntos $X,Y \in \mathcal{X}$ y de coordenadas enteras, al menos uno de los puntos:

$$X_i = X+\delta_i e_i \text{ con } \delta_i = sg(y_i-x_i)$$

distinto de X, es factible.

Teorema 1.- Si P es denso entero, se verifiaca que dados dos puntos $X,Y \in \mathcal{X}^*$, pueden ser unidos por una sucesión de arcos uno a uno direccionales contenidos en el paralelepípedo determinado por los puntos X,Y.

(Entenderemos por arco uno a uno direccional, todo segmento de amplitud uno, paralelo a alguno de los ejes coordenados).

Demostración: Por la definición de poliedro denso entero, podemos obtener una sucesión de puntos:

$$X=X_1,X_2,.....,X_p=Y$$

en la que X_{i+1} es el resultado de aplicar la definición de poliedro denso entero a la pareja de puntos X_i,Y.

Si suponemos una reordenación de las coordenadas, lo cual no supone pérdida de generalidad, se tiene:

$$X(x_1,x_2,...,x_j,x_{j+1},...,x_k,x_{k+1},...,x_n)$$

$$Y(y_1,y_2,...,y_j,y_{j+1},...,y_k,y_{k+1},...,y_n)$$

sea:

$$x_i < y_i \quad i=1,...,j$$

$$x_i > y_i \quad i=j+1,...,k$$

$$x_i = y_i \quad i=k+1,\ldots,n$$

se verifica que siendo $X_i(x_1^i, x_2^i, \ldots, x_n^i)$ es:

$$x_i = x_i^1 \leqslant x_i^2 \leqslant \text{-------} \leqslant x_i^p = y_i \quad i=1,\ldots,j$$
$$x_i = x_i^1 \geqslant x_i^2 \geqslant \text{-------} \geqslant x_i^p = y_i \quad i=j+1,\ldots k$$
$$x_i = x_i^1 = x_i^2 = \text{-------} = x_i^p = y_i \quad i=k+1,\ldots n$$

en cada paso se altera una única coordenada; el número total de pasos es:

$$p-1 = \sum_{i=1}^{j} (y_i - x_i) + \sum_{i=j+1}^{k} (x_i - y_i)$$

c.q.d.

Teorema2.- Si P es un poliedro denso entero, el poliedro P_1 que resulta de añadir a P las restricciones $x_i \geqslant k_i$, es también denso entero.

Demostración: Si $X,Y \in X_1^*$ (enteros de P_1), entonces $X,Y \in X^*$ ya que $X_1^* \subset X^*$ y ambos pueden ser unidos por un camino uno a uno direccional contenido en el paralelepípedo que determinan; este camino no cortará al hiperplano $x_i = k_i$ ya que las coordenadas i-ésimas de X e Y han de ser mayores o iguales que k_i, por ser ambos, puntos pertenecientes a P_1.

c.q.d.

Corolario: Si P es un poliedro denso entero, el poliedro P_1 que resulta de añadir a P las restricciones $x_i \leqslant k_i$ es también denso entero.

Teorema 3.- Si P es un poliedro denso entero, se verifica que dados dos puntos $X,Y \in P$ de coordenadas enteras, eficientes, pueden ser unidos por una sucesión de arcos uno a uno

direccionales de forma que todos los puntos intermedios pertenecen a $A(2,S_o)$.

Demostración: supongamos que los puntos vienen dados por:

$$X(x_1,x_2,\ldots,x_j,x_{j+1},\ldots,x_k,x_{k+1},\ldots,x_n)$$

$$Y(y_1,y_2,\ldots,y_j,y_{j+1},\ldots,y_k,y_{k+1},\ldots,y_n)$$

con:

$$x_i < y_i \quad i=1,\ldots,j$$
$$x_i > y_i \quad i=j+1,\ldots,k$$
$$x_i = y_i \quad i=k+1,\ldots,n$$

Formaremos una sucesión de puntos:

$$X_1,X_2,\ldots\ldots,X_p$$

de manera que $X_1=X$, $X_p=Y$ y $X_j \in A(2,S_o)$ $j=1,2,\ldots,p$

Llamemos $I=\{1,2,\ldots,j\}$ y $J=\{j+1,\ldots,k\}$; cualquier coordenada x_i que crezca, ha de ser para $i \in I$, mientras que si decrece será $i \in J$.

La sucesión de puntos exigida por el teorema viene dada por el siguiente algoritmo, donde supondremos $X \neq Y$, ya que si $X=Y$ es trivial.

a.- Por ser P denso entero, y ninguna coordenada x_i de X $i \in I$ puede crecer, existe alguna coordenada x_i $i \in J$ $x_i \geq y_i$ que puede decrecer; esto origina un nuevo punto Z que pertenece a $A(2,S_o)$ ya que $X \in A(1,S_o)$.

b.- Si alguna coordenada z_i $i \in I$ con $z_i < y_i$ puede aumentar en una unidad, incrementarla. Llamar Z al nuevo punto así obtenido y volver a b.

c.- Si $z_i = y_i$ paratodo $i \in I$ también ha de ser $z_i = y_i$ para

todo $i \in J$, ya que, en otro caso, Y no sería un punto eficiente; tendríamos Z=Y y habríamos terminado.

d.- Volver al apartado a, llamando X a Z.

Puesto que el número de pasos que separa X de Y es finito y venía dado por:

$$p-1= \sum_{i=1}^{j}(y_i - x_i) + \sum_{i=j+1}^{k}(x_i - y_i) = \sum_{i=1}^{n}|y_i - x_i|$$

el algoritmo anterior es finito.

Nos falta probar que los puntos Z, obtenidos en el apartado b, pertenecen a $A(2,S_0)$, es decir, si el punto $X \in \mathcal{X}$ y $X+e_i \notin \mathcal{X}$, mientras que $X-e_t \in \mathcal{X}$ entonces si $Z=X-e_t+ke_i \in \mathcal{X}$ ha de ser $Z \in A(2,S_0)$.

En efecto, como $X \in \mathcal{X}$ y $X+e_i \notin \mathcal{X}$ implica que $X+ke_i \notin \mathcal{X}$ $(k \geq 1)$, entonces si $Z \in \mathcal{X}$ ha de ser $Z \in A(2,S_0)$

De hecho, al ser $X+ke_i \notin \mathcal{X}$ y $X+ke_i-e_t \in \mathcal{X}$, Z pertenece a $A(1,S_0)$.

c.q.d.

Corolario: Si P es un poliedro denso entero y X,Y son dos puntos eficientes de coordenadas enteras, podemos unir X con Y por arcos uno a uno direccionales, de forma que no existen dos puntos consecutivos pertenecientes a $A(2,S_0)-A(1,S_0)$, y cualquier punto intermedio pertenece al paralelepípedo determinado por los puntos X,Y.

Demostración: Siguiendo el algoritmo del teorema anterior, vemos que unicamente al decrecer una coordenada en el apartado a, podemos salir de $A(1,S_0)$ y entrar en $A(2,S_0)-A(1,S_0)$, pero inmediatamente, al aplicar b, volvemos a $A(1,S_0)$; si la

aplicación de b, no fuese procedente, significaría que en a, no llegamos a salir de $A(1,S_0)$.

c.q.d.

Teorema 4.- Si $a_{ij} \geq 0 \ \forall i,j$, en el poliedro:

$$P: \sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad i=1,\ldots,m \qquad x_j \geq 0$$

se verifica que P es denso entero.

Demostración: Sean $X,Y \in X^*$:

$$X(x_1,x_2,\ldots,x_j,x_{j+1},\ldots,x_k,x_{k+1},\ldots,x_n)$$

$$Y(y_1,y_2,\ldots,y_j,y_{j+1},\ldots,y_k,y_{k+1},\ldots,y_n)$$

y supongamos:

$$x_i < y_i \quad i=1,\ldots,j$$
$$x_i > y_i \quad i=j+1,\ldots,k$$
$$x_i = y_i \quad i=k+1,\ldots,n$$

Si llamamos A a la matriz de los coeficientes $A=(a_{ij})_{m\,n}$, $B=(b_1,\ldots,b_m)'$ y $A_j=(a_{1j},a_{2j},\ldots,a_{mj})$, se tiene:

$$AX = x_1A_1 + \text{------} + x_nA_n \leq B$$

$$AY = y_1A_1 + \text{------} + y_nA_n \leq B$$

El punto $Z(x_1,\ldots,x_j,y_{j+1},\ldots,y_k,x_{k+1},\ldots x_n)$ es factible ya que:

$$AZ = x_1A_1 + \text{---} + x_jA_j + y_{j+1}A_{j+1} + \text{---} + y_kA_k + x_{k+1}A_{k+1} + \text{---} + x_nA_n \leq$$

$$x_1A_1 + \text{---} + x_jA_j + x_{j+1}A_{j+1} + \text{---} + x_kA_k + x_{k+1}A_{k+1} + \text{---} + x_nA_n =$$

$$= AX \leq B$$

análogamente se vería que: $AZ \leqslant AY \leqslant B$

Dado que los puntos:

$$X_1 (x_1+\Delta x_1 , \ldots , x_j +\Delta x_j , y_{j+1} , \ldots , y_k , y_{k+1} , \ldots , y_n)$$

e

$$Y_1 (x_1 , \ldots , x_j , y_{j+1} +\Delta y_{j+1} , \ldots , y_k +\Delta y_k , x_{k+1} , \ldots , x_n)$$

con $0 \leqslant \Delta x_i \leqslant y_i - x_i$ $i=1,\ldots,j$ y $0 \leqslant \Delta y_i \leqslant x_i - y_i$ $i=j+1,\ldots,k$ verifican que:

$$AX_1 \leqslant AY \leqslant B$$

$$AY_1 \leqslant AX \leqslant B$$

X_1 e Y_1 son factibles y determinan caminos uno a uno direccionales que unen Z con X y Z con Y, contenidos en el paralelepípedo determinado por Z,X y Z,Y; así, tenemos determinado un camino que une X con Y, uno a uno direccional y dentro del paralelepípedo que determinan.

c.q.d.

3.4.- Puntos eficientes en un poliedro denso entero

3.4.1.- Algoritmo de búsqueda

Hemos demostrado que toda pareja de puntos eficientes en un poliedro denso entero, puede ser unida por un camino uno a uno direccional, cuyos puntos pertenecen a $A(1,S_0)$, salvo ciertos puntos aislados que pueden pertenecer a $A(2,S_0)-A(1,S_0)$.

El punto inicial será pues, un punto eficiente entero; este punto puede ser obtenido a partir del óptimo real en alguna dirección $z=\alpha_1 x_1+\alpha_2 x_2+\text{-----}+\alpha_n x_n$ con $\alpha_i \geqslant 0$ para todo i; sea $Y(y_1,y_2,\ldots,y_n)$ dicho punto, que verificará:

$$AY=y_1A_1+y_2A_2+\text{-----}+y_nA_n \leqslant B$$

Sea $a_{ij} \geqslant 0 \ \forall i,j$, el punto $Z(z_1,z_2,\ldots,z_n)$ tal que $z_i=\lfloor y_i \rfloor$ verifica:

$$AZ=z_1A_1+z_2A_2+\text{----}z_nA_n=\lfloor y_1 \rfloor A_1+\lfloor y_2 \rfloor A_2+\text{-----}+\lfloor y_n \rfloor A_n \leqslant$$

$$y_1A_1+y_2A_2+\text{----}+y_nA_n=AY \leqslant B$$

es decir, el punto Z es factible y de coordenadas enteras; aplicando a Z el procedimiento de aumentar cualquiera de sus coordenadas, siempre que se mantenga la factibilidad, reiteradamente, hasta que no se pueda aumentar ninguna de ellas, llegaremos a un punto X que será el punto inicial eficiente.

