UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE MATEMÁTICAS

TESIS DOCTORAL

MODIFICACIONES DEL ALGORITMO DE GRADO MÍNIMO PARA LA RESOLUCIÓN DE SISTEMAS SPARSES

CARMELO HERRERA SÁNCHEZ

Las Palmas de Gran Canaria, 1996

70/1995-96

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
UNIDAD DE TERCER CICLO Y POSTGRADO

Reunido el día de la fecha, el Tribunal nombrado por el Excmo. Sr. Rector Magfco. de esta Universidad, el/a aspirante expuso esta TESIS DOCTORAL.

Terminada la lectura y contestadas por el/a Doctorando/a las objeciones formuladas por los señores miembros del Tribunal, éste calificó dicho trabajo con la nota de APTO CUM LAUDE
Las Palmas de Gran Canaria a 15 de julio de 1996.

El/a Presidente/a: Dr. D. Gabriel Winter Althaus,

El/a Secretario/a: Dr. D. Antonio Suárez Sarmiento,

El/a Vocal: Dr. D. José Ramón Franco Brañas,

El/a Vocal: Dr. D. Vicente Novo Sanjurjo,

El/a Vocal: Dr. D. Pedro José Jiménez Olivo,

El/a Doctorando/a: D. Carmelo Herrera Sánchez,

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DOCTORADO EN MATEMÁTICAS

DEPARTAMENTO DE MATEMÁTICAS

PROGRAMA DE DOCTORADO DE MÉTODOS FINITOS EN INGENIERÍA

Modificaciones del algoritmo de grado mínimo para la resolución de sistemas sparses

Tesis Doctoral presentada por D. Carmelo Herrera Sánchez

Dirigida por el Dr. D. Pedro Almeida Benítez

El Director — Pedro Almeida Benítez

El Doctorando — Carmelo Herrera Sánchez

Mayo 1996

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Doctorado en Matemáticas

Departamento de Matemáticas

Programa de Doctorado de Métodos de Elementos Finitos en Ingeniería

TESIS DOCTORAL

Modificaciones del algoritmo de Grado Mínimo para la resolución de sistemas sparses.

Carmelo Herrera Sánchez

Mayo 1996

AGRADECIMIENTOS

A Pedro Almeida Benítez , director de la tesis, por su constancia y apoyo para que este trabajo tuviese un final feliz.

A Gustavo Montero , por su ayuda inestimable , en el manejo del Código Neptuno.

A todos mis compañeros del Departamento de Matemáticas.

A mi familia

ÍNDICE

Capítulo 1. Introducción

1.0 Introducción.

A la hora de resolver un sistema de ecuaciones lineales Ax = b, donde A es una matriz de orden n simétrica y definida positiva, donde b es un vector que pertenece a un espacio vectorial de dimensión n llamado término independiente, y x es el vector solución, hay cuatro fases que se pueden identificar durante el proceso computacional, que son:

❶ Ordenación: Consiste en encontrar una buena ordenación para la matriz A, mediante intercambios adecuados de filas y columnas, obteniéndose éstos a través de una matriz de permutación P.

❷ Factorización: Obtención de la factorización de la matriz A. La matriz permutada PAP^t se factoriza mediante el algoritmo de Cholesky, produciendo una matriz subtriangular L.

❸ Almacenamiento: La información que proporciona el factor L se almacena mediante algún esquema adecuado. Con la información dada por el factor L de Cholesky obtenido a través de PAP^t se puede establecer el esquema de almacenamiento o bien con las matrices L y U obtenidas por el método de Gauss.

❹ Resolución de los sistemas triangulares obtenidos después de la factorización.

A la hora de elegir un método para la resolución, hemos de cubrir 2 objetivos que son:

❶ Reducir el almacenamiento en el ordenador.

❷ Reducir el tiempo de ejecución.

Con respecto al almacenamiento de matrices dispersas en el ordenador, hemos de tener en cuenta dos aspectos fundamentales respecto a la memoria:

- Memoria principal: Ésta es la memoria empleada para retener los valores numéricos.

- Memoria suplementaria: Ésta es la memoria usada para los indicadores, subíndices y cualquier otra información que se necesite para tener implementada la matriz en el ordenador.

En cuanto al tiempo de ejecución, debemos considerar los cuatro pasos señalados anteriormente para que se obtenga el proceso completo de cálculo, éstos son: ordenación, factorización, almacenamiento y resolución de los sistemas.

Aún después de haber encontrado la ordenación adecuada, los tiempos de ejecución pueden variar, ya que todavía quedan por realizar muchas operaciones hasta llegar a la solución del sistema.

Toda vez que hemos de pagar por la memoria en el ordenador, independientemente de cómo sea empleada, cualquier evaluación de la demanda de memoria para un método de resolución ha de incluir una descripción de la forma en que la matriz o matrices implicadas van a ser almacenadas, de manera que la memoria suplementaria se pueda incluir junto con la memoria principal en la demanda de memoria. La comparación de dos estrategias distintas con respecto al criterio de almacenamiento puede proporcionar estructuras de datos diferentes, teniendo memoria suplementarias muy distintas. Así, un método que es superior en términos de memoria principal puede ser inferior cuando la memoria suplementaria se incluye en la comparación. Este punto se ilustra en la figura 1.0.1.

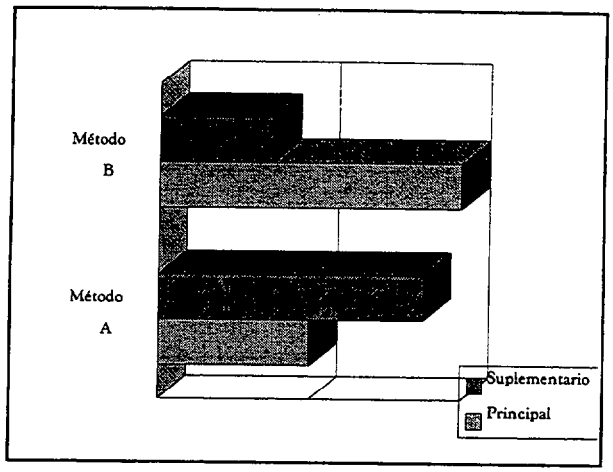


Fig. 1.0.1.

Como simple ejemplo, consideremos las dos ordenaciones de la matriz simétrica de la figura 1.0.2 junto con sus factores correspondientes L y $\overline{L}$. Los elementos del triángulo inferior de L (excluyendo la diagonal principal) están almacenados fila a fila en un vector, con otro vector auxiliar que registra sus subíndices columna. Un tercer vector indica la posición de cada fila, y un cuarto vector contiene las entradas diagonales de L. La matriz $\overline{L}$ está almacenada empleando el llamado esquema de almacenamiento de perfil o envolvente, que se describirá más adelante. Los no ceros de A están indicados por un asterisco *, y con $\otimes$ denotaremos las entradas de L o $\overline{L}$ en los que se producen rellenos de huecos [2].

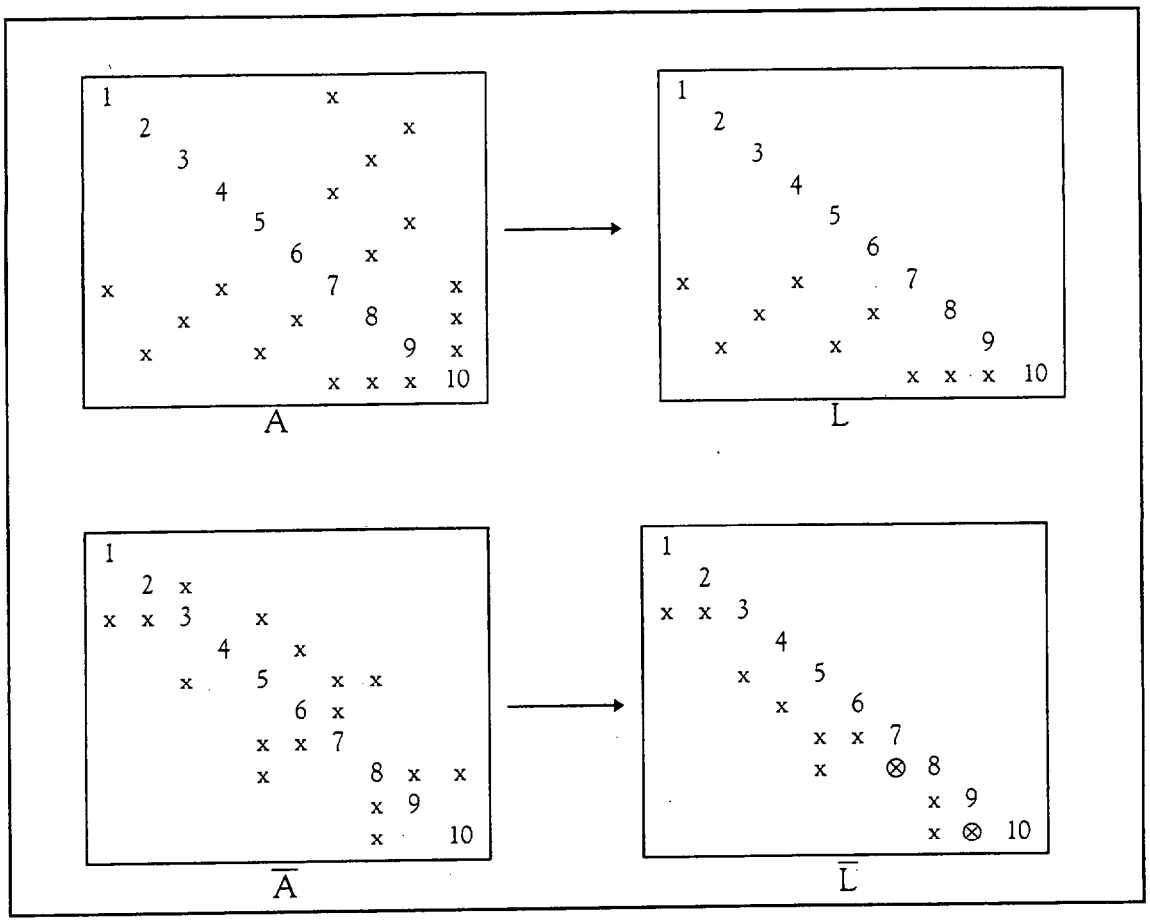


Fig. 1.0.2.

Los ejemplos de las figuras 1.0.2 y 1.0.3 ilustran algunos aspectos importantes referentes a las ordenaciones y esquemas de almacenamiento. Superficialmente, la ordenación 1, correspondiente a A parece que es mejor que la ordenación 2 puesto que no produce ningún relleno de huecos, mientras que la última ordenación da lugar a dos posiciones en las que se producen relleno de huecos. Además, el esquema de almacenamiento empleado para $\overline{L}$ parece inferior que el usado para L, puesto que está explotada la dispersión de la matriz, lo cual el almacenamiento para $\overline{L}$ que la dispersión en L, si está explotada. Sin embargo, a causa de las diferencias en la memoria suplementaria, la segunda combinación ordenación/ almacenamiento origina una demanda de almacenamiento total inferior. Por supuesto, las diferencias en este caso son triviales, pero el punto de vista es válido. A medida que aumentamos la sofisticación de nuestro procedimiento de almacenamiento, explotando más y más ceros, la memoria principal disminuye, pero la suplementaria normalmente aumenta. A

veces compensa ignorar algunos ceros, puesto que la memoria suplementaria requerida para explotarlos es mayor que la disminución en memoria principal.

En resumen:

❶ Los esquemas de almacenamiento para matrices dispersas suponen dos componentes: memoria principal y memoria suplementaria.

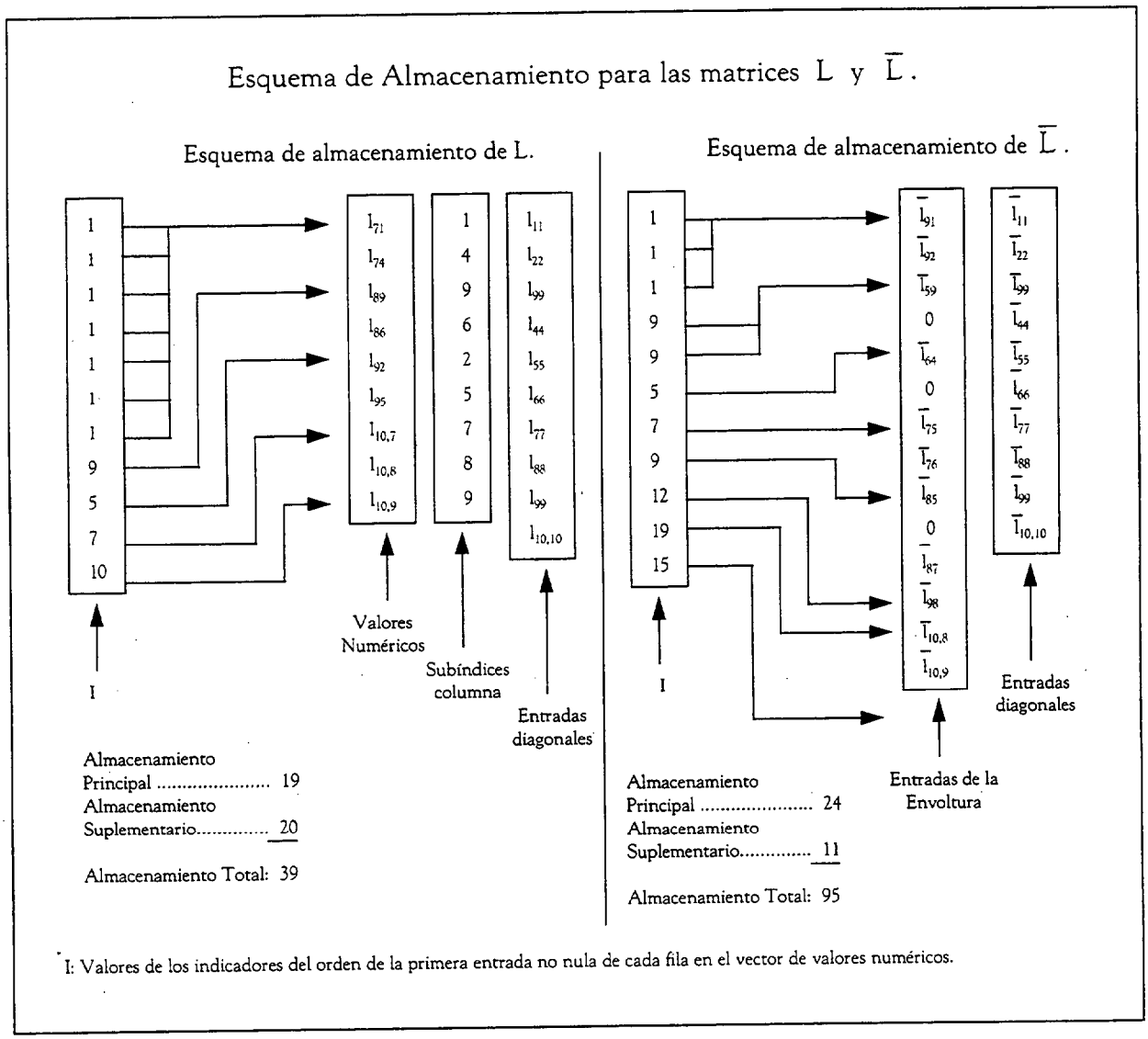


Fig. 1.0.3.

❷ Las comparaciones de las estrategias de ordenación han de tener en cuenta el esquema de almacenamiento a emplear, y la comparación va a ser prácticamente relevante.

En cuanto al tiempo de ejecución, consideramos los cuatro pasos del proceso completo de cálculo: ordenación, factorización, almacenamiento y resolución.

Debemos establecer el esquema de almacenamiento apropiado para L, y con objeto de ello, determinaremos su estructura. Este paso de almacenamiento varía también en coste dependiendo del esquema de ordenación y almacenamiento empleados. Finalmente, los diferentes experimentos realizados, muestran que los distintos esquemas de almacenamiento pueden llevar a diferencias substanciales en la producción de operaciones aritméticas por segundo durante la factorización y la resolución triangular. Normalmente, la ejecución de un programa de matriz dispersa debería ser a grandes rasgos proporcional a la cantidad aritmética ejecutada. Sin embargo, las diferentes ordenaciones y estructuras de datos pueden dar lugar a notables discrepancias en la constante de proporcionalidad. Así, los totales de operaciones aritméticas pueden no ser una medida fiable para la comparación de métodos de resolución, o como mucho han de ser empleados con cautela. La constante de proporcionalidad está afectada no solo por la estructura de datos sino también por la arquitectura del ordenador, el compilador y el sistema operativo.

Además de la variación de los costes respectivos al ejecutar cada uno de los pasos anteriores, las comparaciones de distintas estrategias dependen a menudo del contexto particular en el que el problema se resuelva. Si la matriz del sistema dado se va a resolver solamente una vez, una comparación de estrategias debería seguramente incluir el tiempo de ejecución requerido para producir la ordenación y establecer el esquema de almacenamiento.

Sin embargo, a veces se han de resolver muchos problemas diferentes que tienen la misma estructura, y puede ser razonable ignorar este coste de inicialización en comparación de métodos, puesto que el volumen de tiempo de la ejecución es dada por la factorización y resoluciones triangulares. Incluso en otras circunstancias, hay que resolver sistemas que solo difieren en el vector de términos independientes. En este caso, puede que sea razonable comparar las estrategias simplemente en base a sus respectivos tiempos de resolución triangular.

En resumen, los puntos principales son:

❶ La resolución global de $Ax = b$ supone cuatro pasos básicos. Sus tiempos relativos de ejecución, en general, varían substancialmente en esquemas diferentes de ordenación y de almacenamiento.

❷ Dependiendo del contexto del problema, los tiempos de ejecución de algunos de los pasos mencionados anteriormente pueden ser irrelevantes en la práctica cuando se comparan métodos.

1.1 Factorización de Cholesky.

Sea A una matriz simétrica que denotaremos por A_{i*} y A_{*i} la fila y columna enésima, respectivamente.

En el análisis de matrices dispersas, necesitamos contar el número de no ceros en un vector o matriz; emplearemos $\eta(\square)$, para indicar el número de entradas no ceros en $\square$.

Si A es una matriz real de orden n x n, simétrica y definida positiva, entonces existe una única factorización LL^t, siendo L una matriz triangular inferior y con entradas diagonales positivas.

En la matriz A después de factorizada se tiene la unicidad del factor L; su demostración es la siguiente:

$A = LL^t$; $A = L_1L_1^t \Rightarrow LL^t = L_1L_1^t \Rightarrow L^{-1}L = L_1^t (L^t)^{-1} = \Delta$, siendo Δ una matriz diagonal. Como $\Delta = L^{-1}L$, se deduce que $L = L_1 \Delta$.

De $\Delta = L_1^t (L^t)^{-1}$, se obtiene $L_1^t = \Delta L^t$ luego $L_1 = L^t \Delta = L\Delta$, sustituyendo $L_1 = L\Delta$ en $L = L_1 \Delta$, se obtiene $L = L_1 \Delta = L \Delta \Delta \Rightarrow L = L\Delta^2$, luego $\Delta^2 = I$, por lo tanto $\Delta = (\partial_{ii})$ siendo $\partial_i = \pm 1$, y la matriz L queda determinada salvo el sig-

no de las columnas; y las entradas diagonales de L son positivas y también las de L_1, luego $\Delta = I$ y por lo tanto la factorización de L_1 única.

❑ Distintas formas en que puede ser calculada L.

Existen varias formas de calcular L y estas son:

- ❑ Producto externo
- ❑ Orlado
- ❑ Producto interno

❑ El producto externo es el llamado algoritmo de Cholesky, éste esquema se puede describir partiendo de $A = A_0 = H_0 = \begin{pmatrix} d_1 & v_1^t \\ v_1 & \bar{H}_1 \end{pmatrix}$, descomponiendo esta matriz en $A = L_1 \begin{pmatrix} 1 & 0 \\ 0 & H_1 \end{pmatrix} L_1^t = L_1 A_1 L_1^t$ y haciendo $A_1 = \begin{pmatrix} 1 & 0 \\ 0 & H_1 \end{pmatrix} = L_2 A L_2^t$; de ésta manera llegamos a $A_{n-1} = L_n I_n L_n^t$ y después de dar n pasos el algoritmo queda:

$$A = L_1 L_2, \ldots, L_n L_n^t, \ldots, L_2^t L_1^t = LL^t$$

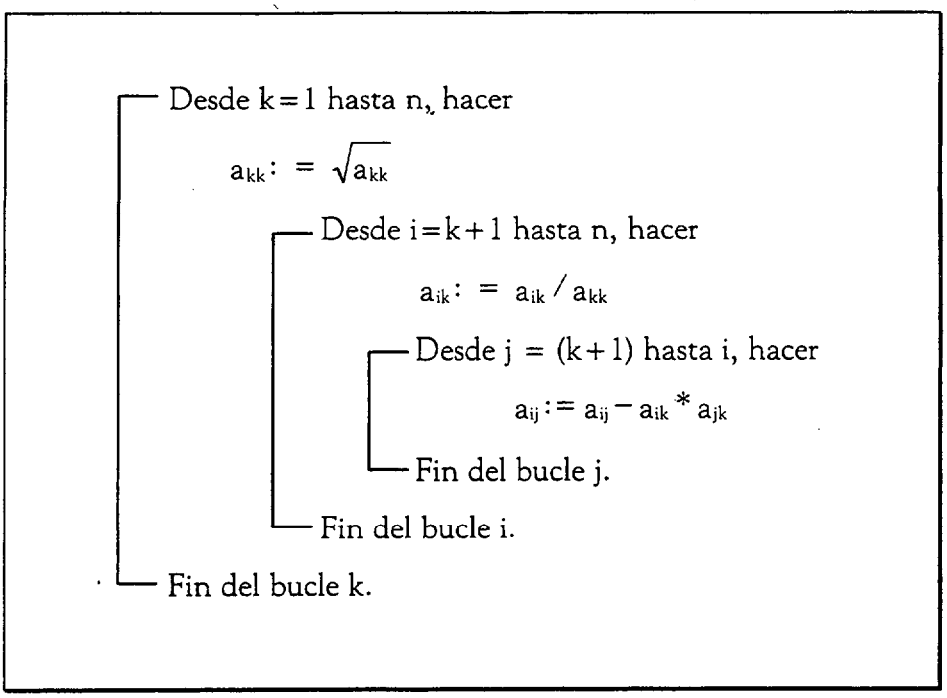
La i-ésima columna de L es la i-ésima columna de L_i. En el esquema del producto externo las columnas de L se calculan una a una. Al mismo tiempo cada paso supone la modificación de H_i por el producto externo $V_i V_i^t / d_i$ para dar H_i, que es la submatriz que queda por factorizar. Por lo tanto quedan las siguientes relaciones:

$$l_{kk} = \sqrt{a_{kk}}$$

$$l_{ik} = a_{ik} / l_{kk} \qquad \text{para } k+1 \le i \le n$$

$$l_{ij} = a_{ij} - l_{ij} * l_{jk} \qquad \text{para } k+1 \le j \le i \le n$$

De estas relaciones nace el siguiente algoritmo:



❑ En el esquema del orlado las relaciones de recurrencia son:

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-i} l_{ik}\, l_{jk} \right) / l_{jj} \qquad \text{para } j \leq i-1$$

$$l_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{1/2}$$

De esta manera, las filas de L se calculan una vez cada vez.

La parte de la matriz que aún queda por factorizar no será abordada hasta que la parte de L se vaya a calcular.

La eliminación Gaussiana simétrica (método de Cholesky) aplicada a una matriz simétrica y definida positiva, no necesita pivoteo para mantener la estabilidad numérica, luego podemos reordenar A simétricamente sin tener en cuenta la estabilidad numérica y la podemos reordenar antes de la factorización numérica.

Como el orden se puede determinar antes de la factorización entonces podemos determinar las posiciones de los rellenos de huecos que se pueden producir du-

rante la factorización. De éste modo la estructura de datos para almacenar la matriz L lo podemos hacer antes de la factorización numérica real.

❑ Esquema del producto interno. Veamos por último el esquema del producto interno para determinar las entradas de L.

Identificando las entradas de A con las entradas de LL^t que ocupan su misma posición, tenemos que :

$$A = LL^t \rightarrow a_{ij} = \sum_{K=1}^{n} l_{ik} l^t_{kj} = \sum_{K=1}^{n} l_{ik} l_{kj} = \sum_{K=1}^{j} l_{ik} l_{kj} \rightarrow a_{ij} = \sum_{K=1}^{j} l_{ik} l_{kj}$$

Según estas relaciones quedan las siguientes fórmulas:

Para j=1, 2, ..., n, se tiene que calcular:

$$ljj = \left(ajj - \sum_{k=1}^{j-1} l_{jk}^2 \right)^{1/2}$$

Para l=j+1, j+2, ..., n, se tiene que calcular:

$$lij = \frac{\left(aij - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right)^{1/2}}{l_{jj}}$$

Según estas últimas relaciones podemos elaborar el siguiente algoritmo:

```
Desde j=1 hasta n, hacer
    Desde i=1 hasta n, hacer
        Desde k=1 hasta (j-1), hacer
            aij:= aij - aik * ajk
        Fin del bucle k
        Si i=j, entonces aij:= √aij
            sino aij:=aij/ajj
        Fin si
    fin del bucle i
Fin del bucle j.
```

Como la versión del producto externo, las columnas de L se calculan una por una, o sea, se trata de una factorización columna a columna, la parte de la matriz que queda sin factorizar no se aborda durante el procesamiento del esquema, por eso se denomina interno.