El punto $X(x_1,x_2,\ldots,x_n)$ divide el espacio, y por consiguiente el poliedro, en 2^n cuadrantes, en cada uno de los cuales buscaremos los puntos eficientes; para evitar que podamos llegar por distintos caminos a un mismo punto del poliedro,

tomaremos como puntos iniciales de dichos 2^n cuadrantes, los puntos correspondientes a los vértices de la bola unidad con norma l_∞, cuyo vértice superior es X, teniendo además cada uno de ellos un sentido de movimiento.

Así, en el caso n=3 con el punto $X(x_1,x_2,x_3)$, se originan las ramas:

$$X_1(\underline{x}_1,\underline{x}_2,\underline{x}_3) \qquad X_2(\overline{x_1^{-1}},\underline{x}_2,\underline{x}_3)$$
$$X_3(\underline{x}_1,\overline{x_2^{-1}},\underline{x}_3) \qquad X_4(\underline{x}_1,\underline{x}_2,\overline{x_3^{-1}})$$
$$X_5(\overline{x_1^{-1}},\overline{x_2^{-1}},\underline{x}_3) \qquad X_6(\overline{x_1^{-1}},\underline{x}_2,\overline{x_3^{-1}})$$
$$X_7(\underline{x}_1,\overline{x_2^{-1}},\overline{x_3^{-1}}) \qquad X_8(\overline{x_1^{-1}},\overline{x_2^{-1}},\overline{x_3^{-1}})$$

donde la raya inferior significa que la coordenada solo puede crecer y la superior, que solo puede decrecer.

Como el punto $X\in X_e^*$ las ramas 1 y 8 pueden ser abandonadas, ya que en X_1 no podrá aumentar ninguna de sus coordenadas por ser eficiente, y X_8 solo conduciría a puntos dominados por X; evidentemente, y por ser P denso entero, las zonas de búsqueda son conexas uno a uno, y disjuntas, por lo que está garantizado que llegaremos a todos los puntos eficientes.

Como hemos demostrado, pueden restringirse los movimientos a la zona $A(1,S_o)$, salvo algún punto aislado que podría estar en $A(2,S_o)-A(1,S_o)$; algebraicamente, las zonas $A(1,S_o)$ y $A(2,S_o)$, vienen definidas por:

$$P_1 : \sum_{j=1}^{n} a_{ij}x_j \leqslant b'_j \text{ con } b'_j = b_j - \max_j(|a_{ij}|) = b_1 - \max(a_{ij})$$

$$A(1,S_o) = P \cap P_1^c$$

$$P_2 : \sum_{j=1}^{n} a_{ij}x_j \leqslant b''_j \text{ con } b''_j = b_j - 2\max(|a_{ij}|) = b_j - 2\max(a_{ij})$$

$$A(2,S_o) = P \cap P_2^c$$

La indicación de que nos hemos salido de $A(1,S_0)$, es que el punto sobre el que nos encontramos sea factible en P_1 ; en este caso, debe crecer alguna coordenada que nos devuelva a $A(1,S_0)$.

Como la búsqueda la estamos haciendo por cuadrantes, sabemos las variables que pueden crecer o decrecer en cada una de las ramas, por lo que, en lugar de P podemos considerar:

$$P_1' : \sum_{j=1}^{n} a_{ij}\, x_j \leq b_i' \quad \text{con } b_i' = b_i - \max_{j\in I}(a_{ij})$$

siendo I el conjunto de coordenadas que van creciendo. La zona $P \cap P_1'^{c}$ es, generalmente, menor que $A(1,S_0)$, con lo cual restringiremos la zona de búsqueda.

Puede demostrarse facilmente que el punto inicial de cada cuadrante puede unirse con cualquier punto eficiente de dicho cuadrante, a través de un camino uno a uno direccional cuyos puntos están contenidos en $P \cap P_1'^{c}$, salvo algunos que lo están en $A(2,S_0)$, y desde los cuales los únicos movimientos permitidos son los de crecimiento en alguna de sus coordenadas (análogo al teorema 3).

El algoritmo que damos a continuación es válido para cada rama y X representa inicialmente, el punto de partida de cada rama; junto a cada punto X usaremos un vector de estado $E(e_1,e_2,\ldots,e_n)$ tal que $e_i=1$ si la coordenada x va creciendo, $e_i=-1$ si va decreciendo y $e_i=0$ si está fija.

1.- Creación de P_1' correspondiente a la rama que iniciamos con X

2.- Insertar X en un kd-árbol

3.- Desde i=1 hasta n, ejecutar:

a.- Si la coordenada i-ésima puede crecer:

a1.- Hacer $x_i = x_i + 1$; llamar X al nuevo punto y volver a 2

a2.- $e_i = 0$

b.- Si la coordenada i-ésima puede decrecer:

b1.- Hacer $x_i = x_i - 1$; llamar X al nuevo punto y volver a 2

b2.- $e_i = 0$

4.- Fin

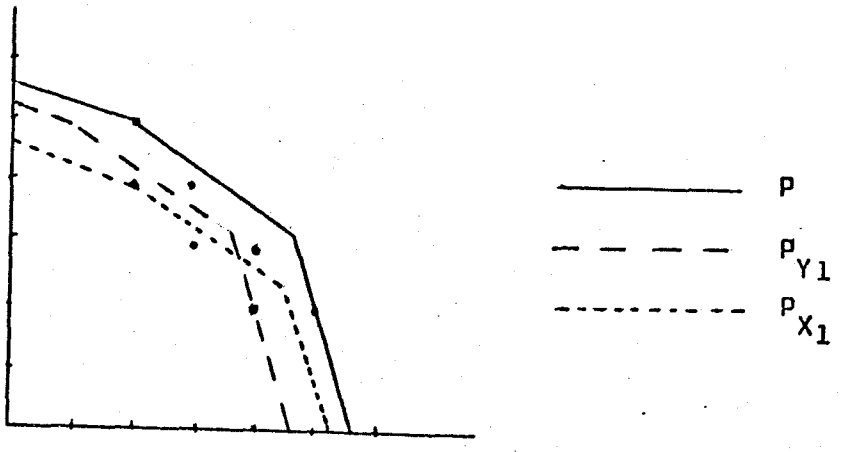
El algoritmo es recursivo y de ahí la necesidad de llamar siempre X al punto actual; observemos que una coordenada puede crecer, si su estado es de crecimiento y, al hacerlo, el nuevo punto es factible; por otra parte, una coordenada puede decrecer, si su estado es de decrecimiento, y el punto pertenece a $P \cap P_I'^{c}$ (o lo que es igual, no es factible para P_I') con alguna coordenada en estado de crecimiento, ya que en otro caso solo conduciría a puntos dominados.

Si el poliedro es muy amplio, podemos restringir las coordenadas de los puntos a unos intervalos determinados $[c, C]$ con c como cota inferior y C como superior, y hacer que el vector de estado valga cero cuando una coordenada sobrepase alguna de las cotas establecidas.

Además, en poliedros donde la búsqueda exhaustiva de los vectores eficientes pudiera hacerse demasiado costosa o imposible, pueden elegirse aleatoriamente diversas direcciones que nos proporcionen distintos vectores iniciales y buscar los vectores eficientes en un entorno de cada uno de ellos.

Ejemplo 1.- Consideremos el poliedro:

$$P:\ \begin{matrix} x + 3y \leqslant 17 \\ 2x + 3y \leqslant 19 \\ 3x + y \leqslant 17 \end{matrix} \qquad A_1 = \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad A_2 = \begin{bmatrix}3\\3\\1\end{bmatrix} \qquad B = \begin{bmatrix}17\\19\\17\end{bmatrix} \qquad D = B - AX$$



El poliedro es denso entero por ser $a_{ij} \geqslant 0 \ \forall i,j$ y partiremos del punto inicial X(4,3), que es un punto eficiente obtenido por redondeo del óptimo real en la dirección z=x+y para el problema de maximización. El punto X(4,3) sería el primer punto eficiente, y origina las ramas

$$X_1(\bar{3},\underline{3}) \text{ e } Y_1(\underline{4},\bar{2})$$

Las cotas para las variables son $0 \leqslant x \leqslant 5, 0 \leqslant y \leqslant 5$

El poliedro P_1' correspondiente a las ramas generadas por X_1 e Y_1, lo indicaremos por P_{X_1} y P_{Y_1} respectivamente.

$$P_{X_1}:\ \begin{matrix} x + 3y \leqslant 14 \\ 2x + 3y \leqslant 16 \\ 3x + y \leqslant 16 \end{matrix} \qquad P_{Y_1}\ \begin{matrix} x + 3y \leqslant 16 \\ 2x + 3y \leqslant 17 \\ 3x + y \leqslant 14 \end{matrix}$$

Rama generada por $X_1(\bar{3},\underline{3})$

$X_1(\bar{3},\underline{3})$ $\quad D = \begin{bmatrix}5\\4\\5\end{bmatrix}$ $\quad X_1 \in P_{X_1}$

∇x no procede por estar en P

Δy si ya que $D - A_2 \geqslant 0$: $X_2(3,4)$

$X_2(\bar{3},\underline{4})$ $\quad D=\begin{bmatrix}2\\1\\4\end{bmatrix}$ $\quad X_2 \notin P_{X_1}$ $\quad \nabla x$ si: $X_3(\bar{2},\underline{4})$ $\quad \Delta y$ no procede

$X_3(\bar{2},\underline{4})$ $\quad D=\begin{bmatrix}3\\3\\7\end{bmatrix}$ $\quad X_3 \in P_{X_1}$ $\quad \nabla x$ no procede $\quad \Delta y$ si: $X_4(\bar{2},\underline{5})$

Como en $X_4(\bar{2},\underline{5})$ la y ha llegado a su cota superior y la x solo decrece, esta rama solo puede conducir a puntos dominados, así que hemos concluido con ella.

Rama generada por $Y_1(\underline{4},\bar{2})$

$Y_1(\underline{4},\bar{2})$ $\quad D=\begin{bmatrix}7\\5\\3\end{bmatrix}$ $\quad Y_1 \in P_{Y_1}$ $\quad \Delta x$ si: $Y_2(\underline{5},\bar{2})$ $\quad \nabla y$ no procede

Como antes, en Y_2 la x ha llegado a su cota superior y la y decrece, por lo que hemos terminado con esta rama.

3.4.2.- Experiencias computacionales

El programa lo hemos titulado multobj1; la constante n1 representa el número máximo de variables y m1 el de restricciones; el tipo vect1 servirá para almacenar vectores de componentes enteras y el vect2, reales. El tipo a1 es para la matriz de los coeficientes y el c1 para los términos independientes.

N representará el número de variables y m el de restricciones; nv indicará el número de vectores visitados y ne el de eficientes.

A es la matriz de los coeficientes y d los términos independientes; b2 es d-ax y b3 es variable para cada rama, y

resulta de sumar a b2 el máximo coeficiente entre las variables que van creciendo en dicha rama.

Ci y cs almacenarán las cotas inferiores y superiores correspondientes a cada variable; e será el vector de estado.

La función fac vale 0 si el punto es infactible, 1 si es factible pero no pertenece a P y 2 si es factible para P .

Las funciones cr y de nos dicen cuando el vector puede aumentar o disminuir una de sus componentes.

La rutina avant es el núcleo del programa y la que desarrolla el algoritmo dado en el apartado anterior, con la particularidad de que en la componente 0 de cada vector, guarda el resultado de la función fac.

La rutina top es la encargada de conducir el punto de partida, resultado de redondear por defecto algún eficiente real, o cualquier factible, hasta un punto eficiente entero.

El programa principal lee los datos y ejecuta para cada una de las ramas el procedimiento avant; la variable l indica en cada momento la rama a sondear.

Lo hemos aplicado a los poliedros siguientes:

a.-

$$\begin{array}{rrrl} 9x_1 + & 21x_2 + & 2x_3 & \leq 180 \\ 17x_1 + & 8x_2 + & 15x_3 & \leq 290 \\ 35x_1 + & 7x_2 + & 14x_3 & \leq 256 \\ 4x_1 + & 19x_2 + & 23x_3 & \leq 304 \end{array}$$

$$x_i \geq 0 \qquad x_1 \leq 7 \qquad x_2 \leq 8 \qquad x_3 \leq 13$$

Se obtienen 28 puntos eficientes, para los que fue necesario visitar 155 puntos de un total de 440 puntos factibles. Los puntos eficientes son

4	4	6	4	2	7	2	7	7	1	0	13
4	6	5	4	0	8	3	7	3	3	1	10
6	6	0	7	1	0	0	8	6	3	3	9
6	4	1	6	2	2	1	8	1	3	5	8
5	6	2	5	1	5	1	6	8	2	3	10
5	5	3	6	0	3	2	2	11	0	5	9
5	3	4	3	6	7	2	1	12	2	4	9

b.-

$$
\begin{aligned}
9x_1 + 21x_2 + 2x_3 + 15x_4 + 6x_5 &\leq 240\\
18x_1 + 8x_2 + 16x_3 + 13x_4 + 24x_5 &\leq 303\\
35x_1 + 21x_2 + 44x_3 + 20x_4 + 7x_5 &\leq 210\\
60x_1 + 20x_2 + 7x_3 + 30x_5 &\leq 365\\
28x_1 + 11x_2 + 11x_3 + 9x_4 &\leq 57\\
13x_1 + 11x_2 + 39x_3 + x_4 + 9x_5 &\leq 355\\
x_1 + x_2 + x_3 + x_4 + x_5 &\leq 15
\end{aligned}
$$

$$x_i \geq 0 \qquad x_1 \leq 2 \qquad x_2 \leq 5 \qquad x_3 \leq 4 \qquad x_4 \leq 6 \qquad x_5 \leq 12$$

El número de puntos eficientes obtenidos es de 40, de un total de 485 puntos visitados y 732 puntos factibles.

c.-

$$
\begin{aligned}
9x_1 + 21x_2 + 2x_3 &\leq 436\\
17x_1 + 8x_2 + 15x_3 &\leq 610\\
35x_1 + 7x_2 + 14x_3 &\leq 704\\
4x_1 + 19x_2 + 23x_3 &\leq 672
\end{aligned}
$$

$$x_i \geq 0 \qquad x_1 \leq 15 \qquad x_2 \leq 16 \qquad x_3 \leq 21$$

Se obtienen 66 puntos eficientes, para los que fue necesario visitar 417 puntos de un total de 4917 puntos factibles.

d.- El mismo caso que el b, poniendo como términos independientes 506 698 845 950 352 720 y 40, y las cotas 7 10 9 11 17 respectivamente, se obtienen 1490 puntos eficientes y 11941 puntos visitados; el total de puntos factibles es de 156665.

e.- El mismo procedimiento aplicado a resolver la ecuación diofántica:

$$18x_1 + 13x_2 + 15x_3 + 12x_4 + 16x_5 = 174$$

con $x_i \geq 0$ $x_1 \leq 6$ $x_2 \leq 8$ $x_3 \leq 7$ $x_4 \leq 5$ $x_5 \leq 5$

buscando los puntos eficientes de la inecuación:

$$18x_1 + 13x_2 + 15x_3 + 12x_4 + 16x_5 \leq 174$$

y guardando unicamente aquellos para los que se verifica la igualdad, se obtienen 75 soluciones y 1911 puntos visitados.

f.- Análogamente para la ecuación:

$$25x_1 + 36x_2 + 12x_3 + 9x_4 + 34x_5 + 23x_6 + 13x_7 + 11x_8 + 15x_9 + 17x_{10} = 792$$

se obtienen 5868 soluciones y 387128 puntos visitados.

Programa multobj1

```
Program multobj1(input,output);
const n1=15;m1=12;
type vect1=array[0..n1] of integer;
     vect2=array[1..n1] of real;
     a1=array[1..m1,1..n1] of real;
     c1=array[1..m1] of real;
     pnodo=^nodo;
     nodo=record x:vect1;
          iz,de:pnodo
     end;
var raiz:pnodo;
    i,j,l,m,n,k,k1,nv,ne:integer;
    a:a1;
    b2,b3,d:c1;
    b,x,y,e,ci,cs:vect1;
    t:boolean;

function fac(d:c1):integer;
var i,l,J:integer;
begin l:=2;J:=2;
  for i:=1 to m do
  begin if(d[i]<0) then l:=0 else
          if((d[i]-b3[i])<0) then l:=1;
    if (l<J) then J:=l
  end;
  fac:=J
end;

function cr(J:integer;y:vect1;d:c1):integer;
var i,l:integer;
begin l:=1;i:=0;
  while ((l=1) and (i<m)) do
  begin i:=i+1;if(d[i]<a[i,J]) then l:=0
  end;
  cr:=l
end;

function de(J:integer;y:vect1;d:c1):integer;
var i,l:integer;
begin l:=1;i:=0;
  while ((l=1) and (i<m)) do
  begin i:=i+1;if(d[i]<-a[i,J]) then l:=0
  end;
  de:=l
end;

function dom(v,w:vect1):boolean;
var t:boolean;
begin t:=true;i:=0;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<w[i]) then t:=false
  end;
  dom:=t
end;
```

```
procedure imprimir(p:pnodo);
var i:integer;
begin
  if (p<>nil) then
  begin
    imprimir(p^.de);
    for i:=1 to n do write(p^.x[i]:3);ne:=ne+1;
    write('   ');if ((ne mod (20 div n))=0) then writeln;
    imprimir(p^.iz)
  end;
end;

function test1(b:vect1;p:pnodo;k:integer):boolean;
var l:integer;
    t:boolean;
begin t:=true;l:=(k mod n)+1;
  if (p<>nil) then if dom(p^.x,b) then t:=false else
  begin t:=test1(b,p^.de,l);
    if (t and (b[k]<=p^.x[k])) then t:=test1(b,p^.iz,l);
  end;
  test1:=t;
end;

procedure insertar2(b:vect1;var p:pnodo;k:integer);
var l:integer;
begin l:=(k mod n)+1;
  if (p=nil) then
  begin new(p);p^.x:=b;p^.iz:=nil;p^.de:=nil
  end else
  begin if b[k]<=p^.x[k] then insertar2(b,p^.iz,l)
    else insertar2(b,p^.de,l);
  end;
end;

procedure borrar(b:vect1;var p:pnodo;k:integer);
var raiz2:pnodo;

procedure recorrer(p:pnodo);
begin if (p<>nil) then
  begin recorrer(p^.de);
    recorrer(p^.iz);
    if not(dom(b,p^.x)) then insertar2(p^.x,raiz2,k);
    dispose(p);
  end;
end;

begin raiz2:=nil;
  recorrer(p);
  p:=raiz2;
end;

procedure test2(b:vect1;var p:pnodo;k:integer);
var l,j:integer;
begin l:=(k mod n)+1;
  if (p<>nil) then if dom(b,p^.x) then borrar(b,p,k) else
  begin test2(b,p^.iz,l);
    if b[k]>p^.x[k] then test2(b,p^.de,l);
  end;
end;
```

```
procedure insertar(b:vect1;var p:pnodo;k:integer);
var j,l:integer;
begin l:=(k mod n)+1;
  if (p=nil) then
  begin new(p);p^.x:=b;p^.iz:=nil;p^.de:=nil
  end else if not(dom(p^.x,b)) then if dom(b,p^.x) then
  begin borrar(b,p,k);insertar2(b,p,k)
  end else
  begin if b[k]<=p^.x[k] then
    begin if test1(b,p^.de,l) then insertar(b,p^.iz,l)
    end else
    begin test2(b,p^.iz,l);insertar(b,p^.de,l)
    end;
  end;
end;

procedure avant(y,e:vect1;d:c1);
var i,j,l:integer;
    d1:c1;
    z:vect1;
begin nv:=nv+1;
  k:=1;insertar(y,raiz,k);
  for i:=1 to n do if((e[i]=1) and (cr(i,y,d)=1)) then
  begin z:=y;z[i]:=z[i]+1;
    if(z[i]>=cs[i]) then e[i]:=0;
    for j:=1 to m do d1[j]:=d[j]-a[j,i];
    z[0]:=fac(d1);
    avant(z,e,d1);e[i]:=0
  end else
  begin if (y[0]=1) then
    begin  j:=0;for l:=1 to n do if (e[l]>0) then j:=j+1;
      k:=1;
      if (j>0) then
      if ((e[i]=-1) and (de(i,y,d)=1)) then
      begin z:=y;z[i]:=z[i]-1;
        if(z[i]<=ci[i]) then e[i]:=0;
        for j:=1 to m do d1[j]:=d[j]+a[j,i];
        z[0]:=fac(d1);
        avant(z,e,d1);e[i]:=0
      end;
    end;
  end;
end;

procedure top(var x:vect1);
var i,j:integer;
    z:vect1;
begin j:=0;z:=x;
  while (j=0) do
  begin j:=1;
    for i:=1 to m do
    begin d[i]:=b2[i];for j:=1 to n do d[i]:=d[i]-a[i,j]*z[j]
    end;
    for i:=1 to n do
    begin if ((cr(i,z,d)=1) and (z[i]<cs[i])) then
      begin z[i]:=z[i]+1;
        for j:=1 to m do d[j]:=d[j]-a[j,i];
        j:=0;x:=z;
      end;
    end;
```

```
  end;
end;

begin
  raiz:=nil;nv:=0;ne:=0;
  read(n,m);
  for i:=1 to m do for j:=1 to n do read(a[i,j]);
  for i:=1 to m do read(b2[i]);
  for i:=1 to n do read(ci[i]);
  for i:=1 to n do read(cs[i]);
  for i:=1 to n do read(x[i]);tor(x);
  for l:=2**n-1 downto 0 do
  begin k1:=l;for i:=1 to n do
    begin j:=k1 mod 2;
      if (j=0) then e[i]:=-1 else e[i]:=1;
      k1:=k1 div 2
    end;
    t:=true;
    for i:=1 to n do
    begin if ((x[i]<=ci[i]) and (e[i]=-1)) then t:=false;
      if ((x[i]>=cs[i]) and (e[i]=1)) then e[i]:=0
    end;
    if t then
    begin
      for i:=1 to n do if (e[i]>=0) then y[i]:=x[i] else
      begin y[i]:=x[i]-1;if(y[i]<=ci[i]) then e[i]:=0
      end;
      for i:=1 to m do
      begin d[i]:=b2[i];for j:=1 to n do d[i]:=d[i]-a[i,j]*y[j]
      end;
      for i:=1 to m do
      begin b3[i]:=0;
        for j:=1 to n do if((e[j]=1) and (b3[i]<a[i,j])) then
        b3[i]:=a[i,j]
      end;
      y[0]:=fac(d);
      if (y[0]>0) then avant(y,e,d) else nv:=nv+1
    end;
  end;
  writeln(nv);
  imprimir(raiz);
  writeln(ne)
end.
```

3.5.- Relajaciones en un poliedro

Dado el poliedro P

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i=1,2,\ldots m$$
$$x_j \geq 0 \quad j=1,2,\ldots,n$$

vamos a estudiar diversos tipos de relajaciones que usaremos en la búsqueda de sus puntos eficientes, cuando el poliedro no sea denso entero.

a.- El poliedro P^1 definido por:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta_i \quad \text{con} \quad \Delta_i = 1/2 \sum_{j=1}^{n} |a_{ij}| \quad i=1,2,\ldots,m$$
$$x_j \geq 0 \quad j=1,2,\ldots,n$$

contiene los redondeos de los puntos de P, es decir, dado cualquier punto factible $X(x_1, x_2, \ldots, x_n) \in P$ el punto $[X]([x_1], [x_2], \ldots, [x_n])$ tal que $[x_j]$ es el redondeo a entero de x_i, pertenece al poliedro P^1; además, P^1 puede contener otros puntos.