1.2 Factorización de matrices dispersas.

Cuando se factoriza una matriz A que sea dispersa se pueden producir rellenos de huecos o sea términos que en L se hacen no nulos y en la matriz A eran nulos, este proceso de relleno de huecos se le conoce cómo efecto fill-in (literatura anglosajona).

Al resolver un sistema de ecuaciones lineales en el que la matriz que lo define es dispersa, el orden en que se tratan las filas o las columnas tiene una importancia fundamental. Por ejemplo, si consideramos la matriz simbólica A de un sistema de ecuaciones lineales cuyo patrón de elementos distintos de cero es el de la figura 1.2.0 y se utiliza la eliminación de Gauss para resolver dicho sistema, los 46 elementos que en el transcurso de dicha factorización se hacen distintos de cero (elementos de relleno ó fill-in) son los que en la figura 1.2.1 aparecen representados por ⊗ [23].

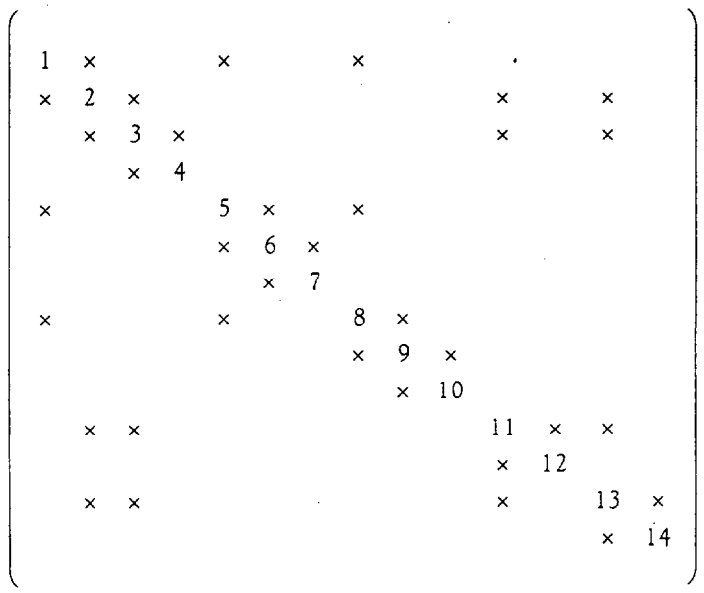


Fig 1.2.0.

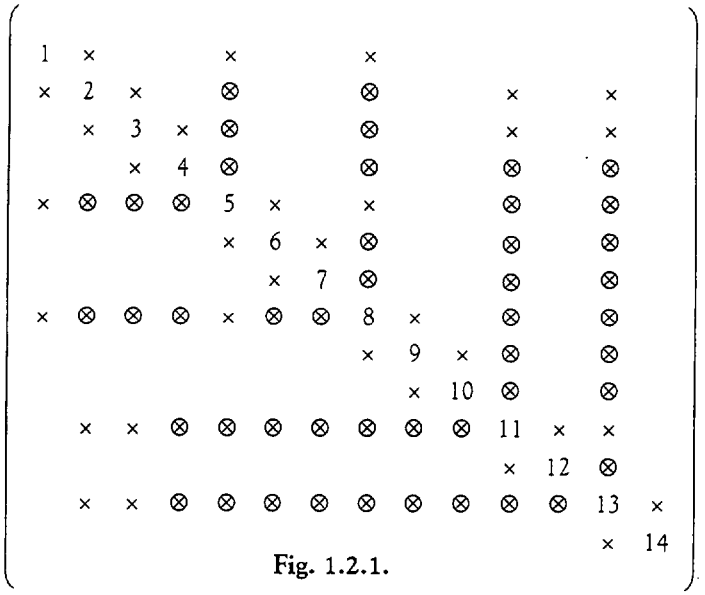


Fig. 1.2.1.

Por el contrario si se reordenan las filas y las columnas de acuerdo con un criterio que veremos más adelante (algoritmo de grado mínimo), el patrón de elementos distintos de cero de la matriz resultante es el de la figura 1.2.2, si esta nueva matriz reordenada se factoriza también mediante eliminación de Gauss, el número de elementos cero que se harían distintos de cero sería nulo.

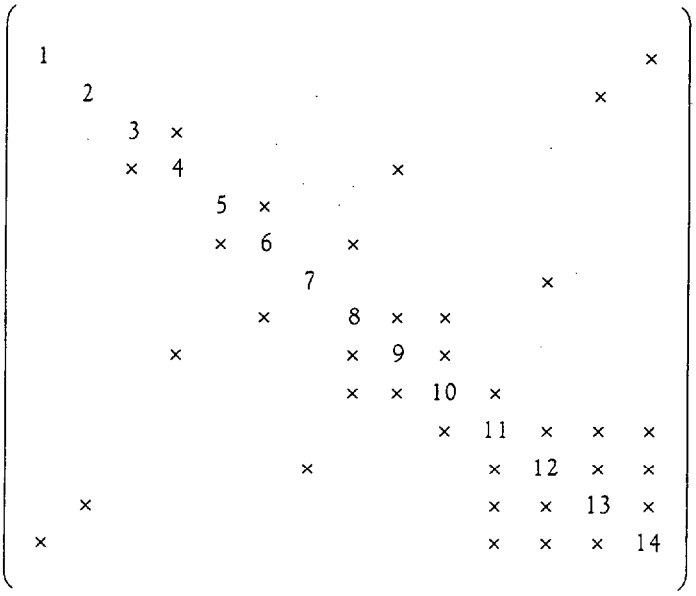


Fig. 1.2.2

Trabajar con matrices reordenadas en las que el número de elementos rellenos se reduce considerablemente presenta tres ventajas fundamentales:

❶ Disminución del número de posiciones de memoria que han de reservarse para los nuevos elementos que se harán distintos de cero en un proceso de factorización.

❷ Disminuciones del número de operaciones a realizar, y por lo tanto, el tiempo total de cálculo para factorizar la matriz y resolver el correspondiente sistema.

❸ Mejora de la estabilidad numérica del proceso global de resolución del sistema al disminuir el número de elementos a considerar y por tanto disminuir la probabilidad de encontrar grandes diferencias entre ellos, errores de cancelación, etc.

En particular, el número de estos no-ceros extras creados durante la eliminación depende muchísimo de la elección de los elementos pivotes. Así una mala elección del pivote no sólo puede llevar a una inestabilidad numérica, sino también estropear el modelo original de la matriz dispersa.

Es por lo tanto deseable encontrar una secuencia de pivotes que nos aseguren la estabilidad numérica sino que también limiten las entradas fill-in lo más posible. En el método de Cholesky para matrices A simétricas y definidas positivas, la situación es favorable porque en este caso la selección del pivote no es crucial para la estabilidad numérica. De este modo, en vez de seleccionar pivotes consecutivos en la diagonal se podrían seleccionar en otro orden sin perder la estabilidad numérica, con lo cual se minimiza el efecto fill-in. Ésto nos conduce a encontrar una matriz de permutación P tal que $PAP^t = LL^t$ teniendo un factor de Cholesky que es tan disperso como sea posible.

El fenómeno de rellenos de huecos desempeña un papel muy importante durante la eliminación dispersa. Si estos rellenos son muchos, no sólo pueden destruir cualquier estructura de dispersidad sino también dar al traste con la consideración de la matriz como dispersa.

Más adelante veremos que para evitar este relleno de huecos se recurre a efectuar una reordenación de filas y columnas que quedarán materializadas por un con-

junto de permutaciones, de tal forma que en la matriz resultante al factorizar aparezcan menos elementos de relleno que en la matriz original ya que, si el sistema a resolver es Ax=b y se le aplica a A un conjunto de permutaciones de elementos representado por la matriz de permutación P, el sistema se puede reescribir(P A P^t) Px = P b ya que $P^tP = I$.

Haciendo y = Px y c = P b se tiene que B y = c, donde B = P A P^t, siendo B la matriz A reordenada, siendo también la matriz B dispersa y simétrica, y si A es definida positiva también lo es B.

La idea es encontrar una matriz P adecuada que produzca el menor relleno posible al factorizar B. Si la matriz A es de orden n, el número posible de ordenaciones es n!, que evidentemente es imposible analizarlas todas.

Anteriormente, durante la premultiplicación por A se produce un intercambio de filas de A, y la posmultiplicación de A por P produce un intercambio de columnas de A.

Una buena elección de la matriz de permutación P puede dar resultados muy buenos en los rellenos de huecos. Una mala elección de la matriz de permutación P puede tener efectos dramáticos.

O Veamos un ejemplo en las figuras 1.2.3 y 1.2.4.

$$\text{Sea } A = \begin{pmatrix} 1 & x & x & x & x \\ x & 2 & & & \\ x & & 3 & & \\ x & & & 4 & \\ x & & & & 5 \end{pmatrix} = L\,L^t \quad \textit{siendo } L = \begin{pmatrix} 1 & & & & \\ x & 2 & & & \\ x & x & 3 & & \\ x & x & x & 4 & \\ x & x & x & x & 5 \end{pmatrix}$$

Fig 1.2.3.

Se observa que L es una matriz completa donde el efecto fill-in es máximo , si intercambiamos la primera y última filas y columnas de A, se obtiene un factor de Cholesky :

$$PAP^t = \begin{pmatrix} 5 & & & & x \\ & 2 & & & x \\ & & 3 & & x \\ & & & 4 & x \\ x & x & x & x & 1 \end{pmatrix} = LL^t \quad \text{siendo} \quad L = \begin{pmatrix} 5 & & & & \\ & 2 & & & \\ & & 3 & & \\ & & & 4 & \\ x & x & x & x & 1 \end{pmatrix}$$

Fig 1.2.4.

Se observa que el efecto fill-in se ha reducido considerablemente.

1.3 Resolución de sistemas triangulares.

Una vez se ha realizado la factorización, se tienen que resolver los sistemas triangulares $A\,x = b$, siendo $A = L\,L^t$, luego al sustituir queda $L\,L^t\,x = b$, y llamando $L^t\,x = y$ queda $L\,y = b$.

Del sistema $L\,y = b$ se obtiene el vector y, que sustituido en $L^t\,x = y$ se obtiene la solución del sistema.

Si se tiene un sistema triangular de orden n, $Tx = b$, siendo T una matriz regular y además triangular, se tienen dos formas de resolver el sistema que sólo difieren únicamente en el orden en que las operaciones se ejecutan.

Se pueden emplear los productos internos, siendo las ecuaciones dadas por:

$$x_i = \left(b_i - \sum_{k=1}^{n-1} t_{ik} x_k\right) / t_{ii}, \quad \text{para } i = 1,\ 2,\ ,\ldots,\ n.$$

Estas fórmulas originan el siguiente algoritmo:

```
Desde i = 1 hasta n, hacer
    Desde j = 1 hasta (i-1), hacer
        b( i ) = ( b ( i ) - lij * bj ) / lii
        x ( i ) = b ( i )
    Fin del bucle j.
Fin del bucle i.
```

Otra alternativa del método emplea las entradas matriciales de T de la misma manera que la versión del producto externo de la factorización. Sea el sistema T x = b.

$$t_{11}x_1 = b_1$$
$$t_{21}x_1 + t_{22}x_2 = b_2$$
$$\vdots$$
$$t_{i1}x_1 + t_{i2}x_2 + \ldots\ldots\ldots\ldots + t_{ii}x_i = b_i$$
$$\vdots$$
$$t_{n1}x_1 + t_{n2}x_2 + \ldots\ldots + t_{ni}x_i + \ldots\ldots + t_{nn}x_n = b_n$$

$$\begin{bmatrix} t_{11} & & & & & & \\ t_{21} & t_{22} & & & & & \\ t_{31} & t_{32} & t_{33} & & & & \\ & & \vdots & & & & \\ t_{i1} & t_{i2} & t_{i3} & \ldots & t_{ii} & & \\ & & \vdots & & & & \\ t_{n1} & t_{n2} & t_{n3} & \ldots & t_{ni} & \ldots & t_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

Quedando este sistema matricialmente cómo sigue:

$$\begin{bmatrix} T_1 & T_2 & T_3 & \cdots & T_i & \cdots & T_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_i \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdot \\ \cdot \\ \cdot \\ b_i \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{bmatrix}$$

Efectuando los productos se tiene:

$\sum_{j=1}^{n} T_j x_j = b \Rightarrow \sum_{j \neq i} x_j T_j + x_i T_i = b$, particularizando para i, se tiene:

$$x_i T_i(i) = b(i) - \sum_{j \neq i} x_j T_j(i)$$

Debido a que la columna T_j tiene su i-ésima entrada nula para $j \geq i+1$, se tiene :

$$x_i T_i(i) = b(i) - \sum_{j=1}^{i-1} x_j T_i(i)$$

Si llamamos c_i al vector $b(i) - \sum_{j=1}^{i-1} x_j T_j(i)$, se tiene:

$$x_i T_i(i) = x_i t_{ii} = c_i(i) \Rightarrow x_i = \frac{c_i(i)}{t_{ii}}$$

Los vectores c_i verifican lo siguiente:

$c_1 = b$

$c_i = c_i - x_i T_i$ para $i \leq 1$

Dado que $c_i = b - \sum_{j=1}^{i-1} x_j T_j$ resulta que :

$$c_2 = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix} - x_1 \begin{bmatrix} t_{11} \\ t_{21} \\ . \\ . \\ t_{n1} \end{bmatrix}$$

$$c_3 = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix} - x_1 \begin{bmatrix} t_{11} \\ t_{21} \\ . \\ . \\ t_{n1} \end{bmatrix} - x_2 \begin{bmatrix} 0 \\ t_{22} \\ . \\ . \\ t_{n2} \end{bmatrix}$$

Lo cual permite las siguientes ecuaciones:

$$\text{Para } i = 1, 2, \ldots, n, \quad x_i = \frac{b_i}{t_{ii}}$$

$$\begin{bmatrix} b_{i+1} \\ . \\ . \\ . \\ b_n \end{bmatrix} \leftarrow \begin{bmatrix} b_{i+1} \\ . \\ . \\ . \\ b_n \end{bmatrix} - x_i \begin{bmatrix} t_{i+1\,i} \\ . \\ . \\ . \\ t_{ni} \end{bmatrix}$$

En el sistema $T x = b$, siendo T una matriz regular y triangular se tiene que el número de operaciones necesarias para hallar la solución x es $\sum_i \{\eta(T_{*_i}) / x_i \neq 0\}$. Si en el sistema $T x = b$ suponemos que el vector solución x está lleno se tiene que el numero de operaciones requeridas es $\eta(T)$ y este número de operaciones viene dado por $(n+1)n/2$ [2].

En el esquema del producto interno tenemos las relaciones de recurrencia:

$$\text{Para } j = 1,2,\ldots,n \qquad l_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 \right)^{1/2}$$

$$\text{Para } l = j+1, j+2, \ldots, n \qquad l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}\, l_{jk} \right) l_{jj}$$

En esta versión las columnas de L se calculan una por una ya que se trata de una factorización columna a columna, pero la parte de la matriz que quede sin factorizar no se aborda durante el procesamiento, y éste es el esquema llamado interno.

Si la matriz A es regular y tiene perfil simétrico, su factorización es A = L U ya que se considera la partición $A = \begin{pmatrix} d & w \\ v & \bar{H} \end{pmatrix}$

El número de no ceros de L es $\eta(L) = N + \sum_{i=1}^{N-1} \eta(n_i)$.

El número de operaciones requeridas para computar la factorización triangular L de la matriz A viene dada por : $\frac{1}{2}\sum_{i=1}^{N-1} \eta(n_i)[\eta(n_i)+3] = \frac{1}{2}\sum_{i=1}^{N-1}[\eta(L_{*i})-1][\eta(L_{*i})+2]$.

Los costes para la resolución de sistemas dispersos equivalentes con diferentes ordenamientos también pueden ser distintos.

Por lo tanto, al resolver un sistema de ecuaciones lineales A x = b, lo primero es hallar un buen ordenamiento a través de la matriz de permutación P, la cual incide en el sistema de la siguiente manera :

$$Ax = b \Rightarrow PAx = Pb \Rightarrow PAI = Pb \Rightarrow PAP^tPx = Pb \Rightarrow (PAP^t)Px = Pb,$$

resultando un sistema equivalente $\bar{A}\bar{x} = \bar{b}$ donde $A = \bar{P}AP^t$, siendo $\bar{x} = Px$ y $\bar{b} = Pb$.

Al nuevo sistema A x = b se le aplica la factorización de Cholesky,o sea factorizamos la matriz $PAP^t = LL^t$ que es también una matriz definida positiva. Al resolver el sistema permutado, a menudo se puede conseguir una reducción en el almacenamiento del ordenador y una reducción del tiempo de ejecución.

1.4 Objeto de la tesis.

Con este trabajo hemos pretendido hacer un estudio sobre el problema de resolver sistemas sparse, tanto en lo referente a la complejidad como a lo inherente a ciertos sistemas de estructuras de datos eficientes.

Hemos analizado algoritmos basados en la estructuración mediante teoría de grafos, haciendo comparaciones y modificaciones sobre el algoritmo ICM, en el sentido de mejorar el tiempo de CPU y en el de reducir el almacenamiento en el ordenador.

El objetivo fundamental no obstante, ha sido tratar de explotar el efecto fill-in al factorizar matrices sparse, para ello, basándonos en el algoritmo de grado mínimo, hemos hecho modificaciones que optimizan los tiempos de cálculo y la reducción de memoria, tanto en planteamientos generales como en algunos específicos.

Finalmente, hemos desarrollado aplicaciones de interés para la matemática, en el ámbito de la ciencia de la computación, así como para la ingeniería. Algunos ejemplos los hemos almacenados usando el esquema compacto.

Capítulo 2. Informatización de matrices dispersas mediante grafos.

2.0 Introducción.

En este capítulo consideramos algunas nociones básicas sobre teoría de grafos y establecemos una correspondencia con las matrices.

En particular la teoría de las matrices dispersas y los grafos son dos disciplinas con vínculos convenientemente aplicables.

La teoría de grafos aporta una adecuada herramienta para poder describir algoritmos y también caracterizar la estructura matricial. Los grafos son de gran utilidad para el estudio de matrices sparse.

El patrón de elementos distintos de cero de una matriz dispersa se puede representar mediante grafos, esto trae como consecuencia que muchos resultados de la teoría de grafos pueden aplicarse para obtener mejoras en las prestaciones numéricas de las matrices dispersas. La relación entre matrices sparse y grafos fue introducida por Parter, de ahí que el grafo de eliminación que veremos más adelante también se le llama grafo de Parter.

2.1 Nociones básicas sobre grafos.

Un grafo es una terna ordenada $G = (X, E, f)$, donde X es un conjunto finito no vacío de nodos o vértices, E es un conjunto de aristas y f es una aplicación que asocia a cada arista $a \in E$ un par no ordenado de nodos $\{x, y\}$ que son llamados extremos de la arista a.

Cabe la posibilidad de que entre dos nodos haya más de una arista, las cuales se llamarán aristas paralelas, también pueden existir aristas llamadas bucles cuyos extremos {x, y} coincidan.

Un grafo G = (X, E) se dice que es simple si no posee aristas paralelas ni bucles. Los grafos que se van a utilizar en esta tesis serán simples.

Un ordenamiento α de un grafo G es una aplicación de {1,2, 3, ... , n} sobre X, donde n es el índice del número de nodos o vértices de G. El grafo ordenado por α lo denotamos por $G^{\alpha} = (X^{\alpha} , E)$. En general el grafo G estará desordenado, a menos que lo hagamos constar.

Un grafo se puede asociar a cualquier matriz A. Si A es cuadrada y de orden n, y tiene estructura simbólica simétrica, con todos sus elementos diagonales distintos de cero, se puede definir el grafo asociado a la matriz como:

$$G^{A} = (X^{A} , E^{A})$$

Este grafo es aquel para el cual los n vértices de G^{A} están numerados desde 1 hasta n y $\{x_i , x_j \} \in E^{A} \Leftrightarrow a_{ij} = a_{ji} \neq 0$ e $i \neq j$. Aquí x_i indica el nodo de X^{A} con índice i [27].

La figura 2.1.0 muestra una matriz A y su grafo numerado asociado:

$$A = \begin{bmatrix} 1 & & & & x & & & x & \\ & 2 & & x & & & x & & \\ & & 3 & & x & & x & & x \\ & x & & 4 & & & x & & \\ & & x & & 5 & & & x & \\ x & & & & & 6 & & x & x \\ & & x & x & & & 7 & & x \\ & & & & x & & & 8 & x \\ & x & & & & x & & 9 & x \\ x & & & & & & & 10 & x \\ & & & & & x & x & x & x & 11 \end{bmatrix}$$

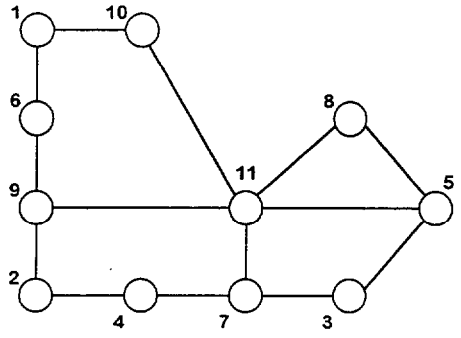


Fig 2.1.0.

Las x indican las entradas no nulas de A.

La suposición de que los elementos de la diagonal son distintos de cero hace que no sea necesario representar los bucles que unen cada nodo consigo mismo. En el caso de que la matriz A no sea simétrica se obtiene un digrafo.

El grafo asociado a una matriz simétrica permanece invariable salvo la enumeración de sus nodos, al aplicarle a la matriz una permutación simétrica (se le premultiplica y posmultiplica por una misma matriz de permutación P). Esta es una propiedad que hace de los grafos un buen instrumento para estudiar matrices dispersas.

Si $B = PAP^t$, los grafos asociados a B y A son idénticos salvo en lo que respecta a su numeración, o sea, para cualquier matriz de permutación $P \neq I$ de tamaño nxn, los grafos no ordenados de A y de $P\,A\,P^t$ son los mismos, pero los ordenamientos asociados son diferentes.

El grafo no ordenado de A representa la estructura que tiene A sin que tenga ningún ordenamiento en particular, representa la clase de equivalencia de la matriz $P\,A\,P^t$. Encontrar una buena permutación para A se puede considerar como encontrar un buen ordenamiento para su grafo.

Veamos en la figura 2.1.1 una matriz A, y su grafo G^A:

$$A = \begin{pmatrix} 1 & x & & & & \\ x & 2 & x & & & x \\ & x & 3 & x & & \\ & x & & 4 & x & \\ & & x & & 5 & x \\ x & & & & x & 6 \end{pmatrix}$$

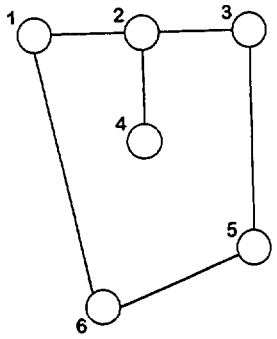


Fig 2.1.1.

Si realizamos dos ordenaciones distintas de la matriz A según las matrices de permutación P y Q, las matrices PAP^t y QAQ^t, así como sus grafos respectivos, quedan reflejados en las figuras 2.1.2 y 2.1.3.

$$PAP^t = \begin{pmatrix} 1 & & x & & x & \\ & 2 & & x & x & \\ x & & 3 & x & & \\ & x & x & 4 & & x \\ x & x & & & 5 & \\ & & & x & & 6 \end{pmatrix}$$

, cuyo grafo es :

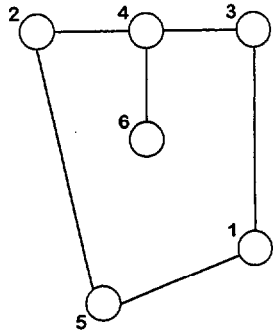


Fig 2.1.2.

$$QAQ^t = \begin{pmatrix} 1 & & x & & & \\ x & 2 & x & & & x \\ & x & 3 & x & & \\ & & x & 4 & x & \\ & & & x & 5 & x \\ & x & & & x & 6 \end{pmatrix}$$

cuyo grafo es :

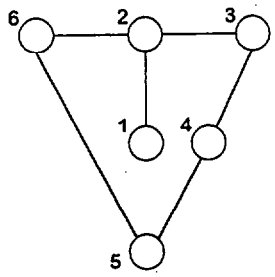


Fig 2.1.3.

Dos nodos x e y en G son adyacentes si $\{x, y\} \in E$, o sea, si hay alguna arista que los una.

Sea $Y \subset X$, el conjunto adyacente de Y, denotado por Adyc(Y), viene definido por: $Adyc(Y) = \{ x \in X\text{-}Y / \{x, y\} \in E \text{ para algún } y \in Y\}$.

El conjunto Adyc(Y) es el conjunto de nodos en G que, no perteneciendo a Y, son adyacentes a algún nodo de Y.