En efecto:

$$x'_i = x_i + \delta_i \qquad -1/2 < \delta_i \leq 1/2$$

$$\sum_{j=1}^{n} a_{ij} x'_j = \sum_{j=1}^{n} a_{ij} (x_j + \delta_j) = \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a_{ij} \delta_j \leq$$

$$b_i + 1/2 \sum |a_{ij}| = b_i + \Delta_i$$

b.- El poliedro P_1 definido por:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i - \Delta_i \quad i=1,2,\ldots,m \text{ con } \Delta_i = 1/2 \sum_{J=1} |a_{ij}|$$

$$x_j \geq 0 \quad j=1,2,\ldots,n$$

verifica la condición de que todos los puntos, resultantes de redondear algún punto de P_1 pertenencen a P.

En efecto, si $X(x_1,x_2,\ldots,x_n) \in P$ y sea $X'([x_1],[x_2],\ldots,[x_n])$ con $[x_i]$ igual al redondeo de x_i; X' pertenece a P ya, que:

$$x'_i = x_i + \delta_i \quad -1/2 < \delta_i \leq 1/2$$

$$\sum_{j=1}^{n} a_{ij} x'_j = \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a_{ij}\delta_j \leq b_i - \Delta_i + \sum_{j=1}^{n} a_{ij}\delta_j \leq$$

$$b_i - 1/2 \sum_{j=1}^{n} |a_{ij}| + \sum_{j=1}^{n} |a_{ij}|\,|\delta_j| \leq b_i - 1/2 \sum_{j=1}^{n} |a_{ij}| + 1/2 \sum_{j=1}^{n} |a_{ij}| = b_i$$

luego $X' \in P$

Tenemos así, que

$$P_1 \subset P \subset P'$$

La región $P'-P_1$ puede hacerse más reducida si incluimos cotas para cada una de las coordenadas.

c.- El poliedro P definido por:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta_i \quad \text{con} \quad \Delta_i = 1/2 \left(\sum_{j=1}^{n} a_{ij}^2\right)^{1/2} n^{1/2} \quad i=1,2,\ldots,m$$
$$x_j \geq 0 \quad j=1,2,\ldots,n$$

también contiene los redondeos de los puntos de P.

En efecto, sea $X(x_1,x_2,\ldots,x_n)$ y $X'([x_1],[x_2],\ldots,[x_n])$ como antes:

$$\sum_{j=1}^{n} a_{ij} x'_j - b_i \leq \sum_{j=1}^{n} a_{ij} x'_j - \sum_{j=1}^{n} a_{ij} x_j = \sum_{j=1}^{n} a_{ij}(x'_j - x_j) \leq$$

$$\|a_i\| \cdot \|X' - X\| = \left(\sum_{j=1}^{n} a_{ij}^2\right)^{1/2} n^{1/2}/2 = \Delta_i$$

luego $\sum_{j=1}^{n} a_{ij}\, x_j \leq b_i + \Delta_i \quad i=1,2,\ldots,m$

Hemos usado que $a_i=(a_1, a_2, \ldots, a_n)$ y que:

$$\| X'-X\| = \sqrt{\sum | x'_j - x_j |^2} \leq \sqrt{\sum (1/2)^2} = 1/2\sqrt{n}$$

d.- Análogamente los redondeos del poliedro P_2

$$P_2: \quad \sum_{j=1}^{n} a_{ij}\, x_j \leq b_i - \Delta_i \quad \Delta_i = 1/2 \left(\sum_{j=1}^{n} a_{ij}^2\right)^{1/2} n^{1/2}$$

pertenecen a P, con lo que se tiene también que:

$$P_2 \subset P \subset P^2$$

e.- Sea el poliedro P^+ definido por:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta_i \quad \text{con } \Delta_i = \sum_{j \in J^+} a_{ij} \quad i=1,2,\ldots,m$$

es decir, Δ_i es la suma de todos los coeficientes positivos de la inecuación i-ésima.

En P^+ están todos los redondeos por exceso de puntos de P; sea $X(x_1, x_2, \ldots, x_n) \in P$ y $X'(\lceil x_1 \rceil, \lceil x_2 \rceil, \ldots, \lceil x_n \rceil)$

$$x'_i = x_i + \delta_i \qquad 0 \leq \delta_i < 1$$

$$\sum_{j=1}^{n} a_{ij} x'_j = \sum_{j=1}^{n} a_{ij} (x_j + \delta_j) = \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a_{ij} \delta_j \leq$$

$$b_i + \sum_{j=1}^{n} a_{ij} \delta_j \leq b_i + \sum_{j \in J^+} a_{ij} = b_i + \Delta_i$$

evidentemente $P \subset P^+$

f.- De la misma forma se define el poliedro P^- por:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta_i \quad \text{con } \Delta_i = -\sum_{j \in J^-} a_{ij} \quad i=1,2,\ldots,m$$

es decir, es la suma de todos los coeficientes negativos, en valor absoluto, de la inecuación i-ésima.

Este poliedro contiene todos los redondeos por exceso de puntos de P; sea $X(x_1, x_2, \ldots, x_n)$ P y $X'(\lfloor x_1 \rfloor, \lfloor x_2 \rfloor, \ldots, \lfloor x_n \rfloor)$

$$x'_i = x_i - \delta_i \qquad 0 \leq \delta_i < 1$$

$$\sum_{j=1}^{n} a_{ij} x'_j = \sum_{j=1}^{n} a_{ij} (x_j - \delta_j) = \sum_{j=1}^{n} a_{ij} x_j - \sum_{j=1}^{n} a_{ij} \delta_j \leq$$

$$b_i - \sum_{j=1}^{n} a_{ij} \delta_j \leq b_i - \sum_{j \in J^-} a_{ij} = b_i + \Delta_i$$

también se verifica que $P \subset P^-$

Esta última relajación es la que nos seŕa de mayor utilidad por ser la que deforma menos la zona eficiente del poliedro original; obsérvese que aplicada a un poliedro denso entero, con $a_{ij} \geq 0$ no produce ninguna deformación; además, esta relajación, que nos sirve para localizar los puntos eficientes Pareto, podría definirse para la búsqueda de eficientes en una dirección cualquiera, sin más que poner:

$$\Delta_i = \sum |a_{ij}|$$

sumando todos los que tengan signo invertido respecto de la dirección a estudiar.

3.6.- Puntos eficientes enteros en un poliedro cualquiera

Cuando el poliedro P es denso entero, hemos demostrado que partiendo de cualquier punto con coordenadas enteras, podemos llegar a cualquier punto eficiente entero; sin embargo, esto no es válido en cualquier tipo de poliedros.

Lo que haremos para resolver el problema general consistirá en relajar el poliedro P a otro P_r, de forma que dos puntos eficientes enteros de P, puedan unirse por un camino uno a uno direccional contenido en P_r.

Las relajaciones estudiadas en el apartado anterior y que verificarán esta propiedad son:

a.- El poliedro P^1 definido por:

$$\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i + \Delta_i \quad \text{con} \quad \Delta_i = 1/2 \sum_{j=1}^{n} |a_{ij}| \quad i=1,2,\ldots,m$$
$$x_j \geqslant 0 \quad j=1,2,\ldots,n$$

hemos demostrado que contiene los redondeos de los puntos de P (y otros puntos, posiblemente).

Si $X(x_1,x_2,\ldots,x_n) \in P$ el punto $X'([x_1],[x_2],\ldots,[x_n])$ tal que $[x_i]$ es el redondeo a entero de x_i, pertenece al poliedro P ya que según vimos:

$$x'_i = x_i + \delta_i \qquad -1/2 < \delta_i \leqslant 1/2$$

$$\sum_{j=1}^{n} a_{ij} x'_j = \sum_{j=1}^{n} a_{ij} (x_j + \delta_j) = \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a_{ij} \delta_j \leqslant$$

$$b_i + 1/2 \sum |a_{ij}| = b_i + \Delta_i$$

Observemos que P_1 no solo contiene a los puntos X, sino a

cualquier punto $Z(z_1, z_2, \ldots, z_n)$ que verifique:

$$Z = \lfloor X + 1/2 \sum_{i=1}^{n} e_i \rfloor$$

ó $$Z' = \lceil X - 1/2 \sum_{i=1}^{n} e_i \rceil$$

El punto Z coincide con X' y por lo tanto, pertenece a P' como hemos demostrado, mientras que el punto Z' verifica que:

$$z'_i = \lceil x_i - 1/2 \rceil = x_i - 1/2 + \delta_i \quad 0 \leqslant \delta_i < 1$$
$$z'_i = x_i + \delta'_i - 1/2 \leqslant \delta'_i < 1/2$$

$$\sum_{j=1}^{n} a_{ij} z'_j = \sum_{j=1}^{n} a_{ij} (x_j + \delta'_j) = \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a_{ij} \delta'_j \leqslant b_i + 1/2 \sum_{j=1}^{n} |a_{ij}| = b_i + \Delta_i$$

Teorema 1.- Si X,Y son dos puntos eficientes enteros de P, existe un camino uno a uno direccional que une X con Y, contenido en P'.

Demostración: Por ser P convexo, el segmento que une X con Y está contenido en P, es decir:

$$Z = \lambda X + (1-\lambda) Y \qquad 0 \leq \lambda \leq 1$$

$$Z \in P \qquad \forall \lambda \in [0,1]$$

El redondeo de cualquier punto Z de este segmento pertenece a P'; este redondeo determina una sucesión de puntos en los que cada uno difiere del anterior en una unidad salvo que el punto Z alcance el valor $k+1/2$ con $k \in N$ en más de una coordenada; pero si esto ocurre, por ejemplo, en las coordenadas z_1 y z_2, ya hemos demostrado que los puntos:

$$Z_1(\lceil z_1\rceil , \lceil z_2\rceil , \lceil z_3\rceil , \ldots , \lceil z_n\rceil)$$

$$Z_2(\lceil z_1\rceil , \lfloor z_2\rfloor , \lceil z_3\rceil , \ldots , \lceil z_n\rceil)$$

$$Z_3(\lfloor z_1\rfloor , \lceil z_2\rceil , \lceil z_3\rceil , \ldots , \lceil z_n\rceil)$$

$$Z_4(\lfloor z_1\rfloor , \lfloor z_2\rfloor , \lceil z_3\rceil , \ldots , \lceil z_n\rceil)$$

pertenecen a P^2, por lo que para pasar de Z_1 a Z_4 lo podemos hacer indistintamente por Z_2 o por Z_3.

Realmente, demostramos que Z_1 y Z_4 pertenecían a P^2 pero no Z_2 y Z_3; para verlo, sea:

$$Z \in P \qquad Z=\sum_{i=1}^{n} z_i e_i$$

e I,J una partición del conjunto $\{1,2,\ldots,n\}$

el punto:

$$Z' =\sum_{i\in I} \lfloor z_i +1/2\rfloor e_i +\sum_{i\in J} \lceil z_i -1/2\rceil e_i =$$

$$\sum_{i\in I} (z_i+1/2-\delta_i)e_i+\sum_{i\in J}(z_i-1/2+\delta_i)e_i = \sum_{i\in I}(z_i+\delta'_i)e_i+\sum_{i\in J}(z_i+\delta'_i)e_i$$

$$0 \leqslant \delta_i < 1 \quad \forall i$$

$$-1/2 < \delta'_i \leqslant 1/2 \quad i \in I$$

$$-1/2 \leqslant \delta'_i < 1/2 \quad i \in J$$

$$\sum_{j=1}^{n} a_{ij}\, z'_i = \sum_{j\in I} a_{ij}(z_j+\delta'_j)+\sum_{j\in J} a_{ij}(z_j+\delta'_j) =$$

$$\sum_{j=1}^{n} a_{ij}z_j+\sum_{j\in I} a_{ij}\delta'_j+\sum_{j\in J} a_{ij}\delta'_j \leqslant b_i+1/2\sum_{j\in I}|a_{ij}|+1/2\sum_{j\in J}|a_{ij}| =$$

$$b_i+1/2\sum_{j=1}^{n}|a_{ij}| = b_i+\Delta_i$$

con lo que queda demostrado el teorema.

b.- Análogamente puede demostrarse para la relajación:

$$P^2 \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta_i$$

$$\Delta_i = 1/2 \left(\sum_{j=1}^{n} a_{ij}^2 \right)^{1/2} n^{1/2}$$

que verificaba que el redondeo de cualquier punto de P pertenece a P^2.

c.- Consideremos la relajación:

$$P^- \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i + \Delta_i$$

$$\Delta_i = -\sum_{j \in J} a_{ij}$$

Si $X \in P$ vimos que el punto $X' = (\lfloor x_1 \rfloor , \lfloor x_2 \rfloor , \ldots , \lfloor x_n \rfloor)$ pertenecía a P^-.

También en este caso, dados dos puntos enteros X,Y de P, los puntos del segmento que une X con Y:

$$Z = \lambda X + (1-\lambda) Y \quad \lambda \in [0,1]$$

están en P; el conjunto de puntos definidos por Z , formarán el camino buscado, si ningún punto del segmento tiene dos coordenadas iguales $z_i = z_j$ y enteras, siendo además $x_i > y_i$ y $x_j > y_j$.

Supongamos que para un cierto punto Z, se verificase que varias coordenadas de las que son distintas en X y en Y, tomasen valores enteros y van decreciendo.

Sean z_1 y z_2 tales coordenadas en $Z(z_1, z_2, \ldots, z_n)$ y $Z' = (z_1, z_2, \lfloor z_3 \rfloor , \ldots , \lfloor z_n \rfloor)$; veamos que los puntos:

$$Z_1 (z_1, z_2, \lfloor z_3 \rfloor , \ldots , \lfloor z_n \rfloor)$$

$$Z_2 (z_1 - 1, z_2 , \lfloor z_3 \rfloor , \ldots , \lfloor z_n \rfloor)$$

$$Z_3(z_1, z_2-1, \lfloor z_3 \rfloor, \ldots, \lfloor z_n \rfloor)$$

$$Z_4(z_1-1, z_2-1, \lfloor z_3 \rfloor, \ldots, \lfloor z_n \rfloor)$$

pertenecen a P^-.

$$Z_1 = Z' \in P^-$$

$$Z_2 = (z_1-1, z_2, z_3-\delta_3, \ldots, z_n-\delta_n); 0 \leq \delta_i < 1$$

$$a_{i1}(z_1-1)+a_{i2}z_2+a_{i3}(z_3-\delta_3)+\text{----}+a_{in}(z_n-\delta_n) =$$

$$\sum_{j=1}^{n} a_{ij}z_j - \sum_{j=1}^{n} a_{ij}\delta_j \leq b_i - \sum_{j\,J} a_{ij} = b_i + \Delta_i \quad (\delta_1=1, \delta_2=0)$$

análogamente, para $Z_3(\delta_1=0, \delta_2=1)$ y $Z_4(\delta_1=1, \delta_2=1)$, lo cual nos garantiza el camino uno a uno direccional que une X con Y, contenido en P^-.

d.- Muy similar demostración puede hacerse para el poliedro:

$$P^+: \sum_{j=1}^{n} a_{ij}x_j \leq b_i + \Delta_i$$

$$\Delta_i = \sum_{j \in J} a_{ij}$$

Cualquiera de las cuatro relajaciones nos serviría en nuestro propósito de alcanzar todos los puntos eficientes de un poliedro dado; no obstante, la que usaremos será la P^-, que es la que menos deforma la zona eficiente, como lo pone de manifiesto el hecho de que $P^- = P$ en los poliedros denso enteros con $a_{ij} \geq 0$.

Como P^- contiene puntos que no corresponden a redondeos de puntos de P, lo someteremos a cotas inferiores y superiores para cada variable.

3.7.- Localización de puntos eficientes

3.7.1.- Algoritmo de búsqueda

El desarrollo que hacemos a continuación está referido a la relajación P^- relativa al poliedro P, aunque puede hacerse para cualquier otra relajación entre las ya estudiadas; hemos insistido que P^- es la que menos deforma la zona eficiente Pareto del poliedro, lo cual no significa que en ciertos casos no pueda ser aplicada, incluso con mayor éxito otro tipo de relajación.

Hemos demostrado que toda pareja de puntos en P^- que sean redondeos por defecto de puntos de P, pueden ser unidos por un camino uno a uno direccional dentro del paralelepípedo que determinan y cuyos vértices están en P^-.

En el algoritmo que seguiremos, el punto inicial puede ser cualquiera obtenido de redondear por defecto un punto $Y(y_1, y_2, \ldots, y_n)$ que sea el óptimo real de P para una cierta dirección $z=\alpha_1 x_1+\alpha_2 x_2+\text{----}+\alpha_n x_n \quad \alpha_i \geq 0$ para todo i.

$$AY=y_1 A_1+y_2 A_2+\text{----}+y_n A_n \leq B$$

Sea $Z(z_1, z_2, \ldots, Z_n)$ tal que $z_i = \lfloor y_i \rfloor = y_i - \delta_i \quad 0 \leq \delta_i < 1$

$$AZ=z_1 A_1+z_2 A_2+\text{----}+z_n A_n=(y_1-\delta_1)A_1+(y_2-\delta_2)A_2+\text{----}+(y_n-\delta_n)A_n=$$

$$y_1 A_1+y_2 A_2+\text{----}+y_n A_n-\delta_1 A_1-\delta_2 A_2-\text{----}-\delta_n A_n \leq$$

$$B-A'_1-A'_2-\text{----}-A'_n$$

siendo $A'_j=[a'_{1j}, a'_{2j}, \ldots a'_{mj}]'$

y $a'_{ij}=a_{ij}$ si $a_{ij}\leqslant 0$ mientras que $a'_{ij}=0$ si $a_{ij}>0$.

Luego Z pertenece efectivamente a P^-; si $Z\notin P$ lo tomaremos como punto inicial, mientras que si $Z\in P$, aumentaremos sus coordenadas manteniendo la factibilidad, hasta que ninguna de ellas pueda ser incrementada; el punto final obtenido lo tomaremos como punto inicial del algoritmo de búsqueda.

En cualquiera de los casos, llamemos $X(x_1, x_2, \ldots, x_n)$ al punto inicial; este punto divide al espacio en 2^n cuadrantes y el punto de partida de cada cuadrante, será un vértice de la bola unidad con norma l_∞ cuyo vértice superior sea X, tal como hicimos en los poliedros denso enteros. Para el caso n=3 y el punto $X(x_1, x_2, x_3)$ se tienen las ramas, $X_1, X_2, \ldots, X_8$ ya expresadas en los poliedros denso enteros. Si $X\notin P$ no podemos abandonar directamente la rama i-ésima como ocurría entonces con las ramas 1 y 8.

Sea P'_i: $\sum_{j=1}^{n} a_{ij} x_j \leqslant b'_i$ $b'_i=b_i-\max(\max_{j\ I}(a_{ij}),0)$ siendo I el conjunto de coordenadas que van creciendo. Se verifica el siguiente teorema.

Teorema 1.- Sea X el punto inicial de una rama e Y un punto eficiente entero de P en dicha rama; ambos puntos pueden ser unidos por un camino uno a uno direccional interior a P^- y al paralelepípedo determinado por ellos, de forma que en el interior de P'_i solo haremos movimientos que hagan crecer alguna de sus coordenadas.

Demostración: Como hemos visto, existe un camino uno a uno direccional que une X con Y en el interior de P^- y en el

paralelepípedo que determinan. En este camino de X a Y, se Z al primer punto que pertenece a P_i' :

$$Z(\underline{z}_1, \underline{z}_2, \ldots, \underline{z}_j, \overline{z}_{j+1}, \ldots, \overline{z}_k, z_{k+1}, \ldots, z_n)$$

$$Y(y_1, y_2, \ldots, y_j, y_{j+1}, \ldots, y_k, y_{k+1}, \ldots, y_n)$$

Como Z es factible debe existir algún $z_i < y_i$ $i=1,\ldots,j$, ya que en otro caso dominaría a Y y éste no sería eficiente. Aumentaremos una de estas coordenadas hasta llegar a $P_i'^c \cap P$.

Al nuevo punto obtenido, como es redondeo de sí mismo, podemos aplicarle el mismo razonamiento; así sucesivamente, tendríamos el camino exigido por el teorema, dado que el número de pasos es finito.

c.q.d.

Damos a continuación un algoritmo recursivo, válido para cada rama con punto inicial X y cuyo vector de estado viene expresado por $E(e_1, e_2, \ldots, e_n)$ con $e_i=1$ si la coordenada x va creciendo, $e_i=-1$ si va decreciendo y $e_i=0$ si está fija.

1.- Creación de P_1' correspondiente a la rama con punto inicial X

2.- Si X está en P insertarlo en el kd-árbol

3.- Desde i=1 hasta n, ejecutar:

a.- Si la coordenada i-ésima puede crecer

a1.- Hacer $x_i = x_i + 1$; llamar X al nuevo punto y volver a 2

a2.- $e_i = 0$

b.- Si la coordenada i-ésima puede decrecer

b1.- Hacer $x_i = x_i - 1$; llamar X al nuevo punto y volver a 2

b2.- $e_i=0$

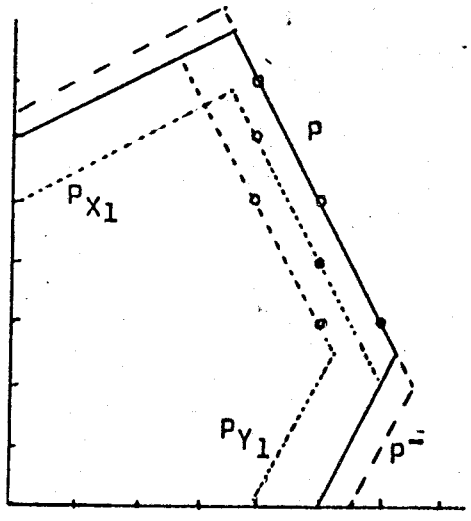
4.- Fin

Debe tenerse en cuenta que una coordenada puede crecer, si su estado es de crecimiento, y al hacerlo, el nuevo punto pertenece a $P^{=}$; por otra parte, una coordenada puede decrecer si su estado es de decrecimiento, el punto pertenece a $P^{-} \cap P_1^{c}$ y, o bien alguna coordenada se encuentra en estado de crecimiento o, si todas decrecen, el punto ni es factible ni es dominado por ningún punto existente en el kd-árbol.