Sea la figuara 2.1.4 una matriz A y su grafo asociado:

$$A = \begin{pmatrix} 1 & x & & & x & & & x \\ x & 2 & & & & & x & \\ & & 3 & & x & x & x & \\ & & & 4 & & & x & x \\ x & & x & & 5 & & & \\ & & x & & & 6 & x & \\ & x & x & x & & x & 7 & \\ x & & & x & & & & 8 \end{pmatrix}$$

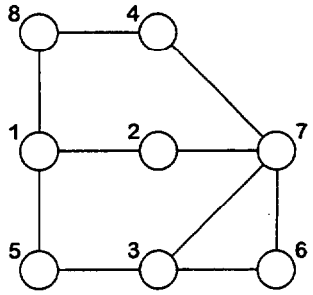


Fig 2.1.4.

Sea el conjunto Y = {1, 7}, luego, el conjunto Adyc(Y) = {2, 5, 8, 3, 6, 4} será el conjunto de nodos de G que no perteneciendo a Y, son adyacentes a algún nodo de Y.

Para Y ⊂ X, el grado de Y que llamaremos δ(Y) es el cardinal del adyacente de Y, o sea |Adyc (Y)| = 6. Cuando Y sea un conjunto unitario con un único nodo y, escribiremos δ(y).

Un subgrafo G' (X', E') es un grafo para el que X'⊂X y E'⊂E. Para Y⊂ X, la sección de grafo G(Y) o subgrafo generado por Y, es el grafo (Y, E(Y)), donde:

$$E(Y) = \{\{x, y\} \in E / x \in Y, y \in Y\}$$

En términos matriciales, la sección del grafo G(Y) es el grafo de una matriz que se obtiene suprimiendo todas las filas y columnas de la matriz G que no correspondan a elementos de Y.

Un subgrafo se dice que es un subgrafo sección cuando X' sólo contiene algunos nodos de G, y E' contiene todos los arcos {x, y} de G tales que x e y pertenecen a X'.

Sea el grafo de la figura 2.1.5:

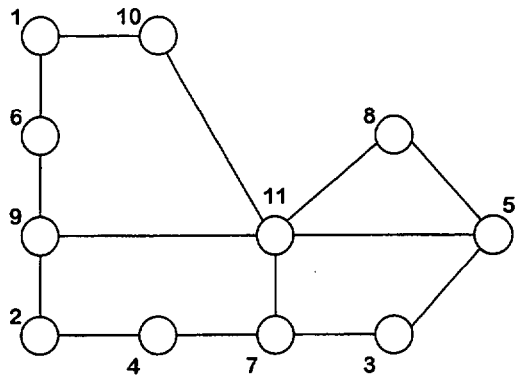


Fig 2.1.5.

Los nodos 3, 5, 7, 8, y 11 junto con los arcos (3, 5), (5, 8), (8, 11), (11, 7), (3, 7) y (3, 11) forman un subgrafo sección.

Se dice que una sección de grafo es una peña o cliqué si los nodos en el subgrafo son adyacentes dos a dos. En términos matriciales, una peña corresponde a una submatriz llena.

Un nuevo concepto que definiremos y que es muy importante es el de conexión en un grafo.

En primer lugar definimos la trayectoria entre dos nodos distintos x e y de G como un conjunto ordenado de nodos distintos $(v_1, v_2, \ldots, v_{i+1})$ tales que:

$$v_{i+1} \in Adyc(v_i),\ i = 1, 2, \ldots, l \text{ con } v_1 = x \text{ y } v_{l+1} = y.$$

Un grafo se dice que es conexo si para cada par de nodos diferentes existe por lo menos una trayectoria que los conecta. Un grafo G es inconexo o desconexo si consta de dos o más componentes conexas, o sea un grafo inconexo está formado por varias componentes conexas. Sea el grafo G=(X, E) y una relación binaria R, definida en X de forma que: " xRy $\Leftrightarrow$ existe un camino en X que conecta x con y ", dicha relación es de equivalencia y a las clases definidas por dicha relación las llamaremos componentes conexas. Dichas componentes conexas determinan una partición en X. Si dichas componentes conexas las llamamos C_i, entonces el subgrafo $G(C_i)$

= $(C_i,E(C_i))$ es un grafo conexo. Todo grafo puede particionarse en componentes conexas. Si G consta de k componentes conexas, entonces la matriz correspondiente será de bloques diagonales siendo cada bloque una componente conexa

En términos matriciales, habría que aclarar que si G es inconexo y consta de k componentes conexas y cada componente está ordenada consecutivamente, la matriz correspondiente será de bloques diagonales correspondiendo cada bloque diagonal a una componente conexa. Diremos que una matriz A es reducible si existe una matriz de permutación P tal que $PAP^t = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}$, siendo A_{11} y A_{22} submatrices cuadradas de órdenes p y n-p respectivamente [22].

En caso contrario se dice que A es irreducible, y en términos de grafos, A es irreducible si y solo si su grafo asociado es conexo.

Por último, el conjunto $Y \subset X$ es un separador de un grafo conexo G si la sección del grafo G(X-Y) es desconexa, o sea, un separador es un conjunto de nodos tal que, quitando los nodos que pertenecen a él y los arcos a ellos unidos en un grafo conexo o componente conexa, resulte un grafo no conexo.

Sea la figura 2.1.6 una matriz A y su grafo correspondiente:

$$A = \begin{bmatrix} 1 & & & x & x & & \\ & 2 & & & & & \\ & x & 3 & x & x & x & \\ x & x & x & 4 & & & x \\ x & & x & & 5 & x & \\ & & x & & x & 6 & x \\ & & & x & & x & 7 \end{bmatrix}$$

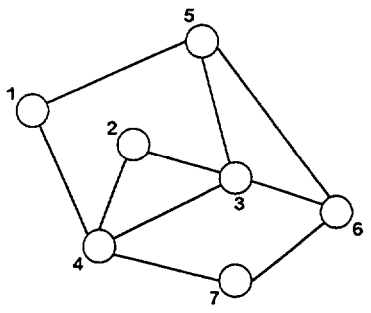


Fig. 2.1.6.

El conjunto $Y=\{x_3, x_4, x_5\}$, es un separador de este grafo puesto que G (X-Y) tiene tres componentes conexas cuyos conjuntos de nodos son $\{x_1\}$, $\{x_2\}$ y $\{x_6,x_7\}$.

Un separador es mínimo si cualquier subconjunto de el no es un separador. En el grafo de la figura 2.1.5, se tiene que el conjunto de nodos formado por el 7 y el 11 es un separador mínimo, al quitar estos nodos del grafo, resultan las componentes conexas {3, 5, 8} y {10, 1, 6, 9, 2, 4}.

Un grafo conexo que no tiene ciclos se le denomina árbol. Los árboles juegan un papel muy importante en el contexto de matrices dispersas, pues una matriz cuyo grafo asociado es un árbol se puede reordenar de tal forma que al factorizarla mediante eliminación de Gauss, no experimenta ningún relleno.

2.2 Representación de grafos en el ordenador.

En general cuando ejecutamos algoritmos sobre grafos, aquellos son bastantes sensibles a la forma en que los grafos están representados. Para nosotros la operación básica empleada es la de componer las relaciones de adyacencia entre nodos, por lo tanto necesitamos una representación que proporcione las propiedades de adyacencia del grafo y que sea económico en lo referente al almacenamiento [26].

Sea G = (X, E) un grafo con n nodos. Un lista de adyacencia para un nodo $x \in X$ es una lista que contiene todos los nodos de Adyc(x). La estructura de adyacencia para G es el conjunto de las listas de adyacencia para todos los $x \in X$. Tal estructura se puede implementar en el ordenador de una forma bastante simple y con bajo coste computacional, almacenando las listas de adyacencia secuencialmente en un vector unidimensional ADYACI junto con un vector de índices XADJ de longitud n+1.

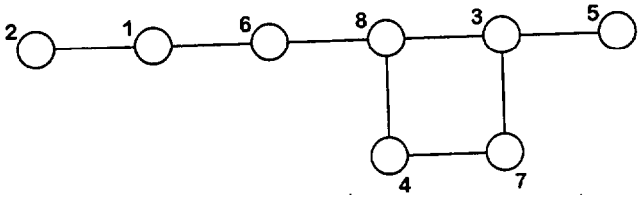
A menudo al objeto de programación resulta conveniente tener una entrada extra en XADJ tal que XADJ(n+1) señale a la siguiente localización de almacenamiento disponible en ADYACI.

❑ **Ejemplo.** Sea la matriz de la figura 2.1.7.

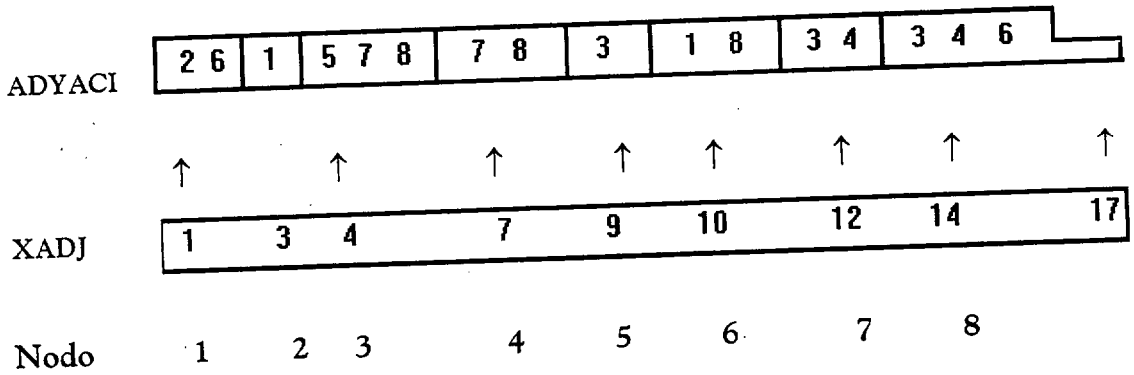
$$A = \begin{pmatrix} 1 & & & & & & & \\ x & 2 & & & & & & \\ & & 3 & & & & & \\ & & & 4 & & & & \\ & & x & & 5 & & & \\ x & & & & & 6 & & \\ & & x & x & & & 7 & \\ & & x & x & & x & & 8 \end{pmatrix}$$

Fig:2.1.7

Su grafo asociado es:



La estructura de ADYACI para este grafo es:



Con la finalidad de examinar todos los vecinos de un nodo se puede utilizar el siguiente segmento de programa:

```
JCOMZO = XADY(NODO)
JSTOP = XADY(NODO + 1)-1
IF (JSTOP.LT.JCOMZO) GOTO 400
   DO 300 J = JCOMZO, JSTOP
      NODVEC = ADYACI(J)
                .
                .
                .
300        CONTINUE
400  CONTINUE
                .
                .
                .
```

El último elemento de XADJ, 17, indica el final del vector ADYACI.

La demanda total de almacenamiento es $|X| + 2|E| + 1$ ya que:

ADYACI es un vector de dimensión $2|E|$

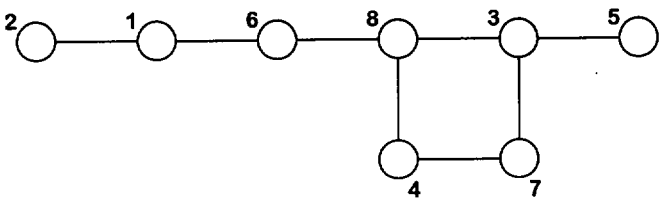
XADJ es un vector de dimensión $|X| + 1$

También existen otros esquemas de almacenamiento y uno de ellos es muy común ya que utiliza una simple tabla de conexiones, que tiene n filas y m columnas, donde definimos $m = \max \{ \delta(x) / x \in X \}$.

La lista de adyacencia para el nodo i se almacena en la fila i.

Si un número de nodos tiene grados menores que m, este tipo de almacenamiento puede resultar ineficiente.

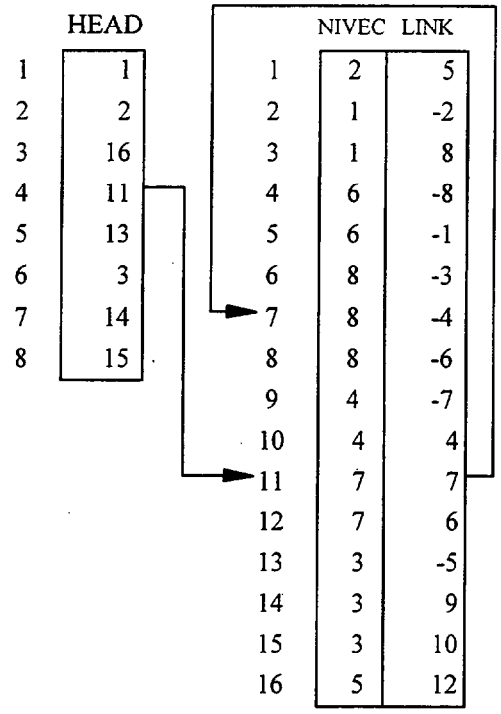
Sea el grafo siguiente:



y su tabla de conexión es:

nodos	vecinos	nodos	vecinos
1	2 6 -	5	3 - -
2	1 - -	6	1 8 -
3	5 7 8	7	3 4 -
4	7 8 -	8	3 4 6

mediante - se indican las posiciones no utilizadas de la tabla.



	HEAD
1	1
2	2
3	16
4	11
5	13
6	3
7	14
8	15

	NIVEC	LINK
1	2	5
2	1	-2
3	1	8
4	6	-8
5	6	-1
6	8	-3
7	8	-4
8	8	-6
9	4	-7
10	4	4
11	7	7
12	7	6
13	3	-5
14	3	9
15	3	10
16	5	12

Los dos esquemas mencionados tienen una clara desventaja. A menos que los grados de los nodos se conozcan a priori, es difícil construir el esquema de almacenamiento cuando el grafo viene dado como una lista de aristas, ya que no conocemos el tamaño definitivo de las listas de adyacencia.

Esta dificultad es superable si se introduce un conjunto de enlaces. El indicador HEAD (i) comienza la lista de adyacencia para el nodo i, donde NVEC contiene un vecino del nodo i y LINK es el indicador de la localización del siguiente nodo vecino de i. Por ejemplo, para recuperar los vecinos del nodo 4, recobramos

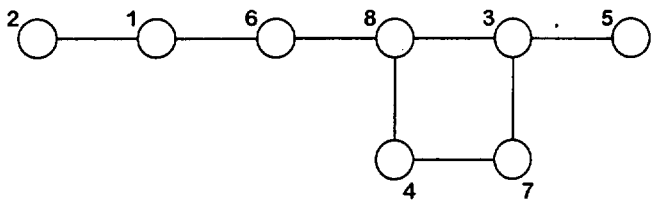
HEAD (4), que es 11. Ahora hallamos NVEC (11), que es 7, suponiendo que el vecino siguiente del nodo 4 es NVEC (7), que es 8.

Finalmente vemos que LINK (7)= -4, que nos indica el final de adyacencia para el nodo 4, en general un LINK negativo de -i indica el final de la lista de adyacencia para el nodo i. La demanda de almacenamiento para este tipo de representación es | X | + 4 | E |, ya que cada arista requiere dos puestos de memoria en NVEC y otros dos en LINK, esto es 2 | E | + 2 | E | = 4 | E | más | X | en HEAD, con lo cual tenemos que la demanda total es | X | + 4 | E |. Si hay espacio suficiente en los vectores NVEC y LINK se pueden añadir dos nuevas aristas.

Por ejemplo si se quiere añadir la arista {3, 6} a la estructura de adyacencia hay que añadir dos nuevas entradas en los vectores NVEC y LINK y luego ajustamos la lista de adyacencia del nodo 3 asignando a LINK(17) el 16, NVEC(17)=6 y a HEAD(3) el 17. La lista de adyacencia del nodo 6 se cambiaría de forma similar situando LINK(18) en 3, NVEC(18) en 3, y HEAD(6) en 18.

Si queremos disminuir el almacenamiento del vector ADYACI, podemos aplicar el procedimiento de almacenar solo aquellos nodos adyacentes que sean posteriores a él.

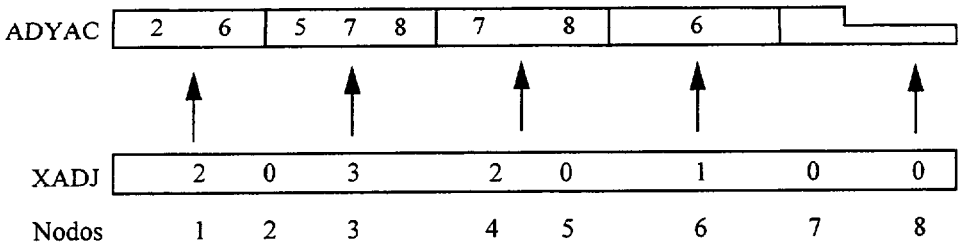
Sea el grafo anterior :



Tomando los elementos no nulos por columnas, por ejemplo XADJ(1)=2, esto indica que en la primera columna hay dos elementos no nulos, XADJ(2)=0, indica que en la segunda columna no hay elementos no nulos: son todos ceros. De esta manera mirando el vector XADJ vemos que hay siempre 2 columnas nulas.

El último cero se ha mantenido para que la dimensión del vector XADJ sea el número de columnas de la matriz. De esta forma se ha reducido el vector ADYACI a

la mitad, ya que no repetimos los nodos almacenados en las columnas anteriores. Por lo tanto. La demanda de almacenamiento total para este esquema es de | X |+| E |, ya que las aristas no se cuentan dos veces.

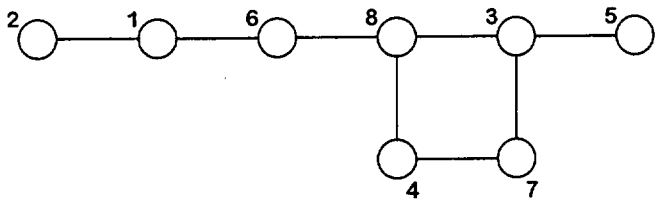


Esta modificación de los vectores lleva consigo una complicación a la hora de leer los nodos vecinos, sobre todo cuando la matriz es muy grande o deja de ser muy dispersa.

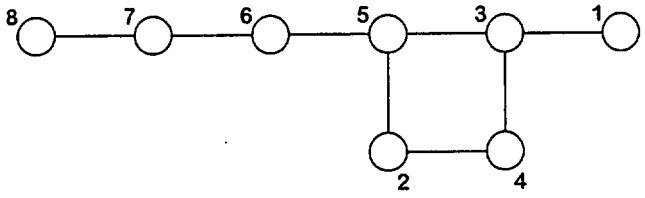
Con la finalidad de examinar los grados de los nodos se utiliza el siguiente segmento de programa:

```
                .
                .
                .
        LL=0
        DO J =1,NUMECU
                IF(J:EQ:1)THEN
                        GRAD(J)=RXADJ(1)
                        LL=GRAD(J)
                END IF
                LK=0
                DOI=1,LL
                        IF (RADYACI(I).NE.J) GO TO 200
                                LK=LK+1
200                             GRAD(J)=LK + RXADJ(J)
                END DO
                LL=LL + RXADJ(J)
        END DO
                .
                .
                .
```

Por otra parte, cuando renumeramos los nodos de un grafo, utilizamos los vectores PERM e INVP para seleccionar los ordenamientos nuevo y original.Sea el grafo anterior:



PERM ↑ ↓ INVP



El grafo G se almacena mediante el par de vectores ADYACI y XADJ que determinan su ordenamiento inicial y nuevo ordenamiento.

PERM(i) = k (N° del nodo inicial)

↑

N° de k en el nuevo ordenamiento

Por tanto INVP (PERM(i)) = i ⇒ INVP (k) =i

2.3 Subrutinas que operan en los grafos.

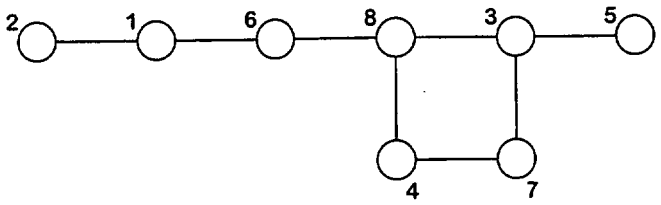
En las distintas subrutinas, el grafo G = (X, E) se almacena empleando un par de vectores como son XADJ y ADYACI.

Los vectores ADYACI y XADJ están referidos a su ordenamiento original. Cuando una subrutina encuentra un nuevo ordenamiento de ésta, se almacena en un

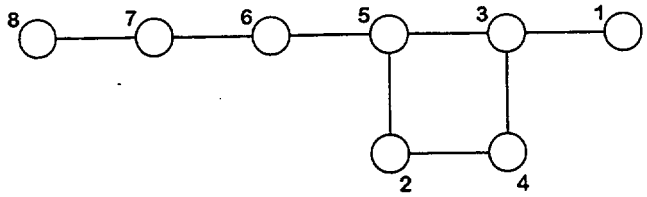
vector PERM donde PERM(i) = k significa que el número del nodo original k es el nodo iésimo en el nuevo ordenamiento.

A menudo también se emplea un vector de permutación afín INVP de longitud n (permutación inversa) que satisface INVP(PERM(i)) = i, es decir, INVP(k) de la posición de PERM donde reside el nodo originalmente numerado k.

❑ **Ejemplo.** Los vectores PERM e INVP del ejemplo anterior:



PERM ↑ ↓ INVP



PERM(1) = 5	INVP(1) = 7
PERM(2) = 7	INVP(2) = 8
PERM(3) = 3	INVP(3) = 3
PERM(4) = 4	INVP(4) = 4
PERM(5) = 8	INVP(5) = 1
PERM(6) = 6	INVP(6) = 6
PERM(7) = 1	INVP(7) = 2
PERM(8) = 2	INVP(8) = 5

$$PERM = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 7 & 3 & 4 & 8 & 6 & 1 & 2 \end{pmatrix}$$

$$INVP = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 7 & 8 & 3 & 4 & 1 & 6 & 2 & 5 \end{pmatrix}$$

A veces se tienen que ejecutar operaciones sólo en parte del grafo G y para implementar estas operaciones muchas de las subrutinas tienen un vector de enteros llamado MASC que es de longitud n que se emplea para ordenar tal grafo. Sólo tienen en cuenta los nodos i para los que MASC(i) $\neq$ 0. También alguna de las subrutinas empleadas tienen un Nº de nodo escogido llamado RAIZ como argumento, con MASC(RAIZ) $\neq$ 0.

La combinación RAIZ y MASC determinan el subgrafo de G a procesar.

Los parámetros utilizados en las subrutinas junto con sus contenidos aparecen listados de la manera siguiente:

❶ **(ADYACI , XADJ):** es el par de vectores enteros que almacena el grafo en su ordenamiento inicial. Las designaciones originales de los nodos adyacentes al nodo i se encuentran en:

$$ADYACI(k),\ XADJ(i) \leq k < XADJ(i+1),\ \text{con}\ XADJ(n+1) = 2|E| + 1.$$

❷ **PERM:** es un vector de enteros de longitud n que contiene al nuevo ordenamiento.

❸ **INVP:** es un vector de enteros de longitud n que contiene la inversa de la permutación.

❹ **MASC:** es un vector de enteros de longitud n utilizado para determinar un grafo parcial de G.Las subrutinas ignoran los nodos para los que MASC(i)=0.

❺ **RAIZ:** es un número de nodo para el que MASC(RAIZ) $\neq$ 0.La subrutina opera normalmente sobre la componente del subgrafo especificada por MASC que contiene al nodo RAIZ.

Capítulo 3. Almacenamiento y ordenamiento de matrices dispersas asociado a sistemas de ecuaciones lineales.

3.0 Introducción.

En este capítulo consideramos varias alternativas para almacenar matrices dispersas que correspondan a sistemas dispersos.

La efectividad del trabajo con matrices dispersas se mide no sólo en términos de los algoritmos que les manipulan sino también en la forma en la que el ordenador se integra dentro del proceso que generan estos algoritmos.

Entonces es lógico pensar que cuanto más eficaz sea el esquema según el cual se almacenan las matrices dispersas en un ordenador y cuanto más ágilmente se pueda recuperar la información relativa a los mismos, mejores serán los resultados que se obtienen cuando se manipulan los algoritmos.

El objetivo fundamental es el reordenamiento de la matriz, explotando su dispersión y controlando el efecto fill-in.

Dentro de las alternativas para el almacenamiento de matrices dispersas veremos el método de la banda y sus variantes así como el método de la envolvente o del perfil, uno de los cuales es utilizado en esta tesis. Tambien existen otros esquemas de almacenamiento para matrices que no tengam ninguna estructura especial como veremos más adelante.

3.1 Método de la banda y su coste computacional.

El método de la banda lo describimos a modo introductorio a fin de resaltar la más adecuada elección del método de almacenamiento en perfil o envolvente.

Sea A una matriz simétrica definida positiva de orden n×n, con entradas a_{ij}. Para la fila i-ésima de A, i=1, 2, ..., n definimos:

$$f_i(A) = m_i\{ j / a_{ij} \neq 0 \} \text{ y } \beta_i = i - f_i(A).$$

El número $f_i(A)$, es el subíndice columna de la primera entrada no nula en la fila i de A. El número $\beta_i(A)$ es la i-ésima anchura de banda de A. Como las entradas diagonales a_{ii} son positivas, se tiene que $f_i \leq i$, por lo tanto $\beta_i \geq 0$.

Definimos la anchura de banda de A por :

$$\beta(A) = \text{Max}\{ \beta_i(A) \ / \ 1 \leq i \leq n \} = \text{Max}\{ |i - j| / a_{ij} \neq 0 \}.$$

Se define la banda de A como:

$\text{Band}(A) = \{ \{i,j\} \ / \ 0 < i - j \leq \beta(A) \}$, que es la zona de A localizada a una distancia de la diagonal principal igual a $\beta(A)$ [24].

En el ejemplo siguiente tenemos $\beta(A)$, así como $\beta_i(A)$ y $f_i(A)$.

$$A = \begin{bmatrix} 1 & x & & x & & & \\ x & 2 & & & & & \\ 0 & 0 & 3 & & x & x & \\ x & 0 & 0 & 4 & x & & \\ & 0 & x & x & 5 & & x \\ & & x & 0 & 0 & 6 & x \\ & & & 0 & x & x & 7 \end{bmatrix}$$

$\beta(A)$

i	$f_i(A)$	$\beta_i(A)$
1	1	0
2	1	1
3	3	0
4	1	3
5	3	2
6	3	3
7	5	2

Cuando empleamos el método de la banda, los ceros que están fuera de la banda Band(A) se ignoran; los ceros dentro de la banda son almacenados. Esta explotación de ceros es posible en la resolución directa porque Band(A)= Band(L+L^t).

Un método común para almacenar en banda una matriz simétrica A, es el llamado esquema de almacenamiento diagonal. Las $\beta(A)$ subdiagonales del triángulo inferior de A que comprende Band(A) y la diagonal principal de A son almacenadas como columnas en una matriz de tamaño $n(\beta(A)+1)$. Como ejemplo tenemos la matriz de la figura 3.1.0:

$$A=\begin{pmatrix} 1 & x & & x & & & \\ x & 2 & & & & & \\ & & 3 & & x & x & \\ x & & & 4 & x & & \\ & & x & x & 5 & & x \\ & & x & & & 6 & x \\ & & & & x & x & 7 \end{pmatrix}$$

Fig 3.1.0

La matriz de almacenamiento es:

$$\begin{pmatrix} & & & a_{11} \\ & & a_{21} & a_{22} \\ & 0 & 0 & a_{33} \\ a_{41} & 0 & 0 & a_{44} \\ 0 & a_{53} & a_{54} & a_{55} \\ a_{63} & 0 & 0 & a_{66} \\ 0 & a_{75} & a_{76} & a_{77} \end{pmatrix}$$

y se almacenan por filas en la forma: a_{11} a_{21} a_{22} a_{31} a_{32} a_{33} a_{41} a_{42} a_{43} a_{44} a_{52} a_{53} a_{54} a_{55} a_{63} a_{64} a_{65} a_{66} a_{74} a_{75} a_{76} a_{77}.

Otra forma sería la de almacenar la filas completas de la matriz anterior, resultando igual el número de almacenamiento para cada fila:

$$0\ 0\ 0\ a_{11}\ 0\ 0\ a_{21}\ a_{22} \cdots$$

Este esquema de almacenamiento diagonal es muy simple y eficaz siempre y cuando $\beta_i(A)$ no varíe demasiado con i pues entonces serían almacenados un número excesivo de entradas nulas.

El número de operaciones requeridas para factorizar la matriz A, con una anchura de banda β, suponiendo que Band$(L+L^t)$ esté llena, es:

$\beta\ (\beta + 3)\ n / 2 - \beta / 3 - \beta - 2\ \beta / 3$ y el numero de operaciones que se necesitan para resolver el sistema lineal $A\ x = b$, dado el factor de Cholesky L de A es :

$2\ (\beta + 1)\ n - \beta\ (\beta + 1) = (\beta + 1)\ (2\ n - \beta)$.

❑ *Almacenamiento con ancho de banda variable (Jennings)*

Esta forma de almacenamiento es semejante a la anterior excepto que el ancho de banda varía con cada fila [29].

Si queremos almacenar la matriz:

$$\begin{pmatrix} a_{11} & & & & & & & \\ & a_{22} & & & & & & \\ a_{31} & a_{32} & a_{33} & & & & & \\ & a_{42} & & a_{44} & & & & \\ & & a_{53} & a_{54} & a_{55} & & & \\ & & & & a_{65} & a_{66} & & \\ & & & & & a_{76} & a_{77} & \\ & & & & & & a_{87} & a_{88} \end{pmatrix}$$

almacenamos los elementos en el vector:

$a_{11}\ a_{22}\ a_{31}\ a_{32}\ a_{33}\ a_{42}\ 0\ a_{44}\ a_{53}\ a_{54}\ a_{55}\ a_{65}\ a_{66}\ a_{76}\ a_{77}\ a_{87}\ a_{88} \cdots$

Es necesario emplear un vector auxiliar para conocer la posición de cada elemento en su fila. Jennings propone indicar la posición de los elementos de la diagonal principal:

$$AD = \{1, 2, 5, 8, 11, 13, 15, 17\}$$

y cualquier elemento a_{ij} puede ser localizado como AD(i) +j - i.

Por ejemplo:

$$a_{54} = AD(5) + 4 - 5 = 11 + 4 - 5 = 10$$

y el número de elementos almacenados en la fila i de la matriz será: AD(i) - AD(i-1).

Por ejemplo en la fila 5 será: AD(5) - AD(4) = 11 - 8 = 3.

3.2 Método de la envoltura y su coste computacional.

Este método de almacenamiento es un poco más sofisticado que el método de la banda ya que aprovecha la variación de $\beta_i(A)$ con i.

La envoltura de A, denotada por Env(A), se define como:

$$Env(A) = \{\{i,j\} \ / \ i - j \leq \beta_i(A)\}.$$

En términos de subíndices columna $f_i(A)$, se tiene que:

$$Env(A) = \{\{i.j\} \ / \ f_i(A) \leq j < i\}.$$

El cardinal $|Env(A)|$ se denomina perfil de A o tamaño de la envoltura de A y viene dado por :

$$|Env(A)| = \sum_{i}^{n} \beta_i(A)$$

❑ **Sea la matriz de la figura 3.2.0.**

$$A = \begin{pmatrix} 1 & x & x & x & & & \\ x & 2 & & x & & & \\ & & 3 & & x & x & \\ & x & & 4 & & & \\ & & & x & 5 & x & x \\ & x & x & & & 6 & x \\ & & & & & & 7 \end{pmatrix}$$

Fig 3.2.0

La envolvente de esta matriz la forman los elementos inscritos en el polígono, o sea:

Env(A) = {(1,1), (1,2), (1,4), (2,1), (2,2), (2,3), (2,4), (3,3), (3,4), (3,5), (3,6), (4,2), (4,3), (4,4), (5,4), (5,5), (5,6), (5,7), (6,2), (6,3), (6,4), (6,5), (6,6), (6,7), (7,7)}

El esquema de almacenamiento por perfil guarda todos los elementos de la envolvente mediante tres vectores:

VAL = Contiene todos los elementos de la envolvente

IFA = Indica los índices f_i de cada fila i

IA = Es un vector de punteros de dimensión m + 1 que indica las posiciones en VAL del primer elemento no nulo de la fila que corresponde al orden del elemento en IA.

Si, por ejemplo, se quiere almacenar la matriz:

$$A = \begin{pmatrix} 1 & 0 & -2 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & -4 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 6 \end{pmatrix}$$

mediante su perfil o envolvente, los vectores que se tienen que definir son:

VECTOR	1	2	3	4	5	6	7	8	9	10	11	12
IFA	1	4	5	6	10	12						
IA	1	1	3	2	4	5						
VAL	1	0	-2	2	3	6	-4	0	4	3	1	6

En este esquema de almacenamiento con respecto al esquema fila/columna se ha reducido el número de posiciones de memora necesarias pues la dimensión del vector IFA es sensiblemente inferior a la de ICO de aquel.

Si la matriz dispersa que hay que almacenar con el esquema de la envolvente es simétrica, sólo será necesario guardar la parte triangular inferior incluida la diagonal principal.

La parte de un programa en Fortran que recupera una fila de una matriz A almacenada según éste esquema es el siguiente:

```
VEC=0
IN = IA(I)
IF = IA(I+1)-1
J=0
DO II=IN, IF
      VEC(IFA(I)+J) = VAL(II)
      J = J+1
END DO
```

Para una matriz simétrica o de perfil simétrico A se tienen que:

$Env(A) = Env(L+L^t)$ y $Env(A) \subset Band(A)$, puesto que :

$$\text{Env}(A) = \{ \{i,j\} \} / 0 < i - j \leq \beta_i(A)\}$$

$$\text{Band}(A) = \{ \{i,j\} 0 < i - j \leq \beta(A)\}$$

$$\beta(A) = Max\{\beta_i(A) / \ 1 \leq i \leq n\}$$

$$\text{Si}\{i,j\} \in Env(A) \Rightarrow 0 < i - j \leq \beta_i(A) \leq \beta(A)$$

$$\Rightarrow 0 < i - j \leq \beta(A) \Rightarrow \{i,j\} \in Band(A). \, Luego, Env(A) \subset Band(A)$$

Suponiendo que solo se exploten aquellos ceros fuera de la envoltura, vamos a determinar el coste aritmético de la ejecución de la solución directa. Para ello introducimos el concepto de frente.

Para una matriz A, la i-ésima amplitud de frente de A se define como:

$$w_i(A) = \left|\{k / k > i \wedge a_{kl} \neq 0 \text{ para algún } l \leq i\}\right|$$

$w_i(A)$ es el número de filas activas en el paso i-ésimo de la factorizacion, o sea es el número de filas de la envoltura de A que intersectan la columna y, luego $w_i(A) = \left|\{j > i / \{i,j\} \in \text{Env}(A)\}\right|$

El número $w_i(A) = \text{Max}\{w_i(A) / 1 \leq i \leq n\}$ se denomina amplitud de frente de A.Sea la matriz de la figura 3.2.1:

$$Sea\ A = \begin{pmatrix} 1 & x & & x & & & \\ x & 2 & & \oplus & & & \\ 0 & 0 & 3 & & x & x & \\ x & \oplus & 0 & 4 & x & & \\ & 0 & x & x & 5 & \oplus & x \\ & & x & 0 & \oplus & 6 & x \\ & & & 0 & x & x & 7 \end{pmatrix}$$

Fig 3.2.1

En la matriz anterior se tiene :

i	$w_i(A)$	$\beta_i(A)$
1	2	0
2	1	1
3	3	0
4	2	3
5	2	3
6	1	3
7	0	2

Si se explotan únicamente estos ceros fuera de la envoltura, el número de operaciones que se necesita para factorizar A en LL^t viene dado por :

$(1/2)\sum_{i=1}^{n} w_i(A)(w_i(A)+3)$ y el número de operaciones necesario para resolver el sistema A x = b, dada la factorizacion LL^t es : $2\sum_{i=1}^{n}(w_i(A)+1)$.

3.3 Otras formas de almacenamiento.

3.3.1 Almacenamiento por coordenadas.

Una forma eficaz de almacenar una matriz dispersa en un ordenador es mediante un conjunto ordenado de una terna (a_{ij}, i, j) donde $a_{ij} \neq 0$ [20].

Si queremos almacenar una matriz por coordenadas en FORTRAN lo podemos hacer mediante la definición de 3 vectores IFI, ICO, y VAL.

IFI e ICO pueden ser INTEGER y VAL puede ser REAL.

IFI= Indica el número de la fila donde está el elemento no nulo.

ICO= Indica el número de la columna donde está el elemento no nulo.

Si queremos almacenar la matriz de la figura 3.3.0.

$$A = \begin{pmatrix} 1 & 0 & 0 & -1 & 0 \\ 2 & 0 & -2 & 0 & 3 \\ 0 & -3 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 & 0 \\ 5 & 0 & -5 & 0 & 6 \end{pmatrix}$$

Fig 3.3. 0

El esquema para almacenarla sería:

VECTOR	1	2	3	4	5	6	7	8	9	10	11
IFI	1	1	2	2	2	3	4	4	5	5	5
ICO	1	4	1	3	5	2	2	4	1	3	5
VAL	1	-1	2	-2	3	-3	4	-4	5	-5	6

Este esquema de almacenamiento presenta un gran inconveniente, como es la dificultad de recuperar fácilmente un vector columna o fila de la matriz.

3.3.2 Almacenamiento por filas / columnas.

Esta forma de almacenar las matrices dispersas es una de las más usadas para matrices que no tengan ninguna estructura especial. Si queremos almacenar por filas (por columnas sería igual) se necesitan tres vectores: IA, ICO y VAL.

VAL = Contiene todos los elementos distintos de cero de la matriz.

ICO = Contiene los subíndices columna de los elementos de VAL

IA = Es un vector de posición (vector de punteros) que marca la posición en VAL e ICO del primer elemento no nulo de la fila que corresponde al orden del elemento que lee de IA.

Si consideramos de nuevo la matriz A de la figura 3.3.0 tenemos:

VECTOR	1	2	3	4	5	6	7	8	9	10	11
*IA	1	3	6	7	9	12					
ICO	1	4	1	3	5	2	2	4	1	3	5
VAL	1	-1	2	-2	3	-3	4	-4	5	-5	6

ICO = Indica el número de columna donde está

VAL = Son los valores de los elementos

La dimensión de IA es n+1 ya que es necesario definir el número de elementos no nulos de la última fila n.

La información relativa a la fila r de una matriz A estará en la posición IA(r) a IA(r+1)-1 de ICO y VAL excepto cuando IA(r+1) = IA(r) en cuyo caso la fila r estará vacía.

Este esquema de almacenamiento tiene una gran difilcutad ya que cuando un elemento se hace no cero a lo largo del proceso de manipulación de la matriz dispersa, se tiene que redefinir toda la estructura.

3.3.3 Almacenamiento de Gustavson.

Gustavson propone almacenar la matriz A con tres vectores AH, IH e IA. El vector AH contiene las entradas no nulas de A, fila a fila sin almacenar los ceros. El vector IH contiene el número de columna de cada entrada de AH.

El vector IA da la posición de la primera entrada de cada fila en AH de modo que la última entrada de IA es igual al número de elementos no nulos de A mas uno.

O Veamos un ejemplo de almacenamiento de Gustavson:

$$A = \begin{pmatrix} 3 & 0 & -4 & 1 & 0 \\ 0 & -2 & 3 & 0 & 0 \\ 0 & -1 & 4 & 0 & -1 \\ -2 & 0 & 0 & -3 & 0 \\ 0 & -5 & 6 & 0 & 2 \end{pmatrix}$$

AH = (3, -4, 1, -2, 3, -1, 4, -1, -2, -3, -5, 6, 2)
IH = (1, 3, 4, 2, 3, 2, 3, 5, 1, 4, 2, 3, 5)
IA = (1, 4, 6, 9, 11, 14)

A simple vista, este método no parece ventajoso: la matriz A tiene 25 entradas, los tres vectores AH, IH e IA tienen 32, sin embargo para una matriz sparse de 1000x1000 con 30 entradas no nulas por fila, habría que almacenar: 30000 + 30000 + 1001 = 61001 datos para los tres vectores AH, IH e IA en lugar de un millón de entradas, lo cual es una ventaja significativa.

3.3.4 Almacenamiento de Fletcher.

El almacenamiento de Fletcher conserva los vectores AH e IH, pero en lugar del vector IA introduce un nuevo vector JEND, que indica el número de entra-

das no nulas de cada fila, y una variable LCMAX que contiene el número de ceros de la matriz A.

En la matriz del ejemplo anterior tenemos que:

AH = (3, -4, 1, -2, 3,3 -1, 4, -1, -2, -3, -5, 6, 2)

IH = (1, 3, 4, 2, 3, 2, 3, 5, 1, 4, 2, 3, 5)

JEND = (3, 2, 3, 2, 3)

LCMAX = (13)

Será también necesario introducir otro vector B que almacenara los términos independientes y una variable N que indicara el número de ecuaciones a resolver.

- Si que remos almacenar la matriz de la figura 3.3.1:

$$A = \begin{pmatrix} 5 & -3 & 0 & 0 & 2 \\ 0 & 2 & 4 & -1 & 0 \\ -1 & 0 & -3 & 0 & 0 \\ 2 & 0 & 0 & -5 & 4 \\ 0 & 0 & 0 & 3 & 6 \end{pmatrix} \qquad B = \begin{pmatrix} 11 \\ 21 \\ 37 \\ 24 \\ 5 \end{pmatrix}$$

Fig 3.3.1

Luego, el esquema de almacenamiento es:

AH = (5, -3, 2, 2, 4, -1, -1, -3, 2, -5, 4, 3, 6)

IH = (1, 2, 5, 2, 3, 4, 1, 3, 1, 4, 5, 4, 5)

JEND = (3, 3, 2, 3, 2)

B = (11, 21, 37, 24, 5)

LCMAX = (13)

N = 5

3.4 Interpretación de los grafos.

Sea A una matriz simétrica de orden nxn y sea $G^A = (X^A , E^A)$ su grafo no dirigido asociado, siendo el conjunto de nodos $X^A = (x_1 , x_2 , ..., x_n)$.

Con el fin de proporcionar una visión del método de la envoltura, tenemos que dar una interpretación teórica de los grafos [18].

❑ Teorema: Para $i < j$, $\{i,j\} \in Env(A) \Leftrightarrow x_j \in Ady(\{x_1, x_2, ..., x_i\})$.

Demostración: Si $x_j \in Ady(\{x_1, x_2, ..., x_i\})$, entonces $a_{jk} \neq 0$ para algún $k \leq i$, de forma que $f_j(A) \leq i$ e $\{i,j\} \in Env(A)$.

Reciprocamente, si $f_j(A) \leq i < j$, esto significa que $x_j \in Ady(x_{f_j}(A))$, que implica $x_j \in Ady(\{x_1, x_2, ..., x_i\})$.

Corolario: Para $i=1, 2, ..., n$, $w_i (A) = |Adyc(\{x_1 , x_2 , ..., x_n \})|$ puesto que $w_i (A) = | j> i / \{i, j\} \in Env(A)\}|$ y según el teorema anterior, $\{i,j\} \in Ady(\{x_1, x_2, ..., x_i\})$, entonces se tiene que $w_i (A) = |Adyc(\{x_1 , x_2 , ..., x_n \})|$.

○ Ejemplo:

Sea la matriz de la figura 3.4.0.

$$A = \begin{pmatrix} 1 & x & & x & & & \\ x & 2 & & & & & \\ & & 3 & & x & x & \\ x & & & 4 & x & & \\ & & x & x & 5 & & x \\ & & x & & & 6 & x \\ & & & & x & x & 7 \end{pmatrix}$$

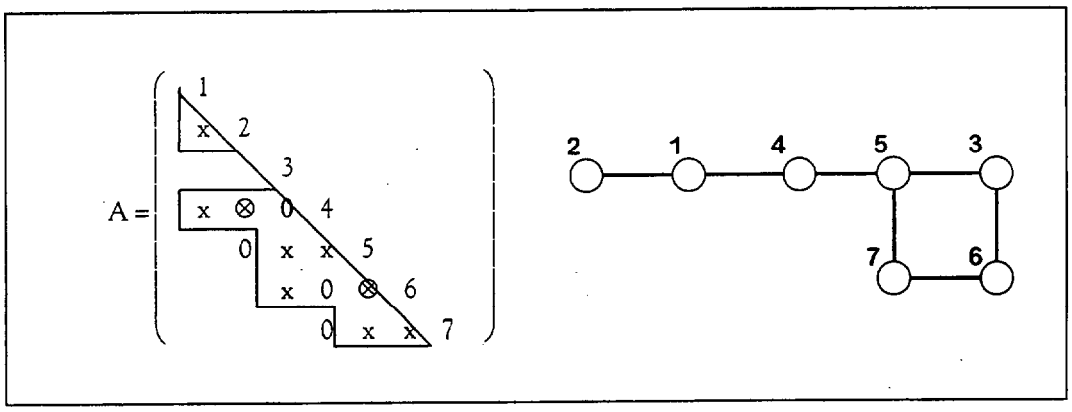
Fig 3.4.0.

Nos vamos a referir al conjunto $Adyc (\{x_1 , x_2 , ..., x_n \})$ como el frente i-ésimo del grafo ordenado a su cardinal como la i-ésima amplitud del frente.

En la matriz de la figura anterior se tiene que los conjuntos de adyacencia son:

$$\text{Adyc}(x_1) = \{x_2, x_4\}$$
$$\text{Adyc}(x_1, x_2) = \{x_4\}$$
$$\text{Adyc}(x_1, x_2, x_3) = \{x_4, x_5, x_6\}$$
$$\text{Adyc}(x_1, x_2, x_3, x_4) = \{x_5, x_6\}$$
$$\text{Adyc}(x_1, x_2, x_3, x_4, x_5) = \{x_6, x_7\}$$
$$\text{Adyc}(x_1, x_2, x_3, x_4, x_5, x_6) = \{x_7\}$$
$$\text{Adyc}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = \varnothing$$

luego, queda la matriz y su grafo asociado de la siguiente forma:



3.5 Ordenamiento de la envoltura.

❑ Algoritmo inverso de Cuthill McKee.

Es un algoritmo que está diseñado para reducir la envoltura de una matriz dispersa de perfil simétrico.

El esquema de CM reduce la amplitud de banda de una matriz a través de la minimización local de los β_i .

La idea en que se basa su estrategia este algoritmo es muy sencilla: como lo que se trata es que los elementos distintos de cero estén lo más cerca posible de la diagonal principal, una vez enumerado un nudo k, si se enumeran inmediatamente después los que están unidos a él que no han sido numerados previamente, se conseguiría que en la fila k ese objetivo se cumpla. El esquema de Cuthill-Mckee reduce la amplitud de banda de una matriz a través de una minimizacion local de los β_i y es empleado como método para reducir el perfil $\sum\beta_i$. La numeración dada por el algoritmo de Cuthill-Mckee conduce a un perfil monótono, siendo un perfil monótono aquel que verifica que :

$$\forall\ k \text{ y } l, \text{ donde } k > l, \text{ se verifica que } b_k \leq b_l .$$

❍ El algoritmo es el siguiente:

1. Construimos una tabla indicando el número de conexiones (que llamaremos grado y lo representamos por $\delta(x)$ de cada nodo)

2. Elegimos un nodo (en un extremo del grafo y con pocas conexiones)

3. Renumeramos los nodos conectados a x_1 en orden ascendente de grado.

Descripción del algoritmo :

Paso 1. Determinar un nodo inicial r y asignar $x_i \leftarrow r$.

Paso 2. Bucle principal.

Para i = 1,2,.....n, hallar todos los vecinos no enumerados del nodo x_i y numerarlos en orden ascendente de grado.

Paso 3. Ordenamiento inverso.

El ordenamiento inverso de Cuthill-Mckee está dado por $y_1, y_2, \ldots, y_n$ donde $y_i = x_{n-i+1}$ para $i = 1,2,\ldots n$, esto es, $y_1 = x_n$, $y_2 = x_{n-1}, \ldots, y_n = x_1$

En el caso de que el grafo G sea inconexo, podremos aplicar el algoritmo anterior para cada componente conexa del grafo.

○ Ejemplo:

Sea la figura 3.5.0 de la matriz A y su grafo asociado:

$$A = \begin{pmatrix} a & & & & & & & & & \\ x & b & & & & & & & & \\ & x & c & & & & & & & \\ & x & & d & & & & & & \\ & & x & x & e & & & & & \\ & & x & & & f & & & & \\ & & x & x & & & g & & & \\ & & & x & x & & & h & & \\ & & & & & & & x & i & \\ & & & & x & x & x & & x & j \end{pmatrix}$$

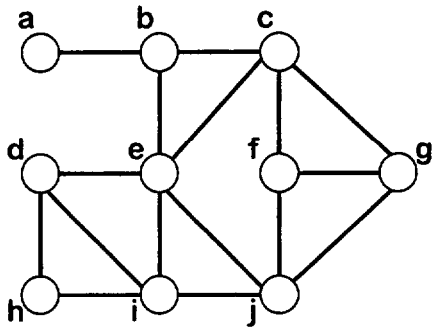


Fig 3.5.0

El grafo de la figura 3.5.0 es un grafo adireccional ya que la matriz A es simétrica.

Para aplicar el algoritmo de ordenamiento de CM, construimos una tabla con los nodos, listas de conexiones y el grado de cada nodo [2].

NODOS	LISTA CONEXIONES	GRADO
a	b	1
b	a,e,c	3
c	b,e,g	3
d	e,h,i	3
e	b,c,d,i,j	5
f	c,g,j	3
g	c,f,j	3
h	d,i	2
i	d,e,j,h	4
j	e,f,g,i	4

Es conveniente seleccionar un nodo inicial que llamaremos x_j en un extremo del grafo y que tenga pocas conexiones. (posteriormente veremos un algoritmo para la búsqueda de este nodo inicial).