Ejemplo 1.-

$$P: \quad \begin{array}{rcl} -x + 2y & \leqslant & 12 \\ 2x + y & \leqslant & 15 \\ 2x - y & \leqslant & 10 \end{array} \qquad A_1 = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix} \quad A_2 = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix} \quad B = \begin{bmatrix} 12 \\ 15 \\ 10 \end{bmatrix} \quad D=B-AX$$

$$P^{-}: \quad \begin{array}{rcl} -x + 2y & \leqslant & 13 \\ 2x + y & \leqslant & 15 \\ 2x - y & \leqslant & 11 \end{array} \qquad B^{-} = \begin{bmatrix} 12 \\ 15 \\ 10 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 13 \\ 15 \\ 11 \end{bmatrix}$$



Tomemos como punto inicial el correspondiente a $\lfloor \max(x) \rfloor$, incrementandolo hasta que sea eficiente; esto nos proporciona el

punto $X_0(6,3)$ que es factible para P y origina las ramas $X_1(\bar{5},\underline{3})$ y $Y_1(\underline{6},\bar{2})$.

$$P_{X_1}: \begin{array}{rcl} -x + 2y & \leqslant & 10 \\ 2x + y & \leqslant & 14 \\ 2x - y & \leqslant & 10 \end{array} \qquad B_{X_1} = \begin{bmatrix} 12 \\ 15 \\ 10 \end{bmatrix} - \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \\ 10 \end{bmatrix}$$

$$P_{Y_1}: \begin{array}{rcl} -x + 2y & \leqslant & 12 \\ 2x + y & \leqslant & 13 \\ 2x - y & \leqslant & 8 \end{array} \qquad B_{Y_1} = \begin{bmatrix} 12 \\ 15 \\ 10 \end{bmatrix} - \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 13 \\ 18 \end{bmatrix}$$

las cotas serán 6 y 7 para x e y respectivamente; la rama de Y_1 podemos abandonarla ya que x está en su cota superior y la y solo decrece.

Rama $X_1(\bar{5},\underline{3})$

$X_1(\bar{5},\underline{3})$ $\quad D = \begin{bmatrix} 11 \\ 2 \\ 3 \end{bmatrix}$ $\quad X \in P_{X_1}$ ∇x no procede; Δy si: $X_2(\bar{5},\underline{4})$

$X_2(\bar{5},\underline{4})$ $\quad D = \begin{bmatrix} 9 \\ 1 \\ 4 \end{bmatrix}$ $\quad X \in P_{X_1}$ ∇x no procede; Δy si: $X_3(\bar{5},\underline{5})$

$X_3(\bar{5},\underline{5})$ $\quad D = \begin{bmatrix} 7 \\ 0 \\ 5 \end{bmatrix}$ $\quad X \in P$ ∇x si: $X_4(\bar{4},\underline{5})$; Δy no procede

$X_4(\bar{4},\underline{5})$ $\quad D = \begin{bmatrix} 6 \\ 2 \\ 7 \end{bmatrix}$ $\quad X \in P_{X_1}$ ∇x no procede; Δy si: $X_5(\bar{4},\underline{6})$

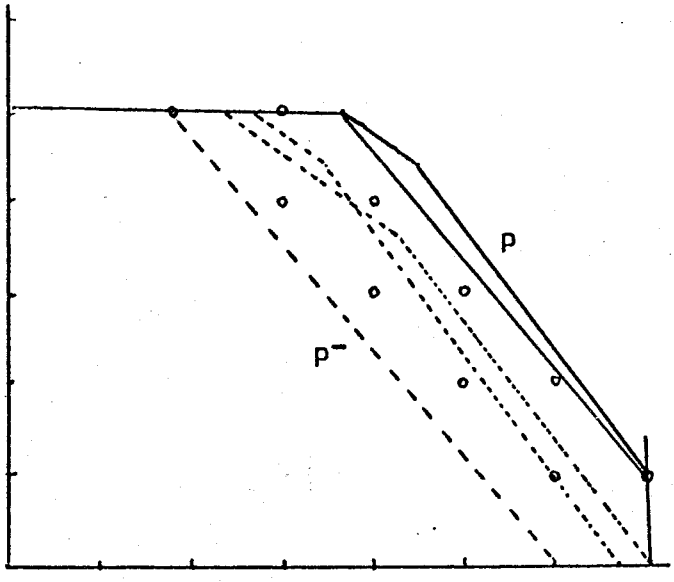
$X_5(\bar{4},\underline{6})$ $\quad D = \begin{bmatrix} 4 \\ 1 \\ 8 \end{bmatrix}$ $\quad X \in P_{X_1}$ ∇x no procede; Δy si: $X_6(\bar{4},\underline{7})$

La y ha llegado a su cota superior y la x decrece, por lo que hemos llegado al final de esta rama. Los puntos eficientes obtenidos han sido $X_0(6,3)$, $X_3(5,5)$ y $X_6(4,7)$.

Ejemplo 2.-

$$P: \quad \begin{array}{rcl} -6x - 5y & \leqslant & -47 \\ 4x + 3y & \leqslant & 31 \\ 3x + 4y & \leqslant & 31 \end{array} \qquad A_1 = \begin{bmatrix} -6 \\ 4 \\ 3 \end{bmatrix} \quad A_2 = \begin{bmatrix} -5 \\ 3 \\ 4 \end{bmatrix} \quad B = \begin{bmatrix} -47 \\ 31 \\ 31 \end{bmatrix}$$

$$P^{-}: \quad \begin{array}{rcl} -6x - 5y & \leqslant & -36 \\ 4x + 3y & \leqslant & 31 \\ 3x + 4y & \leqslant & 31 \end{array}$$



El punto de partida será el $X_o(3,5)$ que es el redondeo del óptimo respecto a z=y; las ramas originadas son $X_1(\bar{2},\underline{5})$, $Y_1(\underline{3},\bar{4})$ y $Z_1(\bar{2},\bar{4})$; las cotas que usaremos serán 7 y 5 para x e y respectivamente.

La rama correspondiente a $Z_1(\bar{2},\bar{4})$ no procede porque este punto no es factible para P^{-}.

Rama $X_1(\bar{2},\underline{5})$

$X_1(\bar{2},\underline{5})$ $\quad D = \begin{bmatrix} -10 \\ 8 \\ 5 \end{bmatrix}$ $\quad X \in P^{-}$ $\quad \nabla x$ no procede

Δy esta en su cota superior

Rama $Y_1(\underline{3},\bar{4})$

$Y_1(\underline{3},\bar{4})$ $\quad D = \begin{bmatrix} -9 \\ 7 \\ 6 \end{bmatrix}$ $\quad X \in P^{-}$ $\quad \Delta x$ si: $Y_2(\underline{4},\bar{4})$

∇y no procede

$Y_2(\underline{4},\bar{4})$ $\quad D=\begin{bmatrix}-3\\3\\3\end{bmatrix}$ $\quad X \in P^-$ x no procede y si: $Y_3(\underline{4},\bar{3})$

$Y_3(\underline{4},\bar{3})$ $\quad D=\begin{bmatrix}-8\\6\\7\end{bmatrix}$ $\quad X \in P^-$ x si: $Y_4(\underline{5},\bar{3})$ y no

$Y_4(\underline{5},\bar{3})$ $\quad D=\begin{bmatrix}-2\\2\\4\end{bmatrix}$ $\quad X \in P^-$ x no procede y si: $Y_5(\underline{5},\bar{2})$

$Y_5(\underline{5},\bar{2})$ $\quad D=\begin{bmatrix}-7\\5\\8\end{bmatrix}$ $\quad X \in P^-$ x si: $Y_6(\underline{6},\bar{2})$ y no procede

$Y_6(\underline{6},\bar{2})$ $\quad D=\begin{bmatrix}-1\\1\\5\end{bmatrix}$ $\quad X \in P^-$ x no procede y si: $Y_7(\underline{6},\bar{1})$

$Y_7(\underline{6},\bar{1})$ $\quad D=\begin{bmatrix}-6\\4\\9\end{bmatrix}$ $\quad X \in P^-$ x si: $Y_8(\underline{7},\bar{1})$ y no procede

la x ha llegado a su cota superior por lo que hemos terminado con esta rama y el único punto eficiente obtenido es el $Y_8(7,1)$.

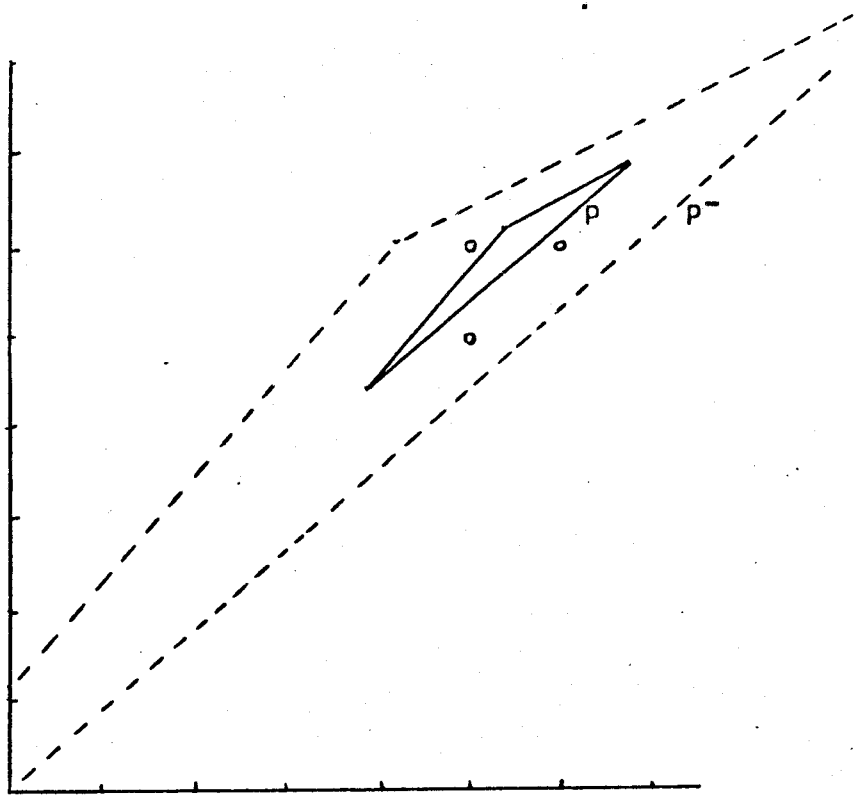
Ejemplo 3.-

$$P: \begin{array}{rcl} -8x + 7y & \leqslant & 0 \\ 7x - 8y & \leqslant & -8 \\ -x + 2y & \leqslant & 7 \end{array} \qquad A_1=\begin{bmatrix}-8\\7\\-1\end{bmatrix} \qquad A_2=\begin{bmatrix}7\\-8\\2\end{bmatrix} \qquad B=\begin{bmatrix}0\\-8\\7\end{bmatrix}$$

$$P^-: \begin{array}{rcl} -8x + 7y & \leqslant & 8 \\ 7x - 8y & \leqslant & 0 \\ -x + 2y & \leqslant & 8 \end{array}$$

En este caso las cotas son $5 \leqslant x \leqslant 6$ y $5 \leqslant y \leqslant 6$; el redondeo del óptimo en la dirección $z=x+y$ es el punto $X_0(6,6)$, que origina las ramas:

$X_1(\underline{5},\bar{6})$ $\qquad Y_1(\bar{6},\underline{5})$ $\qquad Z_1(\bar{5},\bar{5})$



ninguno de los cuales puede variar sus coordenadas en las direcciones indicadas, y ninguno es factible en P, luego no existen puntos eficientes enteros.