Tomando como nodo inicial x_1 = a, el nodo vecino al nodo a es {b} (sólo tomamos nodos no enumerados).

$$\delta(b) = 3 \Rightarrow x_2 = b$$

Nodos vecinos a b no enumerados aún: $\{e,c\} \Rightarrow \begin{cases} \delta(c) = 3 \\ \delta(e) = 5 \end{cases} \Rightarrow \begin{cases} x_3 = c \\ x_4 = e \end{cases}$

Nodos vecinos a x_3 = c no enumerados aún {g} ; $\delta(g) = 3$, luego x_5 = g

Nodos vecinos a x_4 = e no enumerados aún {d,i,j}.

$$\delta(d) = 3 \qquad x_6 = d$$

$$\delta(i) = 4 \qquad x_7 = i$$

$$\delta(j) = 4 \qquad x_8 = j$$

Nodos vecinos a x_5 = g no enumerados aún: {f} $\Rightarrow$ $\delta(f) = 3 \Rightarrow x_9 = f$

Nodos vecinos a x_6 = d no enumerados aún: {h} $\Rightarrow$ $\delta(h) = 2 \Rightarrow x_{10} = h$

Nodos vecinos a x_7 = i no enumerados aún: $\varnothing$

Nodos vecinos a x_8 = j no enumerados aún: $\varnothing$

Nodos vecinos a x_9 = f no enumerados aún: $\varnothing$

Nodos vecinos a x_{10} = h no enumerados aún: $\varnothing$

Por lo tanto, la nueva ordenación de la matriz es : a, b, c, e, g, d, i, j, f, h y el nuevo perfil de la matriz de la figura 3.5.0 será:

$$\begin{pmatrix}
a & & & & & & & \\
x & b & & & & & & \\
 & x & c & & & & & \\
 & x & x & e & & & & \\
 & & x & & g & & & \\
 & & & x & & d & & \\
 & & & x & & x & i & \\
 & & & x & x & & x & j \\
 & & & & x & & & x & f \\
 & & & & & x & x & & & j
\end{pmatrix}$$

El ancho es 4, el número de elementos nulos en la envolvente es 7, el número de elementos de la envolvente es 33.

Si tomamos como nodo inicial, por ejemplo, x_1 = h, tenemos lo siguiente:

Nodos vecinos a h no enumerados aún: {d,i} $\Rightarrow \begin{cases}\delta(d)=3\\ \delta(i)=4\end{cases} \Rightarrow \begin{cases}x_2=d\\ x_3=i\end{cases}$

Nodos vecinos a x_2 = d no enumerados aún: {e} $\Rightarrow$ luego x_4 = e

Nodos vecinos a x_3 = i no enumerados aún: {j} $\Rightarrow$ luego x_5 = j

Nodos vecinos a x_4 = e no enumerados aún: {b,c} $\Rightarrow \begin{cases}\delta(b)=3\\ \delta(c)=4\end{cases} \Rightarrow \begin{cases}x_6=b\\ x_7=c\end{cases}$

Nodos vecinos a x_5 = j no enumerados aún: {f,g} $\Rightarrow \begin{cases}\delta(f)=3\\ \delta(g)=3\end{cases} \Rightarrow \begin{cases}x_8=f\\ x_9=g\end{cases}$

Nodos vecinos a x_6 = b no enumerados aún: {a} $\Rightarrow$ luego $\delta(a) = 1 \Rightarrow x_{10}$ = a

Nodos vecinos a x_7 = c no enumerados aún:$\varnothing$

Nodos vecinos a x_8 = f no enumerados aún: $\varnothing$

Nodos vecinos a x_9 = g no enumerados aún: $\varnothing$

Nodos vecinos a x_{10} = a no enumerados aún: $\varnothing$

La nueva ordenación es: h, d, i, e, j, b, c, f, g, a, siendo el nuevo perfil de la matriz del ejemplo anterior:

$$\begin{pmatrix} h & & & & & & & & \\ x & d & & & & & & & \\ x & x & i & & & & & & \\ & x & x & e & & & & & \\ & & x & x & j & & & & \\ & & & x & & b & & & \\ & & & & & x & c & & \\ & & & x & x & & & f & \\ & & & & x & & x & x & g \\ & & & & & x & & & a \end{pmatrix}$$

Observando el nuevo ordenamiento de la matriz vemos que el ancho de banda no varía, pero el número de ceros ha aumentado hasta ocho, quedando la envolvente con 33 elementos.

Debe destacarse que la selección del nodo inicial es muy importante. Si se toma como nodo inicial el nodo e, el ancho de banda sería de 6, el número de ceros en la envolvente sería 20 y el número total de elementos en la envolvente sería 46. Por lo tanto la elección del nodo de partida para comenzar la enumeración es una cuestión crítica para el resultado del algoritmo.

3.6 Algoritmo inverso de Cuthill-McKee.

George (1972) encontró que el ordenamiento obtenido al invertir el ordenamiento de Cuthill-Mckee resultaba frecuentemente mejor o igual que el ordenamiento original, en términos de reducción del perfil, aunque la anchura de banda permanecía sin mejorarse. A este ordenamiento se le llama ordenamiento inverso de Cuthill-McKee (ICM) [14].

El algoritmo inverso de Cuthill-Mckee reduce la envolvente de una matriz dispersa simétrica.

En el algoritmo inverso se desplaza al elemento (i, j) al lugar (n-j+1, n-i+1).

Si en el ejemplo anterior, donde la ordenación obtenida por Cuthill-Mckee fue: a, b, c, e, g, d, i, j, f, h , aplicamos el ordenamiento inverso, obtendremos la ordenación h, f, j, i, d, g, e, c, b, a, quedando el ordenamiento de la matriz de la siguiente manera:

$$\begin{pmatrix} h & & & & & & & & & \\ & f & & & & & & & & \\ x & x & j & & & & & & & \\ x & & x & i & & & & & & \\ & x & & x & d & & & & & \\ & x & x & & & g & & & & \\ & & & x & x & & e & & & \\ & & & & & x & x & c & & \\ & & & & & & x & x & b & \\ & & & & & & & & x & a \end{pmatrix}$$

Al aplicar el ICM se observa que el ancho de banda ha aumentado y el número de la envolvente permanece igual a 33.

3.7 Algoritmo para la búsqueda de un nodo inicial.

Para abordar el problema de determinar con cuál nodo se comienza el algoritmo de Cuthill-McKee, enunciamos los conceptos de excentricidad, diámetro y nodo periférico de un grafo.

Se define como trayectoria de longitud k desde un nodo x_0 a x_k como un conjunto ordenado de nodos distintos $(x_0 , x_i , ..., x_k)$ donde $x_i \in Adyc(x_{i+1})$ para $0 \leq i \leq k-1$.

La distancia d(x,y) entre 2 nodos x e y en el grafo conexo G = (X, E) es simplemente la longitud de la trayectoria más corta que une los nodos x e y.

Se define como excentricidad de un nodo por $\varepsilon(x) = \max\{d(x,y) / y \in X\}$

El diámetro de G queda fijado por:

$$\delta(G) = \max\{\varepsilon(x) / x \in X\} = \max\{d(x,y) / x,y \in X\}$$

Se dice que un nodo $x \in X$ es periférico si su excentricidad es igual al diámetro del grafo, es decir, si $\varepsilon(x) = \delta(G)$. La figura 3.7.0 muestra un grafo que tiene 8 nodos con un diámetro de 5, los nodos x_2, x_5 y x_7 son nodos periféricos ya que son de máxima exentricidad.

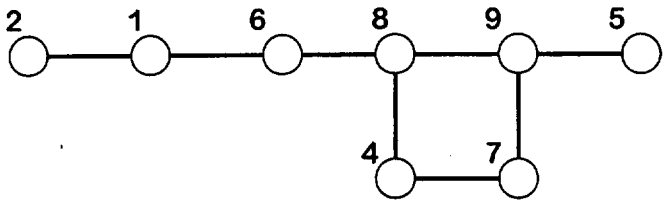


Fig. 3.7.0.

3.8 Algoritmo de George para la búsqueda de un nodo inicial.

Paso 1: Inicialización : Elegir un nodo arbitrario r de X

Paso 2: Generar una estructura con niveles. Se construye la estructura con niveles enraizada en r : $L(r) = \{L_0(r), L_1(r),..., L_{\varepsilon(r)}(r)\}$

Paso 3: Reducción en el último nivel: Elegir un nodo x de un mínimo grado en $L_{\varepsilon(r)}(r)$.

Paso 4: Generar una estructura con niveles.

a) Construir la estructura con niveles generada en x:

$$L(r) = \{L_0(r), L_1(r),..., L_{\varepsilon(r)}(r)\}$$

b) Si $\varepsilon(x) > \varepsilon(r)$, establecer $r \leftarrow x$ e ir al paso 3.

Paso 5: Conclusión: El nodo x es un nodo pseudo periférico.

Sea el grafo de la figura 3.8.0 en el cual se va a hallar un nodo pseudoperiférico:

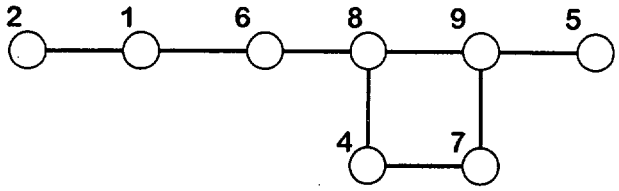
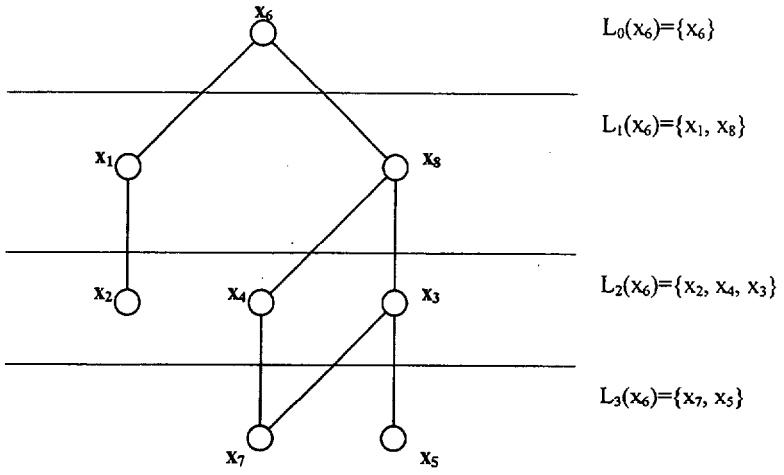
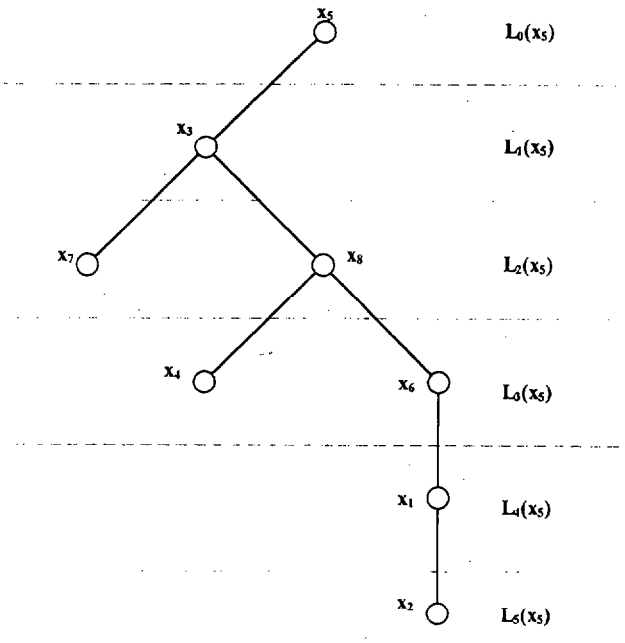
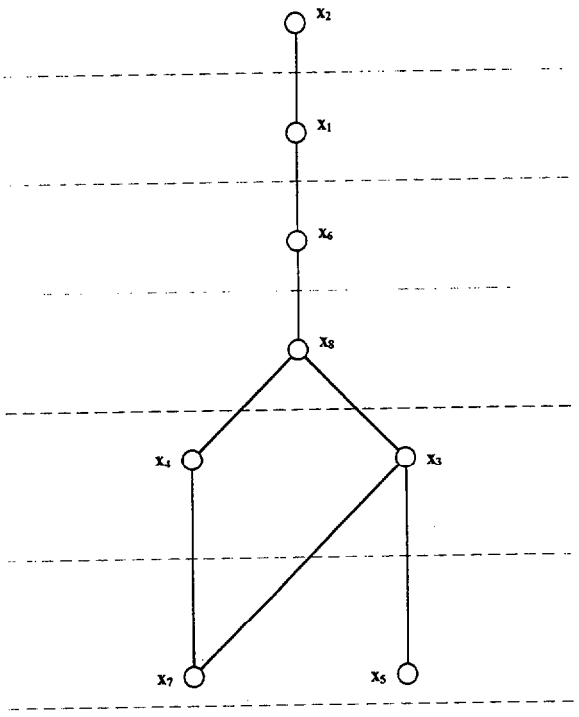


Fig 3.8.0.

Las distintas estructuras enraizadas de dicho grafo son las que se muestran a continuación:







Veamos en el grafo anterior como se genera el nodo pseudo-periférico:

Paso 1 : Inicialización.

Elegir un nodo arbitrario r de X.

Sea r = x_6

Paso 2 : Generar una estructura con niveles. Se construye la estructura con niveles enraizada en r: $L(r)=\{L_0(r),\ L_1(r),\ldots\ldots,L_{e(r)}\}$.

Ahora se halla $L(x_6)=\{L_0(x_6),L_1(x_6),L_2(x_6).....\}$.

$L_0(x_6)=\{x_6\}$.

$L_1(x_6)=\mathrm{Ady}(L_0(x_6))=\mathrm{Ady}(x_6)=\{x_1,x_8\}$.

$L_2(x_6)=\mathrm{Ady}(L_1(x_6))-L_0(x_6)=\mathrm{Ady}(\{x_1,x_8\})-\{x_6\}=\{x_2,x_3,x_4\}$.

$L_3(x_6)=\mathrm{Ady}(L_2(x_6))-L_1(x_6)=\mathrm{Ady}(\{x_2,x_3,x_4\})-\{x_1,x_8\}=$

$=\{x_1,x_5,x_7,x_8\}-\{x_1,x_8\}=\{x_5,x_7\}$.

$L_4(x_6)=\mathrm{Ady}(L_3(x_6)-L_2(x_6)=\mathrm{Ady}(\{x_5,x_7\})-\{x_5,x_7\}=\varnothing$.

Resultando de ésta manera la estructura con niveles.

Paso 3: Ultimo nivel de reducción.

Elegir un nodo x de mínimo grado en $L_{\varepsilon(r)}(r)$.

$d(x_5) = 1,\ d(x_7) = 2, \text{luego}, x = x_5$

Paso 4: Generar una estructura con niveles

a) Construir la estructura con niveles enraizada en x :

$L(x) = \{L_0(x), L_1(x), \ldots, L_{\varepsilon(x)}(x)\}$.

$L(x_5) = \{L_0(x_5), L_1(x_5), L_2(x_5), \ldots.\}$.

$L_0(x_5) = \{x_5\}$.

$L_1(x_5) = \text{Ady}(L_0(x_5)) = \{x_3\}$.

$L_2(x_5) = \text{Ady}(L_1(x_5)) - L_0(x_5) = \{x_7, x_8\}$.

$L_3(x_5) = \text{Ady}(L_2(x_5)) - L_1(x_5) = \{x_4, x_6\}$.

$L_4(x_5) = \text{Ady}(L_3(x_5)) - L_2(x_5) = \{x_1\}$.

$L_5(x_5) = \text{Ady}(L_4(x_5) - L_3(x_5) = \{x_2\}$.

De esta manera ha quedado calculada la estructura con niveles de la figura.

b) Si $\varepsilon(x) > \varepsilon(r)$, establecer $r \rightarrow x$, e ir al paso 3.

Dado que $\varepsilon(x_5) = 5 > \varepsilon(r) = 3$, se establece $r \leftarrow x_5$, siendo, pues, el nuevo nodo r igual a x_5.

Ir al paso 3.

Paso 3: Ultimo nivel de reducción.

El único nodo posible $x \in L_5(x_5)$ es $x = x_2$.

Paso 4: Generar una estructura con niveles.

a)Construir la estructura con niveles enraizada en x :

$L(x) = \{L_0(x), L_1(x), \ldots, L_{\varepsilon(x)}(x)\}$

$L(x_2) = \{L_0(x_2), L_1(x_2), L_2(x_2), \ldots.\}$.

$L_0(x_2) = \{x_2\}$.

$L_1(x_2) = Ady(L_0(x_2)) = \{x_1\}$.

$L_2(x_2) = Ady(L_1(x_2)) - L_0(x_2) = \{x_6\}$.

$L_3(x_2) = Ady(L_2(x_2)) - L_1(x_2) = \{x_8\}$.

$L_4(x_2) = Ady(L_3(x_2)) - L_2(x_2) = \{x_3, x_4\}$.

$L_5(x_2) = Ady(L_4(x_2) - L_3(x_2) = \{x_5, x_7\}$.

$L_6(x_2) = Ady(L_5(x_2)) - L_4(x_2) = \varnothing$.

De esta manera se ha hallado la estructura con niveles de la figura.

b) Si $e(x) > e(r)$, establecer $r \leftarrow x$, e ir al paso 3.

No se verifica que $e(x_2) = 5 > e(r) = 5$.

Paso 5: Conclusión

El nodo x es un nodo pseudo-periférico (i.e.,$x = x_2$ es el nodo pseudo-periferico).

3.9 Implementación del método de la envoltura.

El esquema de almacenamiento que utilizaremos es un esquema propuesto por Jennings, según este esquema para cada fila de la matriz se almacenan todas las entradas desde el primer nocero a la diagonal. Estas porciones de filas se almacenan en lugares contiguos de un vector, no obstante emplearemos una modificación en el que las entradas diagonales se almacenan en un vector aparte.

El esquema tiene un vector de almacenamiento principal ENV que contiene las entradas de la envoltura en cada fila, así como un vector índice auxiliar XENV de longitud n que es empleado para señalar el comienzo de cada porción de fila.

Además de los esquemas de almacenamiento expuestos existe una técnica reciente que consiste en almacenar los elementos de la matriz en forma compacta. En este esquema se sustituye la matriz A por dos matrices de perfil irregular VA y PA respectivamente, siendo VA la matriz de valores en doble precisión y PA la matriz de posiciones. [41].

A modo de ejemplo consideremos la siguiente matriz de orden 8×8 según se muestra en la figura siguiente:

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 & 0 & a_{18} \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 & 0 & 0 \\ a_{41} & 0 & a_{43} & a_{44} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{77} & 0 \\ a_{81} & 0 & 0 & 0 & 0 & 0 & 0 & a_{88} \end{pmatrix}$$

Fig. 3.9.0.

Las matriz VA tendrá en la primera columna los elementos de la diagonal, y en las restantes columnas los elemntos no nulos de cada fila. En la matriz de posiciones PA, figurarán en la primera columna el número de elementos no nulos de cada fila, y en las siguientes las posiciones de columnas respectivas que ocupan. Sean VA y PA de la figura 3.9.0, las siguientes matrices:

$$VA = \begin{pmatrix} a_{11} & a_{12} & a_{14} & a_{18} \\ a_{22} & a_{21} & a_{23} & \\ a_{33} & a_{32} & a_{34} & \\ a_{44} & a_{41} & a_{43} & \\ a_{55} & a_{56} & & \\ a_{66} & a_{65} & & \\ a_{77} & & & \\ a_{88} & a_{81} & & \end{pmatrix} \qquad PA = \begin{pmatrix} 4 & 2 & 4 & 8 \\ 3 & 1 & 3 & \\ 3 & 2 & 4 & \\ 3 & 1 & 3 & \\ 2 & 6 & & \\ 2 & 5 & & \\ 1 & & & \\ 2 & 1 & & \end{pmatrix}$$

Capítulo 4: Métodos dispersos generales.

4.0 Introducción.

En este capítulo consideramos métodos que tratan de explotar todos los elementos ceros existentes en el factor triangular L de A.

El algoritmo que estudiaremos es el llamado algoritmo de grado mínimo, el cual es un algoritmo heurístico para hallar el ordenamiento de la matriz A, y que experimenta bajo relleno de huecos cuando se factoriza. Este algoritmo puede ser ampliamente empleado en aplicaciones industriales.

4.1 Factorización simétrica.

Sea A una matriz simétrica dispersa, la estructura no cero de A está definida por:

$$\text{Nonul}(A) = \left\{\{i,j\} / a_{ij} \neq 0, i \neq j\right\}$$

Supongamos que usando el algoritmo de Cholesky la matriz se factoriza en LL^t. La matriz rellena de A, F(A) es la suma matricial de $L + L^t$. Cuando la matriz objeto de estudio sea evidente a partir del contexto, emplearemos F en vez de F(A). La estructura no cero correspondiente será:

$$\text{Nonul}(F) = \left\{\{i,j\} / l_{ij} \neq 0, i \neq j\right\}$$

Nosotros supondremos que la cancelación numérica exacta no existe, de forma que para una estructura no nula dada Nonul(A) la correspondiente Nonul(F) está completamente determinada, es decir, Nonul(F) es independiente de las entradas numéricas de A. La suposición de no cancelación implica que: Nonul (A) $\subset$ Nonul (F) y el relleno de huecos de la matriz en A puede ser definido por: Rell(A)= Nonul(F) - Nonul(A).

Como ejemplo tomamos la matriz A, donde las entradas del relleno de huecos están indicadas por ⊗ :

$$\begin{bmatrix} 1 & & x & & & x \\ & 2 & \otimes & x & & \\ x & \otimes & 3 & x & & \otimes \\ & x & x & 4 & & x \\ & & & & 5 & \otimes \\ x & & \otimes & x & \otimes & 6 \end{bmatrix}$$

Nonul (A)={(1, 3) (1, 6) (2, 4) (3, 4) (4, 6)}

Relleno(A)={(2, 3) (3, 6) (5, 6)}

x: Nonulo

⊗: fill-in

A continuación veremos cómo el Rell(A) puede obtenerse a partir de Nonul(A).

4.2 Modelo del grafo de eliminación.

Nos referimos ahora a la aplicación de la eliminación Gaussiana simétrica para la matriz A, para los cambios correspondientes en su grafo G(A).

En primer lugar recordemos la versión producto externo del algoritmo de Cholesky:

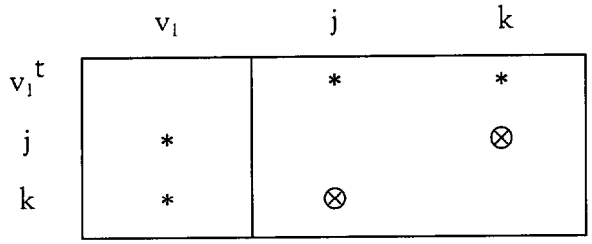
$$A = A_0 = H_0 = \begin{bmatrix} d_1 & v_1^t \\ v_1 & \overline{H}_1 \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 \\ \dfrac{v_1}{\sqrt{d_1}} & I_{n-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{d_1} & \dfrac{v_1}{\sqrt{d_1}} \\ 0 & I_{n-1} \end{bmatrix} = L_1 A L_1^T,$$

donde $H_1 = \overline{H}_1 - \dfrac{v_1 v_1^t}{d_1}$, siendo d_1 un número real positivo y H una matriz cuadrada de orden n-1. El paso básico se aplica a H_1 , H_2 y así sucesivamente.

Partiendo de la suposición usual de que la cancelación exacta no ocurre, la ecuación:

$$H_l = \overline{H}_l - \frac{v_l v_l^t}{d_l}$$

implica que la entrada jk-esíma de H_l es no cero si la entrada que ocupa esa posición en $\overline{H}_l$ es ya no cero o si $(v_l)_j \neq 0$ y $(v_l)_k \neq 0$. Si $(H_l)_{jk}=0$ y $(v_l)_j \neq 0$ y $(v_j)_k \neq 0$, tiene lugar un relleno de huecos en la posición jk-ésima. Gráficamente se tiene:



Completando el primer paso de la factorización, factorizamos H_l.

Establezcamos una correspondencia entre la transformación de H_0 a H_l y los correspondientes cambios en sus respectivos grafos [24].

Indicaremos los grafos H_0 (=A) y H_l por $G(H_0)$ y $G(H_l)$ respectivamente.
$G(H_l)$ se obtiene de $G(H_0)$ mediante:

a) Supresión de x_l y de sus aristas incidentes.

b)Adjunción de aristas de forma que los nodos de $Ady(x_l)$ sean pares adyacentes en $G(H_l)$. Este es el llamado esquema de Parter.

De esta manera la eliminación Gaussiana simétrica se puede interpretar como la generación de una sucesión de grafos llamados "grafos de eliminación".

$$G_i^\alpha = G^{H_l} = (X_i^\alpha, E_i^\alpha), \text{ para } i = 1,2,\ldots,n-1$$

siendo:

$$G_0^\alpha = G^{H_0} = G(H_0) = G(A)$$

donde α representa el ordenamiento de G(A) inducido por A.

Los conjuntos de aristas de E(F) y E(A) se seleccionan según el siguiente:

❑ **Lema:**

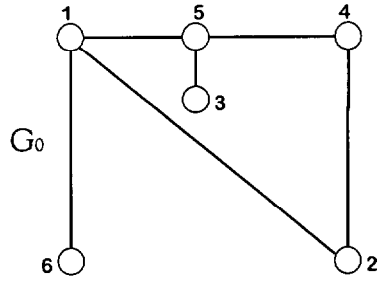
Sea $\{x_i, x_j\} \in E(F) \Leftrightarrow \{x_i, x_j\} \in E(A) \vee \{x_i, x_k\} \in E(F) \wedge \{x_{ki}, x_j\} \in E(F)$, para algú n $k(m_i\{i,j\}$

Demostración:

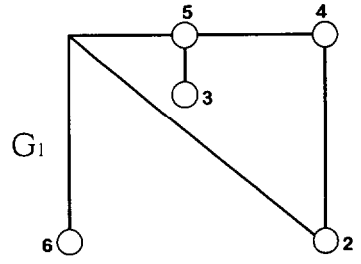
Al pasar de A_{k-1} a A_k las entradas de la k-ésima columna de A_k situadas debajo de la diagonal se anulan, lo cual se traduce en la supresión del nodo k y sus aristas (paso 1). El coeficiente H_k (i, j) es no nulo si $\overline{H}_{k-1}$ (i, j) es no nulo, lo cual significa que G_{k-1} contiene ya la arista {i, j} solo si $(v_{k-1})_i \neq 0$ y $(v_{k-1})_j \neq 0$ y esto siempre que G_{k-1} contenga las aristas {i, k} y {k, j} con $k < m_i$ {i, j}.

Ejemplo de sucesión de grafos de eliminación, cuando se suprime x_i y sus aristas incidentes y a continuación la adjunción de aristas como se describió anteriormente.

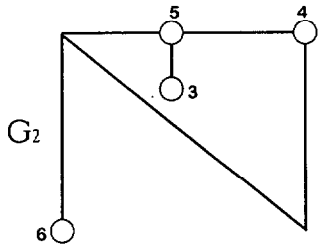
$$\text{Sea } A = H_0 = \begin{bmatrix} 1 & & & & & \\ x & 2 & & \text{sim} & & \\ & x & 3 & & & \\ & & x & 4 & & \\ x & & & x & 5 & \\ x & & & & & 6 \end{bmatrix}$$



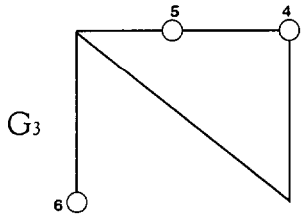
$$H_1 = \begin{bmatrix} 2 & & & & \\ & 3 & & & \\ x & x & 4 & & \\ \otimes & & x & 5 & \\ \otimes & & & \otimes & 6 \end{bmatrix}$$



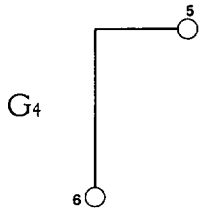
$$H_2 = \left[\left[\begin{array}{ccc} 3 & & \\ & 4 & \\ x & x & 5 \\ & \otimes & \otimes \quad 6 \end{array}\right]\right]$$



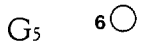
$$H_3 = \left[\left[\begin{array}{ccc} 4 & & \\ x & 5 & \\ \otimes & \otimes & 6 \end{array}\right]\right]$$



$$H_4 = \left[\left[\begin{array}{cc} 5 & \otimes \\ \otimes & 6 \end{array}\right]\right]$$

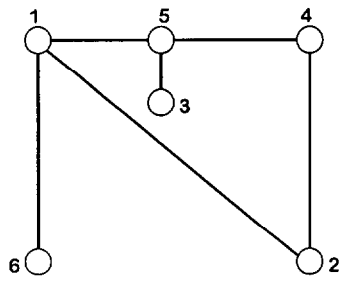


$$H_5 = [[6]]$$



- El factor de eliminación y el grafo correspondiente del ejemplo anterior son:

$$L = \begin{bmatrix} 1 & & & & & \\ x & 2 & & & & \\ & & 3 & & & \\ & x & & 4 & & \\ x & \otimes & x & x & 5 & \\ x & \otimes & & \otimes & \otimes & 6 \end{bmatrix}$$



G(F)

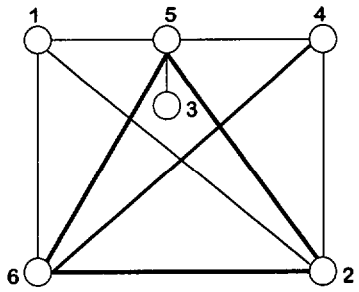
Observando la estructura de L vemos que Rell(A)={(2,5) (2,6) (4,6) (5,6)}.

- **Definición:**

Dada una matriz A simétrica y definida positiva, si L es el factor de Cholesky, se denomina grafo relleno de G(A) al grafo G(F)=(X(F), E(F)) donde $F = L + L^t$.

El conjunto de aristas E(F) consta de todas las aristas de E(A) unidas a todas las aristas añadidas durante la factorización. Al grafo relleno también lo llamaremos grafo de Parter y al método de determinar la eliminación lo denominaremos "modelo explícito". En el ejemplo anterior tenemos, que a la matriz L obtenida le corresponde el grafo G(F) llamado grafo de Parter según la figura 4.2.0:

$$L = \begin{pmatrix} 1 & & & & & \\ x & 2 & & & & \\ & & 3 & & & \\ & x & & 4 & & \\ x & \otimes & x & x & 5 & \\ x & \otimes & & \otimes & \otimes & x \end{pmatrix}$$



G(F)

Fig 4.2.0.

Si la matriz que se ha factorizado es la matriz:

$$A = \begin{bmatrix} 1 & & & & & \\ x & 2 & & sim & & \\ & x & 3 & & & \\ & & x & 4 & & \\ x & & & x & 5 & \\ x & & & & & 6 \end{bmatrix}$$

observamos que comparando la matriz L con la matriz A, vemos que el relleno que se ha producido en A es: Rell (A) = { {2,5}, {2,6}, {4,6},{5,6}}.

Hemos visto que el grafo relleno G(F) puede ser fácilmente construido a partir de la secuencia de grafos de eliminación. Es importante hallar G(F) puesto que contiene la estructura de L.

4.3 Modelación de la eliminación mediante conjuntos accesibles.

En la sección anterior determinamos la sucesión de los grafos de eliminación G_0 , G_1 , ..., G_{n-1} que proporcionan una característica recursiva del conjunto de aristas E(F). A menudo resulta de ayuda tanto en términos teóricos como de cálculo tener la caracterización de G_i y E(F) directamente en términos del grafo original G(A). Nuestro objetivo en esta sección consiste en proporcionar tales caracterizaciones empleando conjuntos accesibles.

Estudiemos en primer lugar, la manera en que se forma la arista de relleno $\{x_4 ,x_6\}$ en el ejemplo de la figura 4.3.0. Sea el grafo G_0, y G_1 el grafo obtenido al eliminar el nodo 1 en G_0 :

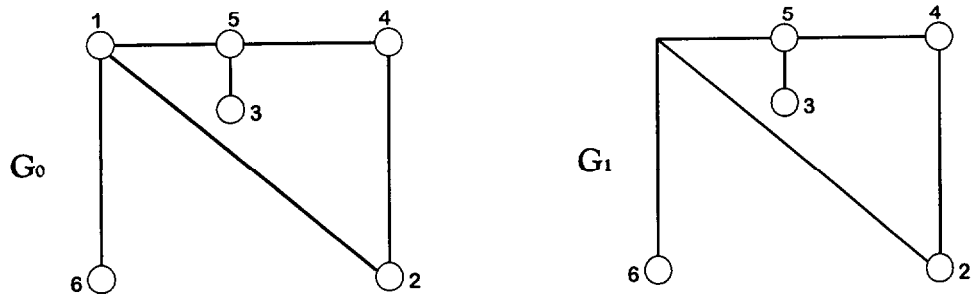


Fig. 4.3.0.

En G_1 está la trayectoria (x_4 , x_2 , x_6) de forma que cuando se elimina x_2, se crea la arista $\{x_4 ,x_6\}$.No obstante, la arista $\{x_2, x_6\}$ no está presente en el grafo original G_0 , se forma a partir de la trayectoria (x_2 , x_1 , x_6) cuando x_1 se elimina en G_0.

Combinando ambos vemos que la trayectoria (x_4 , x_2 , x_1 , x_6) en el grafo original es la responsable de la arista de relleno del hueco $\{x_4 , x_6\}$.

$\{\underline{4, 6}\}$ en G_2

↑

$(4,\underline{2,6})$ en G_1

↑

(4,2,1,6) en G_0

Esta observación nos induce a definir un nuevo concepto, que es el de nodos accesibles.

❑ Definición:

Sea G=(X , E) y S ⊂ X. Sea x ∈ X - S. Se dice que el nodo x es accesible desde un nodo y, a través de S, si existe una trayectoria (y, v_1 , v_2 , ..., v_k ,x), desde y hasta x, tal que $v_i \in S$ para $1 \leq i \leq k$.

Obsérvese que k puede valer cero, de forma que cualquier nodo adyacente a y no perteneciente a S es accesible desde y a través de S, es decir, Ady (y)- S ⊂ Acces (y, S) donde Acces (y, S) es el conjunto de nodos accesibles desde y a través de S, es decir: Acces (y, S) = {x ∈ X-S / x es accesible desde y a través de S}

Para ilustrar la noción de conjuntos accesibles consideremos el ejemplo de la figura 4.3.1:

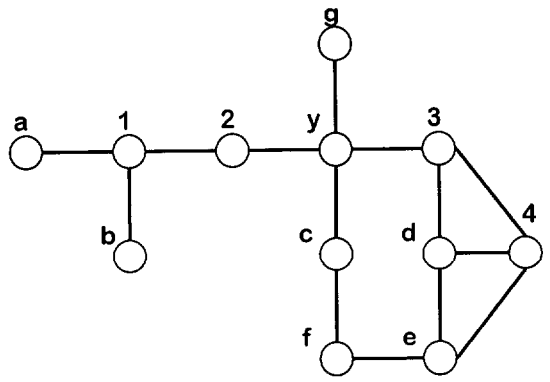


Fig. 4.3.1.

Sea S = {1, 2, 3, 4}, entonces Acces (y, S) = {a, b, c, d, e, g}, puesto que podemos hallar las trayectorias siguientes a través de S:

Las trayectorias a través de S son: (y, 2, 1, a) (y, 2, 1, 6) (y, c) (y ,3 ,d) (y, 3, 4, e) (y, g), por lo tanto Acces(y, S) = {a, b, c, d, e, g}.

A continuación enunciamos un teorema que caracteriza el grafo relleno mediante los conjuntos accesibles:

❑ **Teorema:**

$E(F) = \{\{ x_i , x_j \}/ x_j \in Acces (x_i , \{x_1 , x_2 , \dots , x_{i-1} \})\}$

Demostración:

Si asumimos un $x_j \in Acces (x_i , \{x_1 , x_2 , \dots , x_{i-1} \})$, entonces existe un camino $(x_i, y_1, \dots, y_t, x_j)$ en G(A) con $y_k \in \{x_1, x_2, \dots, x_{i-1}\}$ para $1 \leq k \leq t$. Si $t=0$ ó $t=1$ el resultado sae obtiene inmediatamente. Si $t>1$ por inducción simple en t se demuestra que $\{x_i, x_j\} \in E(F)$.

De la misma manera si $\{x_i, x_j\} \in E(F)$ e $i \leq j$, por inducción en i se observa que el resultado es cierto para $i=1$, como $\{x_i, x_j\} \in E(F)$ esto implica que $\{x_i, x_j\} \in E(A)$. Si suponemos que esto es verdadero para términos menores que i y supongamos que exista una k < que el mínimo{i, j} tal que $\{x_i, x_k\} \in E(F)$ y $\{x_j, x_k\} \in E(F)$. Por inducción puede encontrase un camino de x_i a x_j a través de x_k en la sección del grafo $G(A)(\{x_1, \dots, x_k\} \cup \{x_i, x_j\})$ lo que implica que $x_j \in Acces (x_i,\{x_1 , x_2 , \dots , x_{i-1} \})$.

En términos matriciales, el conjunto $Acces (x_i , \{x_1 , x_2 , \dots , x_{i-1} \})$ es el conjunto de subíndices filas que corresponden a las entradas no cero en el vector columna L*i

Consideremos el ejemplo de la figura 4.3.2:

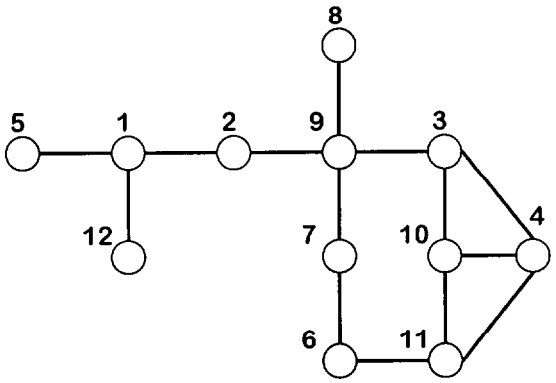


Fig. 4.3.2.

Si $S_i = \{1, 2 , \dots , i \}$ se tiene:

$$\text{Acces}(1, S_0) = \{2, 5, 12\}$$
$$\text{Acces}(2, S_1) = \{5, 12, 9\}$$
$$\text{Acces}(3, S_2) = \{9, 10, 4\}$$
$$\text{Acces}(4, S_3) = \{10, 11, 9\}$$
$$\text{Acces}(5, S_4) = \{2, 9\}$$
$$\text{Acces}(6, S_5) = \{7, 11\}$$
$$\text{Acces}(7, S_6) = \{9, 11\}$$
$$\text{Acces}(8, S_7) = \{9\}$$
$$\text{Acces}(9, S_8) = \{12, 10, 11\}$$
$$\text{Acces}(10, S_9) = \{11, 12\}$$
$$\text{Acces}(11, S_{10}) = \{12\}$$
$$\text{Acces}(12, S_{11}) = \varnothing$$

Resulta, pues, que a partir del teorema anterior la estructura L correspondiente es la siguiente:

$$L = \begin{bmatrix}
1 & & & & & & & & & & & \\
 & 2 & & & & & & & & & & \\
 & & 3 & & & & & & & & & \\
 & & x & 4 & & & & & & & & \\
x & x & & & 5 & & & & & & & \\
 & & & & & 6 & & & & & & \\
 & & & & & x & 7 & & & & & \\
 & & & & & & & 8 & & & & \\
 & x & x & & & & x & x & 9 & & & \\
 & & x & x & & & & & x & 10 & & \\
 & & & x & & x & & & & x & 11 & \\
x & x & & & x & & & & x & x & x & 12
\end{bmatrix}$$

De esta manera hemos caracterizado la estructura de L directamente, en términos de la estructura de A, siendo A la matriz que hemos factorizado. Dicha matriz corresponde al grafo de la figura 4.3.2, y es la que exponemos a continuación:

$$A = \begin{bmatrix}
1 & & & & & & & & & & & \\
 & 2 & & & & & & & & & & \\
 & & 3 & & & & & & & & & \\
 & & x & 4 & & & & & & & & \\
x & x & & & 5 & & & & & & & \\
 & & & & & 6 & & & & & & \\
 & & & & & x & 7 & & & & & \\
 & & & & & & & 8 & & & & \\
 & x & x & & & & x & x & 9 & & & \\
 & & x & x & & & & & x & 10 & & \\
 & & & x & & x & & & & x & 11 & \\
x & x & & & x & & & & x & x & x & 12
\end{bmatrix}$$

El relleno de huecos donde se ha producido efecto fill-in es:

Rell (A)={{2, 5},{4, 9},{5, 9},{9, 10},{7, 11},{9, 11},{2, 12},{6, 12},{9, 12},{10, 12},{11, 12}}.

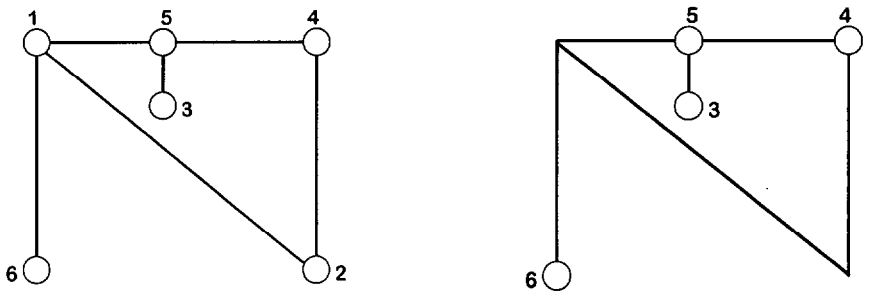
Establezcamos a continuación una caracterización para los grafos de eliminación, en términos de conjuntos accesibles.

Sea $G_0, G_1, G_2 \ldots G_{n-1}$ la sucesión de grafos de eliminación definidos por los nodos $x_1, x_2, \ldots, x_n$ y consideremos el grafo $G_i = (X_i, E_i)$.

Sea y un nodo en el grafo de eliminación $G_i = (X_i, E_i)$. El conjunto de nodos adyacentes a y en G_i viene dado por Acces $(y, \{x_1, x_2, \ldots, x_i\})$ donde el operador Acces se aplica al grafo original G_0. Es decir:

$$Ady_{Gi}(y) = Acces_{Go}(y, \{x_1, x_2, \ldots\ldots, x_n\}) \Leftrightarrow Ady_{Gi}(y) = Acces_{Go}(y, S_i).$$

Consideremos los grafos G_0 y G_2, del ejemplo anterior (Fig 4.3.0):



Sea $S_2 = \{1, 2\}$ el conjunto de nodos eliminados:

$$Ady_{G2}(3) = Acces_{G0}(3, S_2) = \{5\}$$
$$Ady_{G2}(4) = Acces_{G0}(4, S_2) = \{5, 6\}$$
$$Ady_{G2}(5) = Acces_{G0}(5, S_2) = \{3, 4, 6\}$$
$$Ady_{G2}(6) = Acces_{G0}(6, S_2) = \{4, 5\}$$

La importancia de los conjuntos accesibles en la eliminación dispersa yace en esta observación:

Dado un grafo $G = (X, E)$ y una sucesión de nodos de eliminación $x_1, x_2..., x_n$, el proceso total de eliminación se puede describir por ésta sucesión $x_1, x_2..., x_n$, y el operador Acces. Esto se puede considerar como un modelo implícito para la eliminación, opuesto al modelo explícito empleando grafos de eliminación $G_0, G_1, ..., G_{n-1}$.

4.4 Representación en ordenador de los grafos de eliminación.

La eliminación Gaussiana sobre un sistema lineal simétrico disperso, se puede modelar por la secuencia de grafos de eliminación. En esta sección estudiaremos la representación y transformación de los grafos de eliminación sobre un ordenador.

❑ Representacion explícita e implícita.

Los grafos de eliminación son después de todo, grafos simétricos tales que se pueden representar explícitamente empleando uno de los esquemas de almacenamiento descritos anteriormente. No obstante lo que a nosotros nos concierne es que la implementación se deberá confeccionar para la eliminación, de forma que la transformación a partir de uno de los grafos de eliminación al siguiente en la secuencia, se puede llevar a cabo de forma fácil [17].

Revisemos los pasos de la transformación. Sea G_i el grafo de eliminación obtenido tras eliminar el nodo x_i desde G_{i-1}. La estructura de adyacencia de G_i se puede obtener de la siguiente manera:

Paso 1: Determinar el conjunto adyacente $Ady_{Gi-1}(x_i)$ en G_{i-1}.

Paso 2: Eliminar el nodo x_i y su lista de adyacencia a partir de la estructura de adyacencia.

Paso 3: Para cada nodo $y \in Ady_{Gi-1}(x_i)$, el nuevo conjunto adyacente de y en G_i esta dado por la unión de subconjuntos :

$$(\mathrm{Ady}_{G_{i-1}}(y) - \{x_i\}) \cup (\mathrm{Ady}_{G_{i-1}}(x_i) - \{y\})$$

Lo anterior es una formulación algorítmica del esquema de Parter para efectuar la transformación.

Hay dos puntos acerca de la implementación de este algoritmo que debemos mencionar:

- En primer lugar, el espacio empleado para almacenar $Ady_{Gi-1}(x_i)$ en la estructura de adyacencia, se puede reemplazar después del paso 2.

- En segundo lugar, la estructura de adyacencia determinada por G_i puede requerir más espacio que la de G_{i-1}.

A la vista de estas observaciones se ha de emplear una estructura de datos muy flexible en la implementación explícita, que permita el cambio dinámico en la estructura de los grafos de eliminación. La estructura de adyacencia (ADYACI, XADJ) es una buena candidata.

La representacion explícita en el ordenador presenta dos desventajas:

❶ La flexibilidad en la estructura de datos requiere, a menudo, un almacenamiento principal de memoria bastante significativo.

❷ La cantidad máxima de almacenamiento requerida es impredecible, se necesita suficiente almacenamiento para el grafo de eliminación más grande (en el número de aristas) G_i que tiene lugar, esta demanda de memoria puede exceder ampliamente la disponibilidad para el grafo original G_o.

También el máximo de memoria requerido no se conoce hasta el final del proceso íntegro.

Por estas razones debemos pensar en el modelo implícito de la eliminación sparse.

Para la representación implícita empleamos el grafo original G_o y el subconjunto de nodos eliminados $S_i = \{x_1, x_2, ..., x_i\}$.

El conjunto de nodos adyacentes a y en G_i se puede recuperar generando el conjunto accesible Acces(y, S_i) en el grafo original. La representación implícita no tiene ninguna de las desventajas del modelo explícito. El modelo implícito tiene una demanda de almacenamiento pequeña y predecible, preservando la estructura de adyacencia del grafo dado.

Sin embargo la cantidad de trabajo que se necesita para determinar conjuntos accesibles puede ser demasiado grande en los últimos estadios de la eliminación cuando $|S_i|$ es grande.

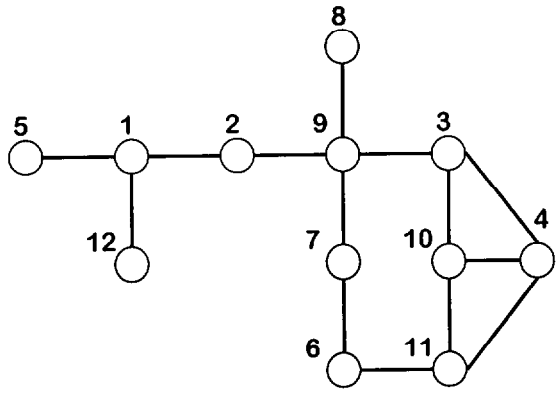
4.5 Grafos Cocientes.

Para formalizar éste nuevo modelo de eliminación, introducimos la noción de grafo cociente. El modelo del grafo cociente es un modelo más adecuado que el implícito para la implementación en el ordenador [24].

Para ello, vamos a clasificar por clases los nodos ya eliminados, juntando en cada clase los nodos conectados entre sí.

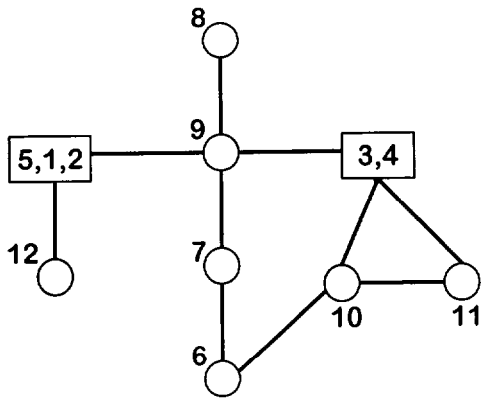
❑ **Ejemplo:**

Sea G_0 el grafo de la figura 4.3.2:



En el ejemplo expuesto si $S = \{1, 2, 3, 4, 5\}$, entonces en la sección del grafo G(S) hay dos componentes conexas, cuyos conjuntos de nodos son $\{1, 2, 5\}$ y $\{3, 4\}$.

Mediante la formación de 2 supernodos o clases obtenemos el siguiente grafo:



Para representar estas componentes conexas en S establezcamos las siguientes clases: $\overline{5} = \{1,2,5\}$ $\overline{3} = \{3,4\}$

Con este nuevo grafo, observamos que la trayectoria $(9,\bar{4},11)$ es de longitud 2 y nos conduce del nodo 9 al 11.

Con esta estrategia, todas las trayectorias son de longitud igual o menor que 2. Esto presenta una notable ventaja sobre la aproximación al conjunto accesible en el grafo original, en el que las trayectorias pueden ser de longitudes arbitrarias menores que n.

Con respecto a la eliminación explícita, ésta otra identificación de nodos se puede implementar en el mismo espacio de memoria, no necesita más espacio que el de la estructura del grafo original.

En definitiva, ésta nueva estructura del grafo se puede usar para generar conjuntos accesibles (en el modelo implícito), o conjuntos adyacentes en los grafos de eliminación (en el modelo explícito) de forma bastante eficente y, no obstante se requiere una cantidad fija de almacenamiento.

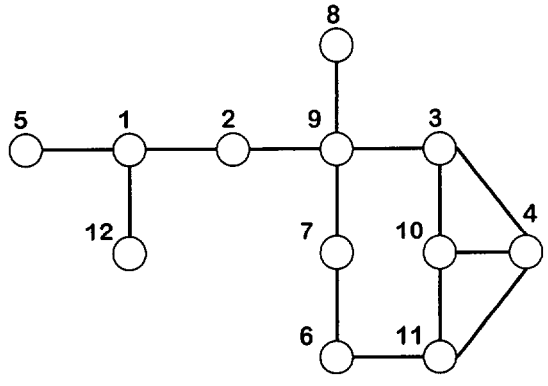
Para formalizar este nuevo modelo de eliminación introducimos la noción de grafo cociente.