Ejemplo 4.-

$$P: \begin{array}{rcl} 6x - 5y & \leqslant & 1 \\ -3x + 4y & \leqslant & 7 \\ -4x + 3y & \leqslant & -1 \end{array} \qquad A_1 = \begin{bmatrix} 6 \\ -3 \\ -4 \end{bmatrix} \qquad A_2 = \begin{bmatrix} -5 \\ 4 \\ 3 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 7 \\ -1 \end{bmatrix}$$

$$P^-: \begin{array}{rcl} 6x - 5y & \leqslant & 6 \\ -3x + 8y & \leqslant & 10 \\ -4x + 3y & \leqslant & 3 \end{array}$$

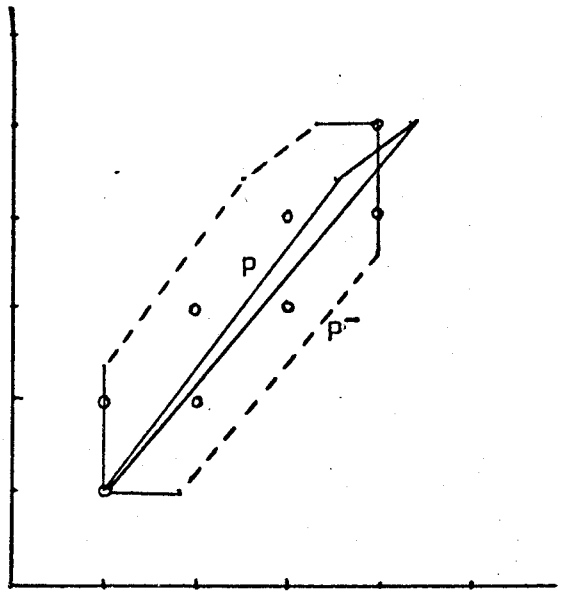
Las cotas serán $1 \leqslant x \leqslant 4 \quad 1 \leqslant y \leqslant 5$; partiremos del punto $X_0(4,5)$ que origina las ramas $X_1(\underline{4},\overline{4})$, $Y_1(\overline{3},\underline{5})$ y $Z_1(\overline{3},\overline{4})$.

Rama $X_1(\underline{4},\overline{4})$

$X_1(\underline{4},\overline{4})$ $\qquad D = \begin{bmatrix} -3 \\ 3 \\ 3 \end{bmatrix}$ $\qquad X \in P^-$ Δx no procede

∇y no procede

Rama $Y_1(\overline{3},\underline{5})$



$Y_1(\bar{3},\underline{5})$ $\quad D=\begin{bmatrix}8\\-4\\-4\end{bmatrix}$ $\quad X \notin P^-$ Fin de rama

Rama $Z_1(\bar{3},\bar{4})$

$Z_1(\bar{3},\bar{4})$ $\quad D=\begin{bmatrix}3\\0\\-1\end{bmatrix}$ $\quad X \in P^-$ ∇x no procede; ∇y si: $Z_2(\bar{3},\bar{3})$

las variables deberán decrecer hasta llegar a un factible o recorrer P^-.

$Z_2(\bar{3},\bar{3})$ $\quad D=\begin{bmatrix}-2\\4\\2\end{bmatrix}$ $\quad X \in P^-$ ∇x si: $Z_3(\bar{2},\bar{3})$; ∇y no procede

$Z_3(\bar{2},\bar{3})$ $\quad D=\begin{bmatrix}4\\1\\-2\end{bmatrix}$ $\quad X \in P^-$ ∇x no procede; ∇y si: $Z_4(\bar{2},\bar{2})$

$Z_4(\bar{2},\bar{2})$ $\quad D=\begin{bmatrix}-1\\5\\1\end{bmatrix}$ $\quad X \in P^-$ ∇x si: $Z_5(\bar{1},\bar{2})$; ∇y no procede

$Z_5(\bar{1},\bar{2})$ $\quad D=\begin{bmatrix}5\\2\\-3\end{bmatrix}$ $\quad X \in P^-$ ∇x no procede; ∇y si: $Z_6(\bar{1},\bar{1})$

Como $Z_6(1,1)$ es factible en P y las coordenadas van decreciendo, hemos terminado.

3.7.2.- Experiencias computacionales

El programa lo hemos titulado multobj2 y lo hemos aplicado a los poliedros siguientes:

a.-

$$
\begin{aligned}
9x_1 + 21x_2 - 2x_3 &\leq 436\\
18x_1 + 8x_2 + 16x_3 &\leq 597\\
35x_1 + 21x_2 - 44x_3 &\leq 1294\\
60x_1 - 20x_2 + 7x_3 &\leq 694\\
28x_1 + 11x_2 - 11x_3 &\leq 253\\
-13x_1 + 11x_2 + 39x_3 &\leq 615
\end{aligned}
$$

$$x_1 \leq 12 \qquad x_2 \leq 18 \qquad x_3 \leq 17$$

Los puntos eficientes obtenidos son:

12 9 17, 9 13 15, 9 14 14, 10 12 15, 8 15 14, 7 18 13 y 11 11 16

para lo cual se han visitado 34 puntos de un total de 2433 puntos factibles.

b.-

$$
\begin{aligned}
2x_1 + 3x_2 + x_3 + 2x_4 + 2x_5 &\leq 18\\
3x_1 + 2x_2 + 2x_3 + x_4 + 2x_5 &\leq 15\\
-6x_1 + 15x_3 &\leq 10\\
-7x_2 + 18x_4 &\leq 10
\end{aligned}
$$

$$x_1 \leq 5 \qquad x_2 \leq 6 \qquad x_3 \leq 7 \qquad x_4 \leq 9 \qquad x_5 \leq 7$$

El número de puntos eficientes obtenidos es de 50, de un total de 255 puntos visitados.

c.-

$$\begin{array}{rrrrrrrrrrr} -6x_1 & + & 23x_2 & + & 19x_3 & - & 15x_4 & + & 13x_5 & < & 123 \\ 8x_1 & + & 12x_2 & + & 9x_3 & + & 7x_4 & + & 8x_5 & < & 457 \\ 17x_1 & - & 15x_2 & - & 21x_3 & + & 18x_4 & + & 20x_5 & < & 113 \\ 9x_1 & - & 13x_2 & + & 14x_3 & + & 22x_4 & - & 9x_5 & < & 74 \\ 8x_1 & + & 7x_2 & + & 4x_3 & + & 12x_4 & + & 9x_5 & < & 398 \end{array}$$

$$x_1 < 16 \qquad x_2 < 12 \qquad x_3 < 15 \qquad x_4 < 11 \qquad x_5 < 7$$

El número de puntos eficientes obtenidos ha sido de 114, y se han visitado 7647 puntos.

e.-

$$\begin{array}{rrrrrrrrrrrrrrr} -2x_1 & + & 3x_2 & + & x_3 & + & 4x_4 & + & 2x_5 & - & x_6 & + & 2x_7 & + & x_8 & - & x_9 & + & 3x_{10} & \leqslant & 15 \\ 3x_1 & - & x_2 & + & 2x_3 & - & 2x_4 & + & x_5 & + & 2x_6 & - & x_7 & + & 3x_8 & + & 2x_9 & - & 2x_{10} & \leqslant & 12 \\ 3x_1 & + & 2x_2 & + & x_3 & + & 2x_4 & + & x_5 & + & 3x_6 & + & 2x_7 & + & 4x_8 & + & x_9 & + & 3x_{10} & \leqslant & 45 \\ 4x_1 & + & 2x_2 & - & x_3 & + & 3x_4 & - & 2x_5 & + & 3x_6 & + & x_7 & - & x_8 & + & 4x_9 & + & x_{10} & \leqslant & 13 \\ x_1 & - & 2x_2 & + & 4x_3 & - & x_4 & + & 4x_5 & - & 2x_6 & + & 3x_7 & - & x_8 & + & x_9 & - & x_{10} & \leqslant & 15 \\ 2x_1 & + & x_2 & + & 4x_3 & + & 3x_4 & + & 2x_5 & + & x_6 & + & 3x_7 & + & 4x_8 & + & 2x_9 & + & 2x_{10} & \leqslant & 37 \\ -x_1 & + & 3x_2 & + & 2x_3 & + & x_4 & + & 3x_5 & + & x_6 & - & 2x_7 & + & 2x_8 & + & x_9 & + & 3x_{10} & \leqslant & 12 \end{array}$$

cotas superiores: (4,4,4,3,4,5,7,5,4,5)

El número de puntos eficientes obtenido es de 1627, para lo cual se han tenido que visitar 113603 puntos.

f.- Este ejemplo es un poliedro no vacío que no contiene ningún punto de coordenadas enteras.

$$
\begin{array}{rcrcrcrcrcrcl}
2x_1 & + & x_2 & + & 2x_3 & & & & & & & & \leqslant 25 \\
 & - & 2x_2 & + & x_3 & + & x_4 & & & & & & \leqslant 0 \\
 & & & - & x_3 & + & x_4 & + & 2x_5 & & & & \leqslant 10 \\
 & & & & & - & x_4 & & & + & x_6 & & \leqslant -5 \\
 & & & & & & & - & x_5 & + & x_6 & + 2x_7 & \leqslant -1 \\
-x_1 & & & & & & & & & & x_6 & + 2x_7 & \leqslant -2 \\
 & & & & & & & & & & & & \\
-2x_1 & - & x_2 & - & 2x_3 & & & & & & & & \leqslant -25 \\
 & & 2x_2 & - & x_3 & - & x_4 & & & & & & \leqslant 0 \\
 & & & & x_3 & - & x_4 & - & 2x_5 & & & & \leqslant -10 \\
 & & & & & & x_4 & & & - & x_6 & & \leqslant 5 \\
 & & & & & & & & x_5 & - & x_6 & - 2x_7 & \leqslant 1 \\
-x_1 & & & & & & & & & - & x_6 & - 2x_7 & \leqslant 2
\end{array}
$$

cotas superiores: (5,6,4,7,4,2,1)

La comprobación de que no existen puntos enteros ha exigido visitar 205 puntos de P .

g.- El siguiente poliedro lo expresamos dando la matriz A ampliada con los términos independientes:

$$
\begin{bmatrix}
9 & 7 & 16 & 8 & 24 & 5 & 3 & 7 & 8 & 4 & 6 & 5 & 310 \\
12 & 6 & 6 & 2 & 20 & 8 & 4 & 6 & 3 & 1 & 5 & 8 & 395 \\
15 & 5 & 12 & 4 & 4 & 5 & 5 & 5 & 6 & 2 & 1 & 5 & 480 \\
18 & 4 & 4 & 18 & 28 & 1 & 6 & 4 & 2 & 9 & 7 & 1 & 395 \\
-12 & 7 & 5 & 12 & 10 & 4 & 6 & -2 & 3 & 7 & 0 & 8 & 345 \\
0 & -15 & 9 & 4 & 12 & 9 & 4 & 3 & 5 & 8 & 13 & 12 & 167 \\
7 & 9 & -12 & 7 & -5 & 8 & 4 & 7 & 6 & 4 & 7 & 14 & 99 \\
10 & 6 & 13 & -10 & 8 & -7 & 9 & 6 & 9 & 4 & -4 & 0 & 172 \\
-5 & 3 & 7 & 2 & -11 & 5 & 9 & -6 & 14 & 7 & 12 & 9 & 169 \\
8 & 0 & 0 & 9 & 15 & -11 & 4 & 12 & 9 & -7 & 13 & 16 & 152
\end{bmatrix}
$$

cotas inferiores: (0,1,6,6,0,0,6,5,0,3,0,0)

cotas superiores: (2,3,9,10,1,2,10,8,2,6,1,2)

El número de vectores eficientes obtenidos es de 1835, y se han visitado 102232 vectores.