Sea G=(X, E) el grafo original, y sea P una partición sobre el conjunto de nodos X:

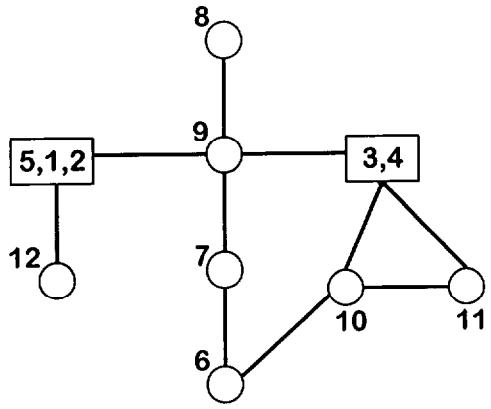
$$P = \{Y_1, Y_2, \ldots, Y_p\} \text{ o sea: } \bigcup_{i=1}^{p} Y_i = X \; ; \quad Y_i \cap Y_j \neq \varnothing \text{ para } i \neq j .$$

Definiremos el grafo cociente G respecto de P como el grafo $(P, \xi) = G/P$, donde $\{Y_i, Y_j\} \in \xi \Leftrightarrow Adj(Y_i) \cap Y_j \neq \varnothing$.

Sea el grafo del ejemplo anterior (Fig. 4.3.2):



Si unimos los nodos 1, 2 y 5 en un supernodo, así como los nodos 3 y 4 en otro supernodo nos queda el grafo cociente siguiente:



Quedando en el grafo la siguiente particion P={{1, 2, 5}, {12}, {7}, {8}, {9}, {6}, {11}, {10}, {3, 4}}, donde observamos que se han fundido los nodos 1, 2 y 5 en un " supernodo " y 3 y 4 en otro " supernodo ".

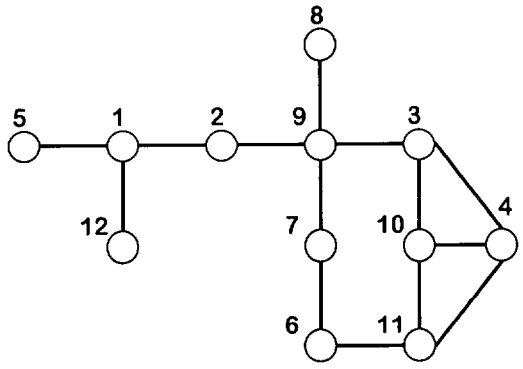
Generalicemos esto a las distintas fases de eliminación.

Sea G=(X, E) el grafo original. Sea $S = S_i = \{1, 2, \ldots, i\}$ el conjunto de nodos eliminados en la fase i, i.e. Asociemos a éste S un grafo cociente, para ello definamos:

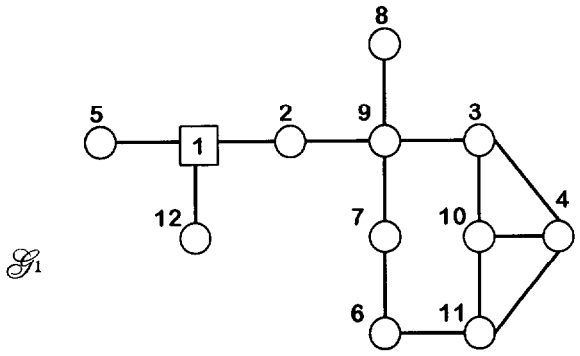
Sea $\zeta(S) = \{C \subset S / G(C)$ una componente conexa en el subgrafo $G(S)\}$ y una partición en X dada por: $\bar{\zeta}(S) = \{\{y\} / y \in X - S\} \cup \zeta(S)$. Esto determina unívocamente el grafo corriente $G / \bar{\zeta}(S)$, que se puede considerar como el grafo obtenido mediante la fusión de conjuntos conexos en S.

A continuación mostramos la secuencia de grafos cocientes del ejemplo anterior, correspondiente al grafo G=(X, E).

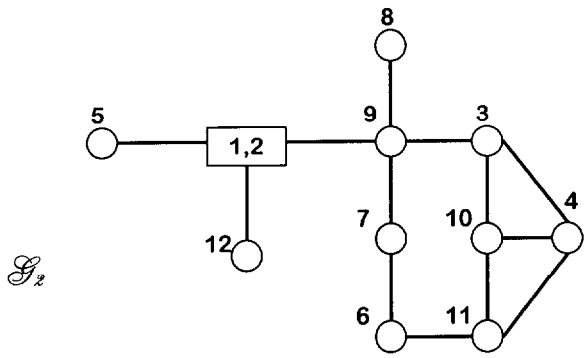
Sea G_0 el grafo de la figura 4.3.2



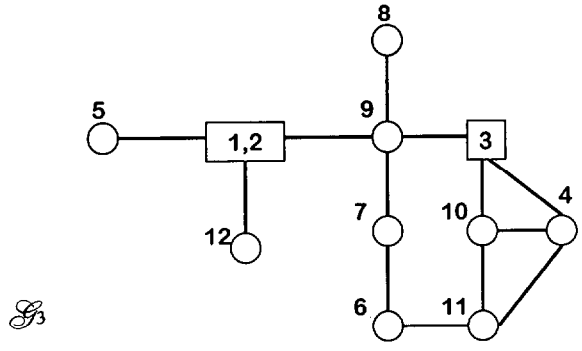
Mediante la fusión de nodos en supernodos queda la siguiente secuencia de grafos cocientes:



$\mathscr{G}_1$

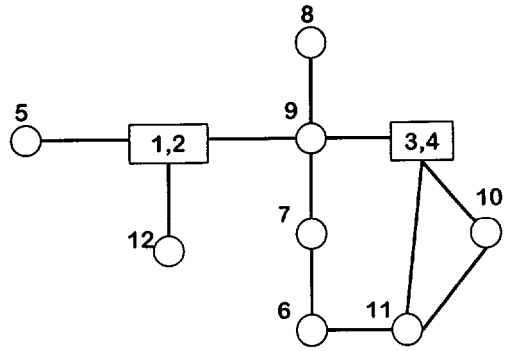


$\mathscr{G}_2$

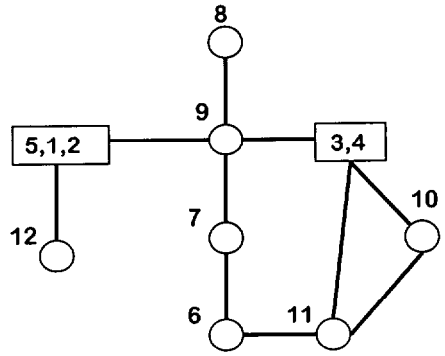


$\mathscr{G}_3$

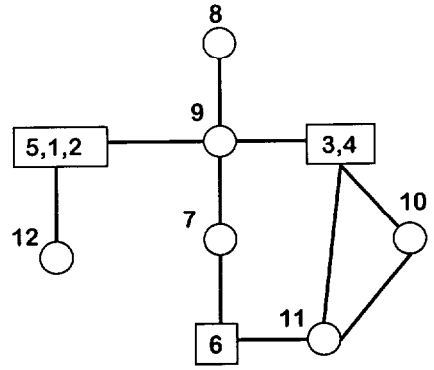
$\mathscr{G}_4$



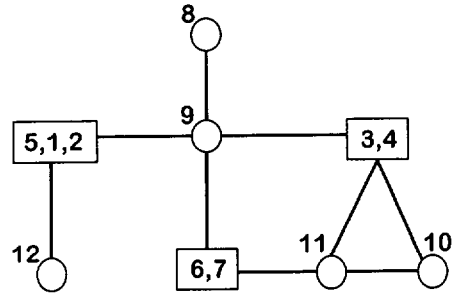
$\mathscr{G}_5$



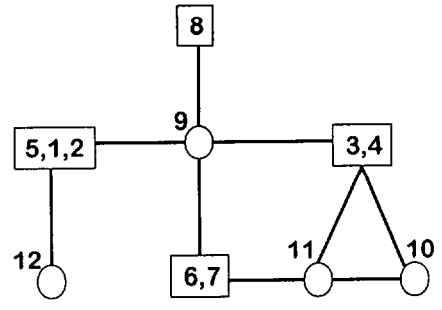
$\mathscr{G}_6$

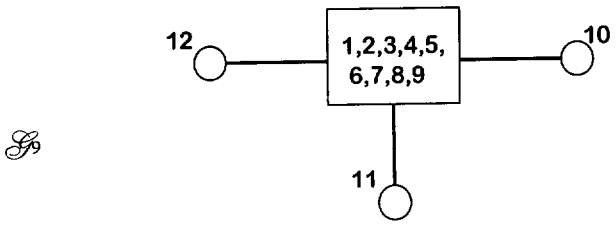


$\mathscr{G}_7$

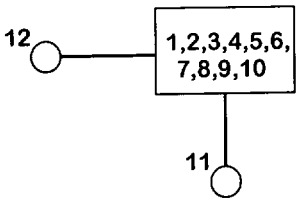


$\mathscr{G}_8$

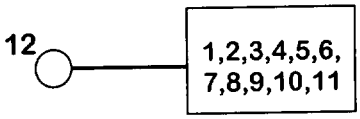




$\mathscr{G}_9$



$\mathscr{G}_{10}$

$\mathscr{G}_{11}$ 

Estudiemos la relevancia de los grafos cocientes en la eliminación.

Sea $x_1, x_2, \ldots, x_n$ la secuencia de nodos eliminados en el grafo G. Sea S_i = {1,2,...,i} con i= 1 , 2 , ... , n . Para cada i, el conjunto S_i induce la partición $\bar{\zeta}(S_i)$ y el correspondiente grafo cociente $\mathscr{G}_i = G / \bar{\zeta}(S_i) = \left(\bar{\zeta}_i(S_i), \xi_i\right)$ teniéndose una sucesión de grafos cocientes $\mathscr{G}_1, \mathscr{G}_2, \mathscr{G}_3, \ldots, \mathscr{G}_n$

Los grafos cocientes de la forma: $\mathscr{G}_i = G / \bar{\zeta}(S_i) = \left(\bar{\zeta}(S_i), \xi_i\right)$ son en realidad, representación de grafos de eliminación.

Para y $\in$ X-S_i, Acceso(y, S_i) = Acceso$_{\mathscr{G}}$(y, $\zeta(S_i)$), la determinación de conjuntos accesibles en el grafo cociente $\mathscr{G}_i$ es bastante simple.

Para un nodo y, no perteneciente a $\zeta(S_i)$, el siguiente algoritmo nos proporciona el conjunto $Acces_{\mathscr{G}}(y, \zeta(S_i))$.

Paso 1: Inicialización:

$A \leftarrow \varnothing$

Paso 2: Determinación de los nodos accesibles:

Para $x \in Ady_{\mathscr{G}_i}(y)$.

Si $x \in \zeta(S_i)$ entonces $A \longleftarrow A \cup Ady_{\mathscr{G}_i}(x)$

si no $A \longleftarrow A \cup \{x\}$

Fin si

Fin para

Paso 3: Salida.

* El conjunto accesible viene dado por A.

La obtención de los grafos de eliminación G_i a partir de los grafos cocientes $\mathscr{G}_i$ se consigue mediante el siguiente algoritmo:

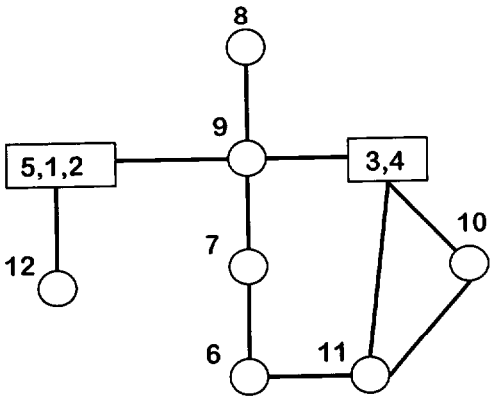
Paso 1: Eliminar los supernodos en $\zeta(S_i)$ y sus aristas incidentes del grafo cociente.

Paso 2: Para cada $C \in \zeta(S_i)$, añadir aristas al grafo cociente, de modo que todos los nodos adyacentes a C sean pares adyacentes en el grafo de eliminación.

❑ **Ejemplo:**

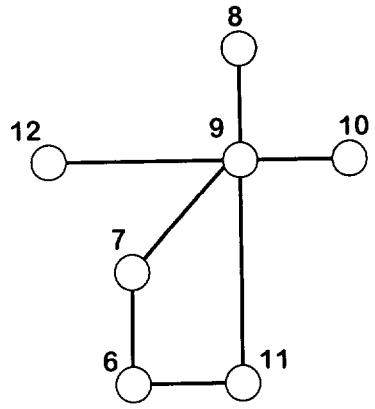
Obtención del grafo de eliminación a partir del grafo cociente del ejemplo anterior:

$\mathscr{G}_5$



Grafo cociente.

G_5



Grafo de eliminación

En términos implícitos, el modelo de grafo cociente se sitúa entre la aproximación de conjuntos accesibles y el modelo de grafos de eliminación, como vehículo para representar el proceso de eliminación.

Conjunto Acces en el Grafo original → Grafo cociente → Grafo de eliminación

La correspondencia entre los tres modelos se resume en la siguiente tabla:

	M I	M C	M E
Representación	S_1 S_2 . . . S_{n-1}	$\mathscr{G}_1$ $\mathscr{G}_2$ . . . $\mathscr{G}_{n-1}$	G_1 G_2 . . . G_{n-1}
Adyacencia	Acces(y, S_i)	Acces$_{\mathscr{G}_i}$(y, $\zeta(S_i)$)	Ady$_{G_i}$ (y)

Siendo M I= modelo implícito , M C= modelo cociente y M E= modelo explícito.

4.5.1 Implementación del modelo del grafo cociente.

Sea el grafo cociente $\mathscr{G} = G / \overline{\zeta}(S)$ inducido por el conjunto eliminado S.

Si s ∈ S, $\overline{s}$ denotará la componente conexa en el subgrafo G(S) que contiene al nodo s.

Sea el grafo de la figura 4.5.0:

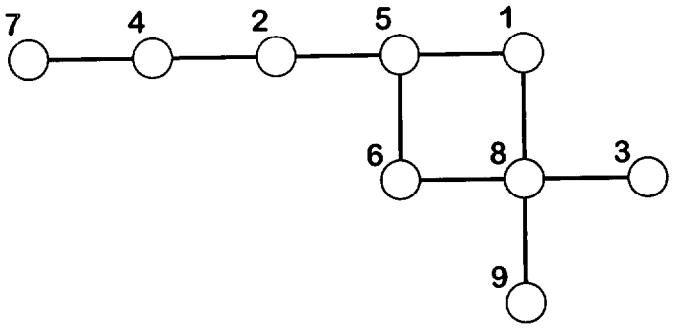
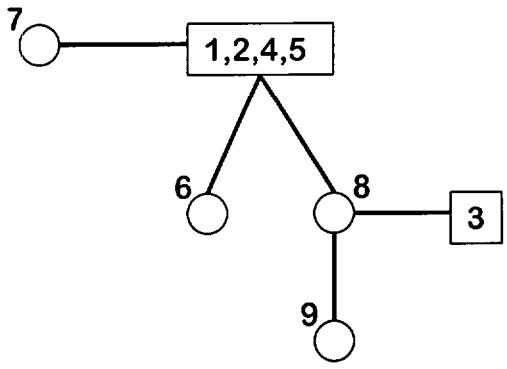


Fig. 4.5.0

Y sea S ={1, 2, 3, 4, 5} el conjunto de nodos eliminados representándose por :

$$\overline{5} = \overline{1} = \overline{2} = \overline{4} = \{1,2,4,5\}$$

Siendo su grafo cociente el siguiente:



Por otra parte, para una $C \subset \zeta(S)$ dado, podemos seleccionar cualquier nodo $x \in C$ y usar x como representante de C, i.e. $\bar{x} = C$.

Antes de discutir la elección del representante en la implementación, estableceremos algunos resultados que sirven para mostrar que el modelo se puede implementar " in situ" i.e. en el espacio previsto por la estructura de adyacencia del grafo original.

Sea G = (X, E) y $C \subset X$ donde G(C) es un subgrafo conexo.

Entonces se tiene: $\sum_{X \in C} |Ady(x)| \geq |Ady(C)| + 2(|C| - 1)$.

En efecto, puesto que G(C) es conexo, hay al menos |C|-1 aristas en el grafo, estas aristas están contadas dos veces.

Sea $x_1, x_2, \ldots, x_n$ la secuencia de nodos y $S_i = \{x_1, x_2, \ldots, x_n\}$, $1 \leq i \leq n$. Para $1 \leq i \leq n$ sea $\mathcal{G}_i = G / \bar{\zeta}(S) = (\bar{\zeta}(S), \xi_i)$, $y \in X - S_i$, Entonces: $|Ady_{G_2}(y)| \geq Ady_{\mathcal{G}_3}(y)| \geq \ldots \geq |Ady_{G_i})y)$.

❑ **Teorema:**

$$\max_{1 \leq i \leq n} |\xi_i| \leq |E|$$

O Demostración:

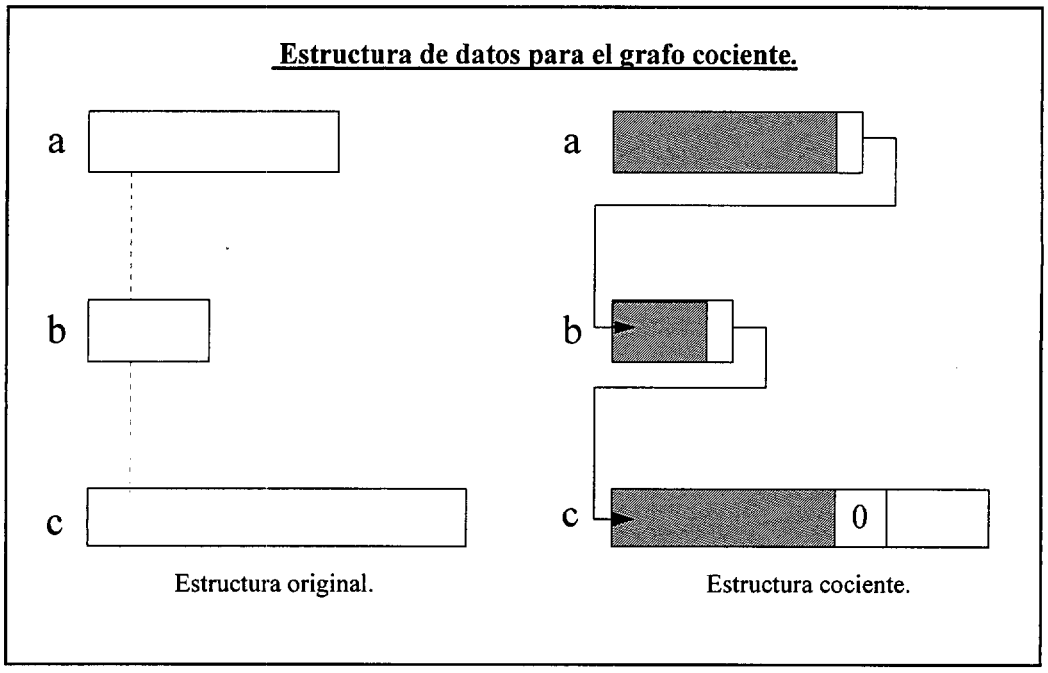
Sean los grafos cocientes $\mathscr{G}_i$ y $\mathscr{G}_{i+1}$.

Si x_{i+1} está aislado en el subgrafo $G(S_{i+1})$ evidentemente $|\xi_{i+1}| = |\xi|$. Si no, de otra manera x_{i+1} se fusiona con alguna componente en Si para formar una nueva componente en S_{i+1}.

Si se utiliza los teoremas anteriores se tiene que $|\xi_{i+1}| \leq |\xi_i|$. Por lo tanto en todos los casos $|\xi_{i+1}| \leq |\xi_i|$ y de aquí $\underset{1\leq i\leq n}{\text{Max}}|\xi_i| \leq |E|$.

Por lo tanto este teorema nos demuestra que la secuencia de grafos cocientes producidos por la eliminación, no requiere más espacio que el de la estructura del grafo original.

La figura que aparece a continuación nos muestra la estructura de datos usados para representar $\text{Ady}_{\mathscr{G}}(C)$, en el grafo cociente $\mathscr{G}$ siendo C={a, b, c}, donde 0 significa el fin de la lista de adyacencia en $\mathscr{G}$.



En el ejemplo, el nodo a se selecciona para hacer el representante de C={a, b, c}. En la implementación es importante seleccionar un representante único para cada $C \in \zeta(S)$.

Sea $x_1, x_2, \ldots, x_n$ la secuencia de nodos, y sea $C \in \zeta(S)$. Elegiremos $x_r \in C$ por ser el representante de C, donde $r = \text{Max}\{ j / x_j \in C\}$, i.e. x_r es el nodo de C que ha sido eliminado en último lugar.

Consideremos ahora como la estructura de adyacencia de $\mathcal{G}_i$ se puede obtener de la de $\mathcal{G}_{i-1}$ cuando se elimina el nodo x_i.

El siguiente algoritmo realiza esta transformación:

Paso 1: (Preparación)

Determinar los conjuntos $T = \text{Ady}_{\mathcal{G}_{i-1}}(x_i) \cap \zeta(S_{i-1})$ y $R = \text{Acces}_{\mathcal{G}_{i-1}}(x_i, \zeta(S_{i-1}))$.

Paso 2: Formación de los nuevos supernodos y particionamiento.

$$\overline{x}_i = \{x_i\} \cup T \quad ; \quad \zeta(S_i) = (\zeta(S_{i-1}) - T) \cup \{\overline{x}_i\}$$

Paso 3: Actualización de la adyacencia.

$$\text{Ady}_{\mathcal{G}_i}(\overline{x}_i) = R$$

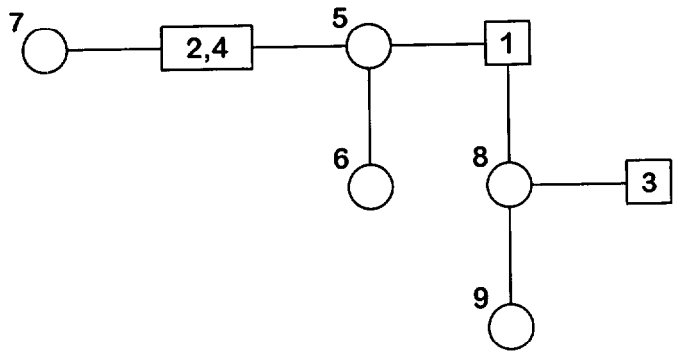
para $y \in R$ $\text{Ady}_{\mathcal{G}_i}(y) = \{\overline{x}_i\} \cup \text{Ady}_{\mathcal{G}_{i-1}}(y) - (T \cup \{x_i\})$

Para ver la realización del algoritmo anterior, tomamos el siguiente ejemplo:

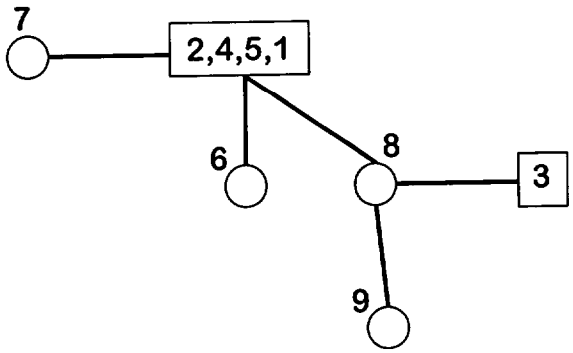
❑ Ejemplo:

Sean los grafos cocientes $\mathscr{G}_4$ y $\mathscr{G}_5$ obtenidos del garfo de la figura 4.5.0:

$\mathscr{G}_4$



$\mathscr{G}_5$



En $\mathscr{G}_4$ se tiene $\zeta(S_4) = \{\overline{x}_1, \overline{x}_3, \overline{x}_4\}$

Aplicando los pasos del algoritmo tenemos:

Paso 1.- (El nodo x_i es x_5)

$$T = \text{Ady}\mathscr{G}_4(x_5) \cap \zeta(S_4) = \{\overline{x}_4, x_6, \overline{x}_1\} \cap \{\overline{x}_1, \overline{x}_3, \overline{x}_4\}$$

$$T = \{\overline{x}_1, \overline{x}_4\}.$$

Paso 2.- El nuevo supernodo es:

$$\overline{x}_5 = \{x_5\} \cup T = \{x_5\} \cup \{\overline{x}_1, \overline{x}_4\} = \{x_5, \overline{x}_1, \overline{x}_4\}$$
$$= \{x_5, \{x_1\}, \{x_2, x_4\}\} \approx \{x_1, x_2, x_4, x_5\}$$

y el nuevo particionamiento es:

$$\zeta(S_5) = (\zeta(S_4) - T) \cup \{\overline{x}_5\} = \{\{\overline{x}_1, \overline{x}_3, \overline{x}_4\} - \{\overline{x}_1, \overline{x}_4\}\} \cup \{\overline{x}_5\} =$$
$$= \{\overline{x}_3, \overline{x}_5\}.$$

Paso 3.- $A = \text{Ady}\ \mathcal{G}_5(\overline{x}_5) = R = \{x_6, x_7, x_8\}$.