Programa multobj2

```
Program multobj2(input,output);
const n1=15;m1=16;
type vect1=array[0..n1] of integer;
     vect2=array[1..n1] of real;
     a1=array[1..m1,1..n1] of real;
     c1=array[1..m1] of real;
     pnodo=^nodo;
     nodo=record x:vect1;
          iz,de:pnodo
     end;
var raiz:pnodo;
    i,j,l,m,n,k,k1,nv,ne:integer;
    a:a1;
    b1,b2,b3,d:c1;
    b,x,y,e,ci,cs:vect1;
    t:boolean;

function fac(d:c1):integer;
var i,l,J:integer;
begin l:=3;J:=3;
  for i:=1 to m do
  begin if((d[i]+b1[i])<0) then l:=0 else
          if(d[i]<0) then l:=1 else
            if((d[i]-b3[i])<0) then l:=2;
    if (l<J) then J:=l
  end;
  fac:=J
end;

function cr(J:integer;v:vect1;d:c1):integer;
var i,l:integer;
begin l:=1;i:=0;
  while ((l=1) and (i<m)) do
  begin i:=i+1;if((d[i]+b1[i])<a[i,J]) then l:=0
  end;
  cr:=l
end;

function de(J:integer;v:vect1;d:c1):integer;
var i,l:integer;
begin l:=1;i:=0;
  while ((l=1) and (i<m)) do
  begin i:=i+1;if((d[i]+b1[i])<-a[i,J]) then l:=0
  end;
  de:=l
end;

function dom(v,w:vect1):boolean;
var t:boolean;
begin t:=true;i:=0;
  while(t and (i<n)) do
  begin i:=i+1;
    if (v[i]<w[i]) then t:=false
  end;
  dom:=t
end;
```

```
procedure imprimir(p:pnodo);
var i:integer;
begin
  if (p<>nil) then
  begin
    imprimir(p^.de);
    for i:=1 to n do write(p^.x[i]:3);ne:=ne+1;
    write('    ');if((ne mod (20 div n))=0) then writeln;
    imprimir(p^.iz)
  end;
end;

function test1(b:vect1;p:pnodo;k:integer):boolean;
var l:integer;
    t:boolean;
begin t:=true;l:=(k mod n)+1;
  if (p<>nil) then if dom(p^.x,b) then t:=false else
  begin t:=test1(b,p^.de,l);
    if (t and (b[k]<=p^.x[k])) then t:=test1(b,p^.iz,l);
  end;
  test1:=t;
end;

procedure insertar2(b:vect1;var p:pnodo;k:integer);
var l:integer;
begin l:=(k mod n)+1;
  if (p=nil) then
  begin new(p);p^.x:=b;p^.iz:=nil;p^.de:=nil
  end else
  begin if b[k]<=p^.x[k] then insertar2(b,p^.iz,l)
    else insertar2(b,p^.de,l);
  end;
end;

procedure borrar(b:vect1;var p:pnodo;k:integer);
var raiz2:pnodo;

procedure recorrer(p:pnodo);
begin if (p<>nil) then
  begin recorrer(p^.de);
    recorrer(p^.iz);
    if not(dom(b,p^.x)) then insertar2(p^.x,raiz2,k);
    dispose(p);
  end;
end;

begin raiz2:=nil;
  recorrer(p);
  p:=raiz2;
end;

procedure test2(b:vect1;var p:pnodo;k:integer);
var l,j:integer;
begin l:=(k mod n)+1;
  if (p<>nil) then if dom(b,p^.x) then borrar(b,p,k) else
  begin test2(b,p^.iz,l);
    if b[k]>p^.x[k] then test2(b,p^.de,l);
  end;
end;
```

```
procedure insertar(b:vect1;var r:rnodo;k:integer);
var j,l:integer;
begin l:=(k mod n)+1;
  if (r=nil) then
  begin new(r);r^.x:=b;r^.iz:=nil;r^.de:=nil
  end else if not(dom(r^.x,b)) then if dom(b,r^.x) then
  begin borrar(b,r,k);insertar2(b,r,k)
  end else
  begin if b[k]<=r^.x[k] then
    begin if test1(b,r^.de,l) then insertar(b,r^.iz,l)
    end else
    begin test2(b,r^.iz,l);insertar(b,r^.de,l)
    end;
  end;
end;

procedure avant(y,e:vect1;d:c1);
var i,j,l:integer;
    d1:c1;
    z:vect1;
begin nv:=nv+1;
  if(y[0]>1) then
  begin k:=1;insertar(y,raiz,k)
  end;
  for i:=1 to n do if((e[i]=1) and (cr(i,y,d)=1)) then
  begin z:=y;z[i]:=z[i]+1;
    if(z[i]>=cs[i]) then e[i]:=0;
    for j:=1 to m do d1[j]:=d[j]-a[j,i];
    z[0]:=fac(d1);
    avant(z,e,d1);e[i]:=0
  end else
  begin if (y[0]<>3) then
    begin  j:=0;for l:=1 to n do if (e[l]>0) then j:=j+1;
      k:=1;
      if((j>0) or ((y[0]=1) and test1(y,raiz,k))) then
      if ((e[i]=-1) and (de(i,y,d)=1)) then
      begin z:=y;z[i]:=z[i]-1;
        if(z[i]<=ci[i]) then e[i]:=0;
        for j:=1 to m do d1[j]:=d[j]+a[j,i];
        z[0]:=fac(d1);
        avant(z,e,d1);e[i]:=0
      end;
    end;
  end;
end;

procedure top(var x:vect1);
var i,j:integer;
    z:vect1;
    d1:c1;
begin j:=0;z:=x;
  while (j=0) do
  begin j:=1;
    for i:=1 to m do
    begin d[i]:=b2[i];for j:=1 to n do d[i]:=d[i]-a[i,j]*z[j]
    end;
    for i:=1 to n do
    begin if ((cr(i,z,d)=1) and (z[i]<cs[i])) then
      begin z[i]:=z[i]+1;
```

```
        for j:=1 to m do d1[j]:=d[j]-a[j,i];
        if (fac(d1)>1) then
        begin j:=0;x:=z;d:=d1
        end;
      end;
    end;
  end;
end;

begin
  raiz:=nil;nv:=0;ne:=0;
  read(n,m);
  for i:=1 to m do for j:=1 to n do read(a[i,j]);
  for i:=1 to m do read(b2[i]);
  for i:=1 to n do read(ci[i]);
  for i:=1 to n do read(cs[i]);
  for i:=1 to m do
  begin b1[i]:=0;for j:=1 to n do if (a[i,j]<0) then
    b1[i]:=b1[i]-a[i,j];
  end;
  for i:=1 to n do read(x[i]);top(x);
  for l:=2**n-1 downto 0 do
  begin k1:=l;for i:=1 to n do
    begin j:=k1 mod 2;
      if (j=0) then e[i]:=-1 else e[i]:=1;
      k1:=k1 div 2
    end;
    t:=true;
    for i:=1 to n do
    begin if ((x[i]<=ci[i]) and (e[i]=-1)) then t:=false;
      if ((x[i]>=cs[i]) and (e[i]=1)) then e[i]:=0
    end;
    if t then
    begin
      for i:=1 to n do if (e[i]>=0) then y[i]:=x[i] else
      begin y[i]:=x[i]-1;if(y[i]<=ci[i]) then e[i]:=0
      end;
      for i:=1 to m do
      begin d[i]:=b2[i];for j:=1 to n do d[i]:=d[i]-a[i,j]*y[j]
      end;
      for i:=1 to m do
      begin b3[i]:=0;
        for j:=1 to n do if((e[j]=1) and (b3[i]<a[i,j])) then
        b3[i]:=a[i,j]
      end;
      y[0]:=fac(d);
      if (y[0]>0) then avant(y,e,d) else nv:=nv+1;
    end;
  end;
  writeln(nv);
  imprimir(raiz);
  writeln(ne)
end.
```

Bibliagrafía

Aho A.V., Hopcroft J.E., Ullman J.D.: The design and analysis of algorithms. Addison Wesley 1974.

Aho A.V., Hopcroft J.E., Ullman J.D.: Data structures and algorithms. Addison Wesley 1983.

Aust R.J.: A dynamic programming branch and bound algorithm for pure integer programming.

Balas E.: An additive algorithm for solving linear programs with zero-one variables. Oper. Res. 1965.

Bazaraa M.S.: Linear programming and network flows. John Wiley 1977.

Bazaraa M.S.: Non linear programming. John Wiley 1979.

Bentley J.L.: Multidimensional binary search trees used for associated searching. ACM 1975.

Bertier P. y Roy B.: Une procedure de resolution pour une classe de problemes pouvant avoir un charactere combinatoire. Int.Comp. Cent. Bull. 1965.

Chankong Vira: Multiobjective Decision Making: Theory and methodology. North Holland 1984.

Dantzig G.B.: Solution of a large scale travelling salesman problem. Oper. Res. 1954.

Faaland B.H., Hillier F.S.: Interior path methods for heuristic integer programming procedures. Op. Research 1979.

Fernández F.R.: Estructuras de dominancia: introducción a la programación múltiple. Dpto. de Est. Fac. de Ciencias de Malaga 1980.

Finkel R.A., Bentley J.L.: Quad-trees; a data structure for retrieval on composite keys, 1974.

Fox B.L.: Data structures and computer science techniques in operations research. Oper. Research 1978.

Gavish B. y Pirkul H.: Zero-one integer programs with few constraints. Efficient branch and bound algorithms. European Jou. of Oper. Res. 1985.

Geoffrion A.M.: Lagrangean Relaxation for integer programming. North Holland 1974.

Geoffrion A.M.: Proper efficiency and the theory of vector maximization. J. Math. Anal. Appl. 1967.

Glover F.: A multiphase dual algorithm for the zero-one integer programming problems. Oper. Res. 1965.

Goddard L.S.: Técnicas matemáticas de investigación operacional. Editorial Alhambra 1969.

Gomory R.E.: Outline of an algorithm for integer solutions to linear programs. Bull. Amer. Math. Soc. 1958.

Gomory R.E.: An algorithm for the mixed integer problem. Math. Amer. Math. Soc. 1960.

Gurari y O. Ibarra : An NP-complete number theoretic problem. University of Minnesota. Computer Science 1977.

Habenicht : Quad trees, a datastructure for discrete vector optimization problems, 1983.

Herbert Moskowitz : Operations Research techniques. Prentice Hall 1979.

Hillier : Introduction to operations research. Ed. Holden Day 1974.

Indice

UNIVERSIDAD DE SEVILLA

FACULTAD DE CIENCIAS MATEMÁTICAS

Reunido el Tribunal [illegible] por los abajo firmantes el día de la fecha, para juzgar la Tesis Doctoral de Antonio Pozo Chica titulada "Eficiencia y estructuras financieras: Curva Pareto eficiente" y otorgarle la calificación de APTO "CUM LAUDE"

Sevilla, 20 de diciembre 1.988

El Vocal, Miguel Sánchez

El Vocal, A. Cantó

El Vocal,

El Presidente,

El Secretario, Julio Bolaños

El Doctorado,