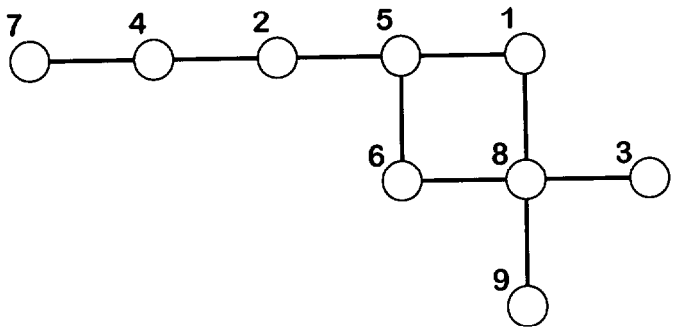
Para y $\in A = \{x_6, x_7, x_8\}$ tenemos:

$$\text{Ady}(x_6) = \{\overline{x}_5\} \cup \text{Ady}_{\mathcal{G}_y}(x_6) - T \cup \{x_5\}) =$$
$$= \{\overline{x}_5\} \cup \{x_5, x_6\} - (\{\overline{x}_1, x_4\} \cup \{x_5\}) = \{\overline{x}_5, x_8\}$$

$$\text{Ady}(x_7) = \{\overline{x}_5\}$$

$$\text{Ady}(x_8) = \{\overline{x}_3, \overline{x}_5, x_6, x_9\}.$$

❑ Vemos con este ejemplo, el efecto producido en la transformación de la estructura de datos del grafo cociente:



ADYACI=	5 8	4 5	8	2 7	1 2 6	5 8	7	1 3 6 9	8

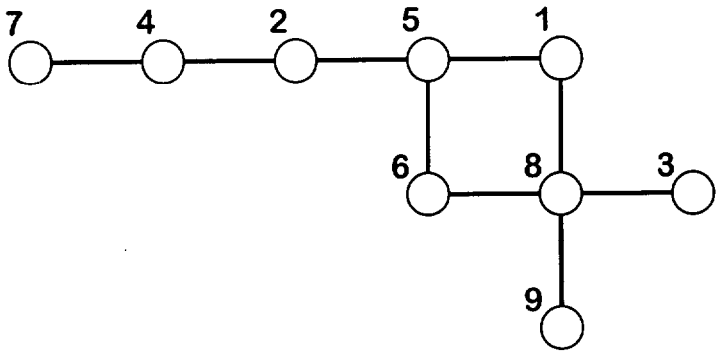
XADJ=(1, 3, 5, 6, 8, 11, 13, 14, 18, 19)

Cada tramos del vector ADYACI representa los nodos vecinos de 1,2, ... ,9 en el grafo anterior.

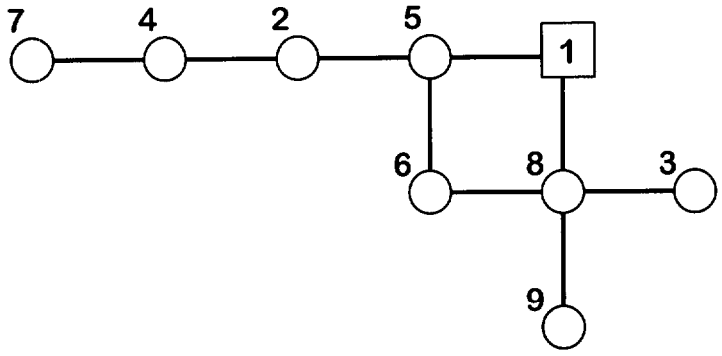
1	5	8		
2	4	5		
3	8			
4	2	7		
5	1	2	6	
6	5	8		
7	4			
8	1	3	6	9
9	8			

Si tomamos de nuevo el grafo anterior y formamos los grafos cocientes:

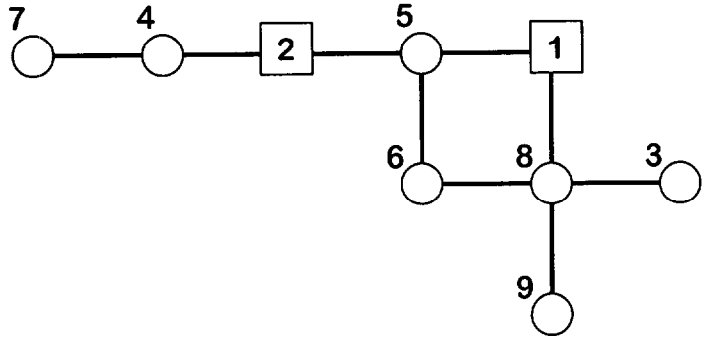
G

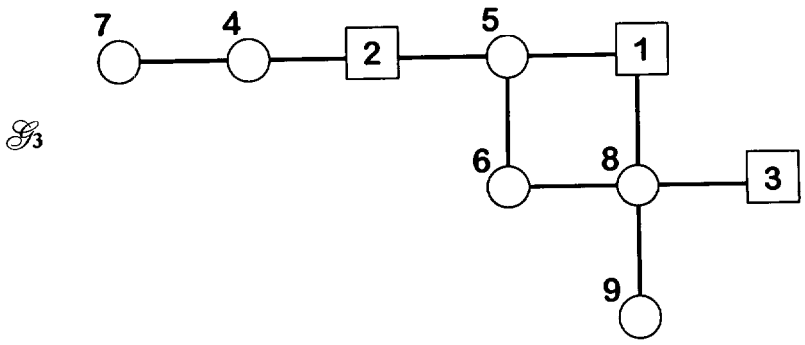


$\mathscr{G}_1$

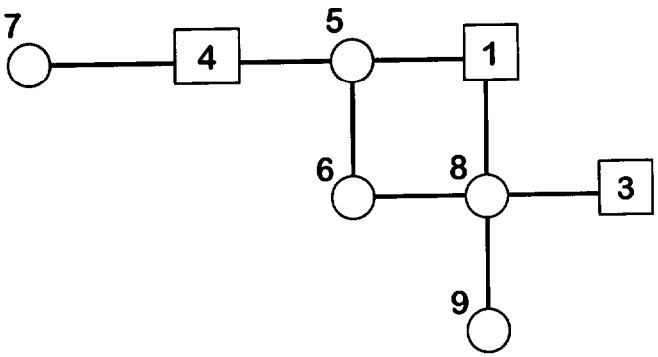


$\mathscr{G}_2$

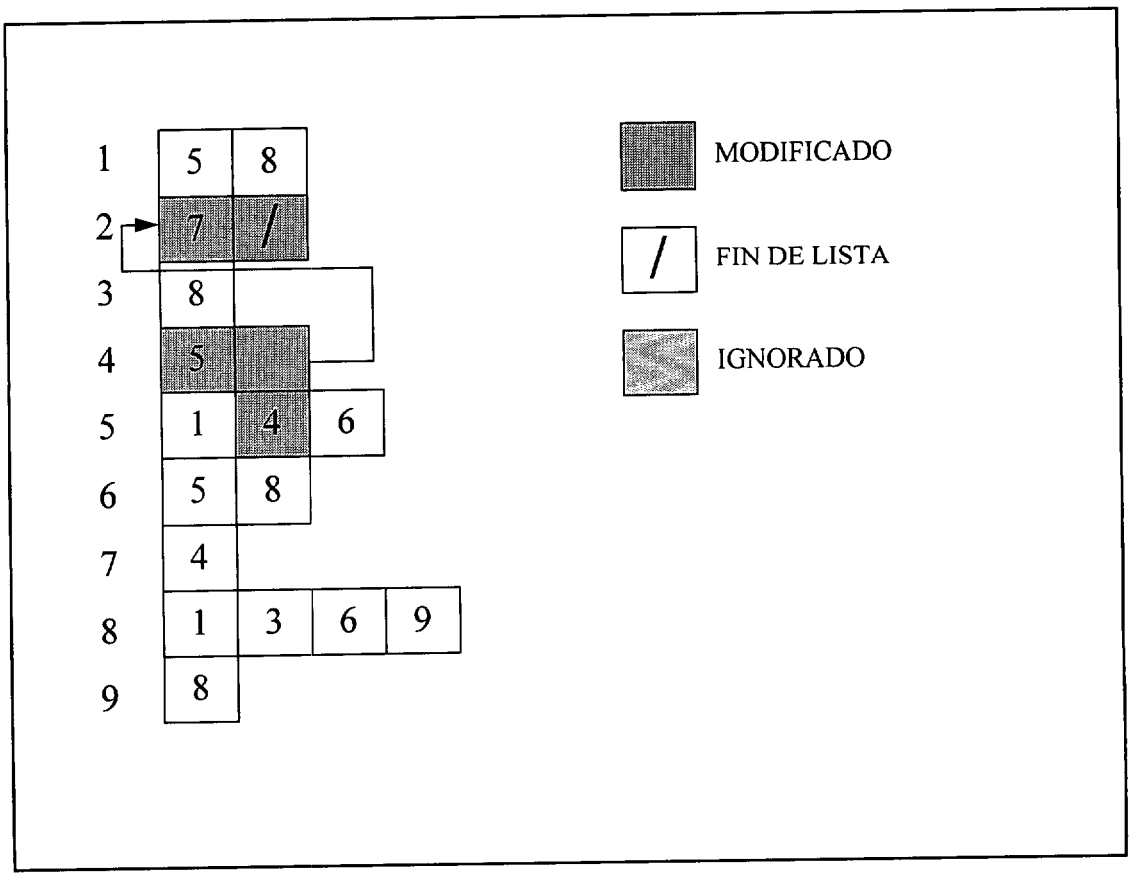


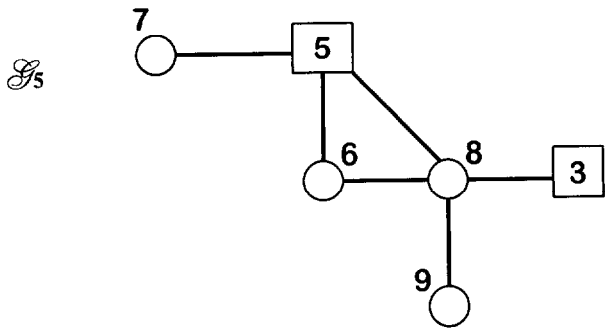


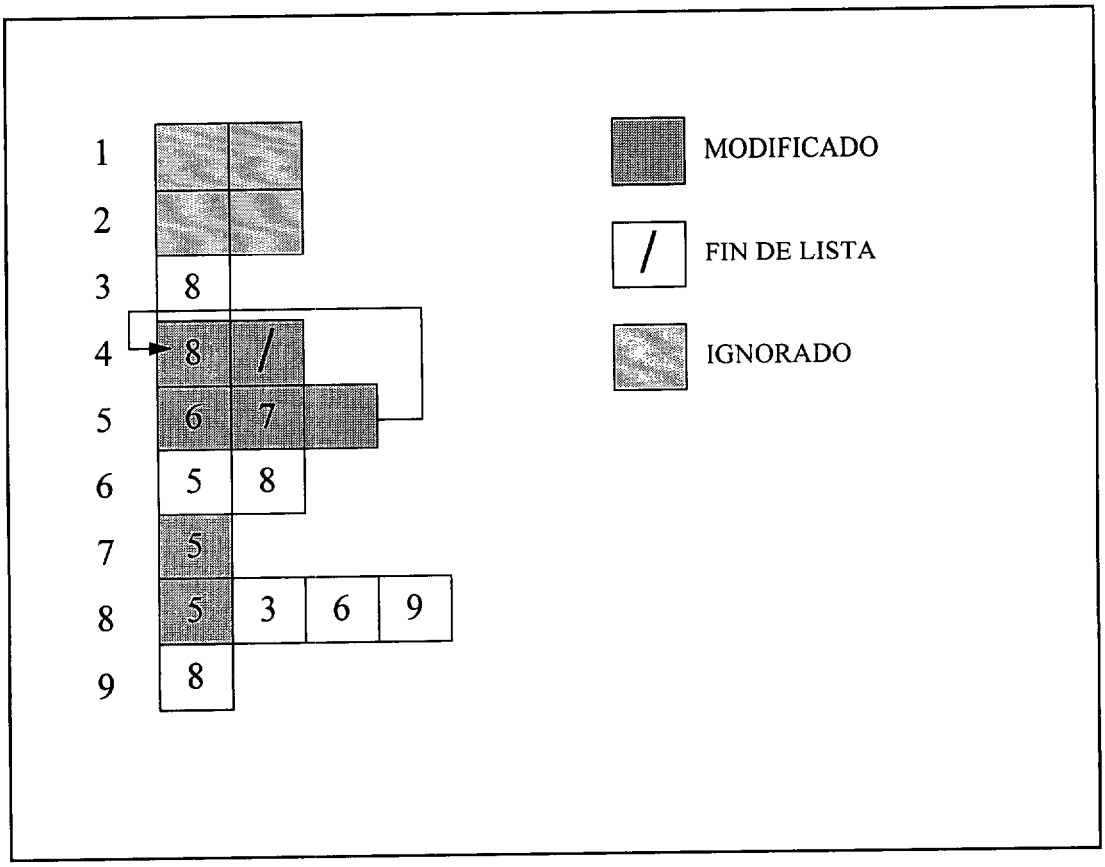
$\mathscr{G}_3$

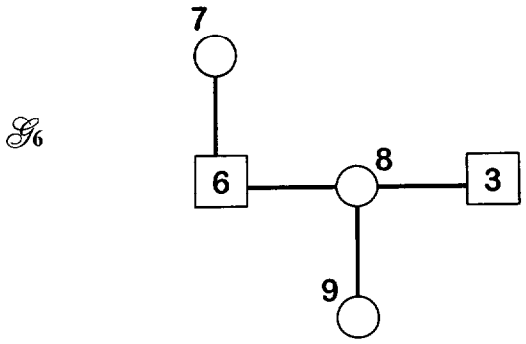


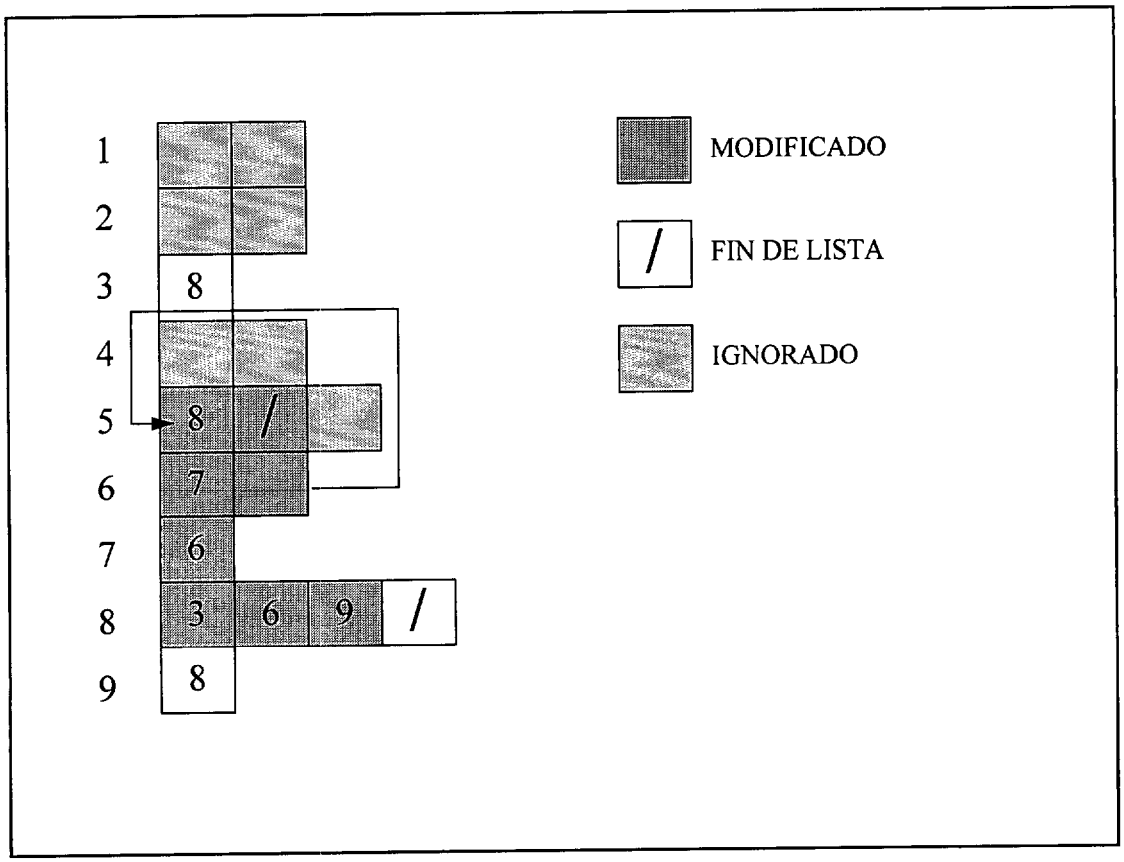
$\mathscr{G}_4$











Veamos la trasformación de los grafos anteriores. La estructura de adyacencia permanece sin cambiar cuando se forman los grafos cocientes $\mathscr{G}_i$, o sea, cuando se forma $\mathscr{G}_1$, $\mathscr{G}_2$ y $\mathscr{G}_3$.

Para trasformar el grafo $\mathscr{G}_3$ en $\mathscr{G}_4$, los nodos x_2 y x_3 se deben unir de forma que en $\mathscr{G}_4$, nuevo conjunto de adyacencia de x_4, contiene a aquellos en el subconjunto $\{x_2, x_4\}$ en el grafo original, habiendose formado el conjunto $\{x_5, x_7\}$.

En la lista de adyacencia del nodo x_5, el nodo vecino x_2 se ha cambiado a x_4 en $\mathscr{G}_4$, cuando el nodo x_4 se convierte en el representante de las componentes del subconjunto $\{x_2, x_4\}$. De la misma manera se forman los grafos cocientes restantes.

Esta forma de representación de los grafos cocientes para la eliminación puede ser usados en la implementación del algoritmo de grado mínimo como veremos más adelante.

Capítulo 5. Algoritmo del Grado Mínimo.

5.0 Introducción.

Sea A una matriz simétrica definida positiva, y sea P una matriz de permutación. Aunque las estructuras no nulas de A y PAP^t son diferentes, sus tamaños son iguales:

$$|\text{Nonul }(A)| = |\text{Nonul }(PAP^t)|$$

Sin embargo el punto crucial es que puede existir una diferencia sustancial entre $|\text{Nonul }(\text{Rell}(A))|$ y $|\text{Nonul }(\text{Rell}(PAP^t)|$ para alguna permutación P.

Idealmente queremos encontrar una permutación P^* que minimice el tamaño de la estructura no nula de la matriz de relleno :

$$|\text{Nonul }((\text{Rell}(P^*AP^{*t}))| = \min |(\text{Rell}(PAP^t)|$$

Así, no existe un algoritmo eficiente para obtener esta P^* óptima para una matriz simétrica en general. Verdaderamente, el problema, cuando A no es simétrica, es muy complicado, es un problema de los llamados NP. Por ello, tenemos que utilizar la heurística que produce un ordenamiento P con una cantidad aceptablemente pequeña, pero no necesariamente mínima $|\text{Nonul}(F(PAP^t)|$.

El esquema de reducción del fill-in más popularmente usado es el algoritmo de grado mínimo (Tinney 1969) que corresponde al esquema de Markowitz (Markowitz 1957) para matrices no simétricas.

El algoritmo de grado mínimo consiste en hacer una renumeración de los nodos en orden creciente de sus grados respectivos.

El algoritmo del grado minimo se basa en la siguiente observación:

Supongamos que $\{x_1,.. . x_{i-1}\}$ han sido enumerados. Se fija el número de no ceros en el grafo relleno para estas columnas. Para reducir el número de no ceros en la i-ésima columna, parece que debe factorizarse la matriz remanente, la columna con la menor cantidad de no ceros debe moverse para convertirse en la columna i. En otras palabras, el esquema debe considerarse como un método que reduce el relleno de una matriz por una minimización local de $\eta(L_{*_i})$ en la matriz factorizada.

El algoritmo de grado mínimo ejecuta una renumeración de los nodos en orden creciente de sus grados respectivos. Los nodos que tengan mayor grado originarán, en teoría, un mayor relleno. La idea sustancial es renumerarlos al final para así evitar el incremento del fill-in durante el proceso.

5.1 El algoritmo básico.

El algoritmo del grado mínimo puede ser descrito fácilmente en términos de ordenamiento de un grafo simétrico. Sea $G_0 = (X, E)$ un grafo no enumerado. Usando el modelo del grafo de eliminación, el algoritmo básico es como sigue:

Paso1. Inicialización $i \leftarrow 1$.

Paso 2. (Selección del grado mínimo).
En el grafo de eliminación $G_{i-1} = (X_{i-1}, E_{i-1})$, se elige el nodo x_i de grado mínimo en G_{i-1}.

Paso 3. (Transformación del grafo).
Se forma el nuevo grafo de eliminación $G_i = (X_i, E_i)$, eliminando el nodo x_i de G_{i-1}.

Paso 4. (Bucle o parada) $i \leftarrow i+1$. Si $i > |X|$, parar. Si no, ir al Paso 2.

Sea el grafo de la figura 5.1.0:

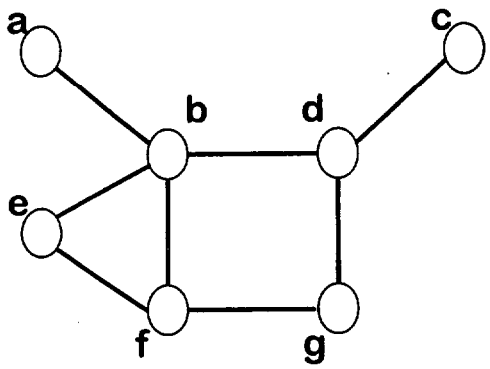
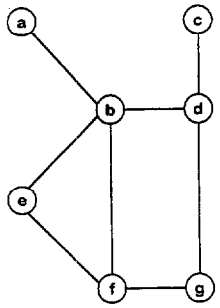
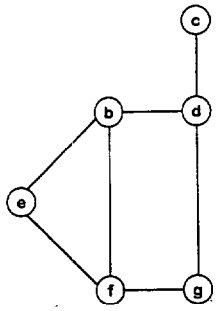
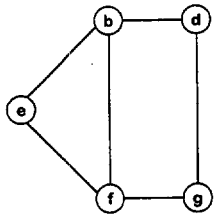
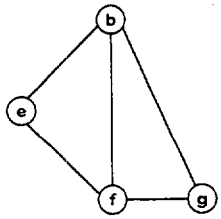
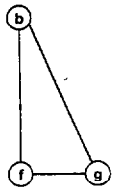
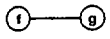
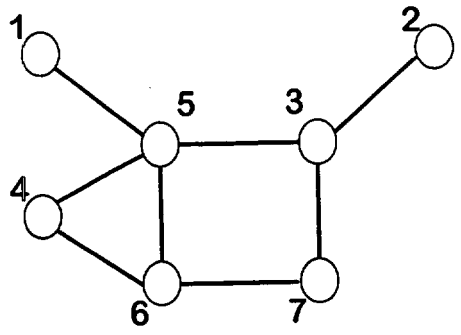


Fig.5.1.0

Si a este grafo le aplicamos el algoritmo del grado mínimo paso a paso se obtiene los siguientes grafos de eliminación, así como los nudos seleccionados quedando el grafo ordenado según dicho algoritmo de la siguiente manera:

Etapa k	Grafo de Eliminación G_{k-1}	Nudo seleccionado	Grado
1		a	1
2		c	1
3		d	2
4		e	2
5		b	2
6		f	1
7		g	0

Después de aplicar el algoritmo del grado mínimo al grafo de la figura 5.1.0, el grafo ordenado es el siguiente:

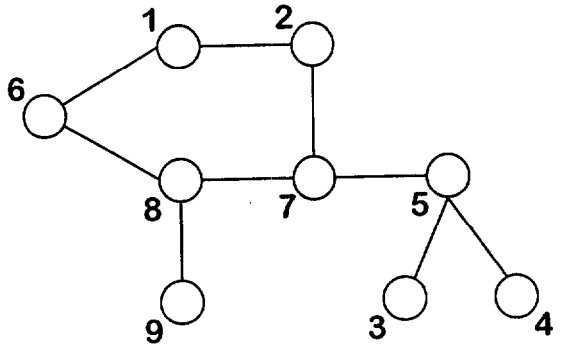


Veamos otro ejemplo de formación de la sucesión de grafos de eliminación de una matriz A simétrica y definida positiva. Sea A la matriz de la figura 5.1.1:

$$A = \begin{bmatrix} 1 & & & & & & & & \\ x & 2 & & & & & & & \\ & & 3 & & & & & & \\ & & & 4 & & & & & \\ & & x & x & 5 & & & & \\ x & & & & & 6 & & & \\ & x & & & x & & 7 & & \\ & & & & & x & x & 8 & \\ & & & & & & & x & 9 \end{bmatrix}$$

Fig 5.1.1.

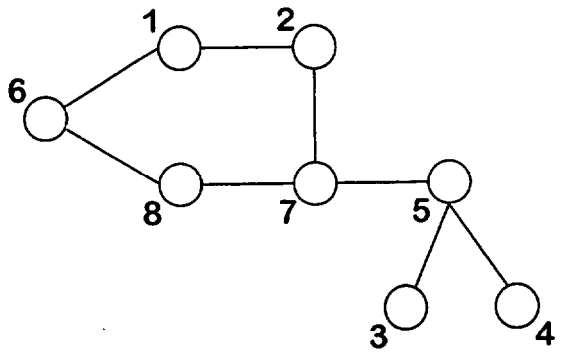
El grafo G_0 asociado a esta matriz y la lista de nodos con sus grados es:



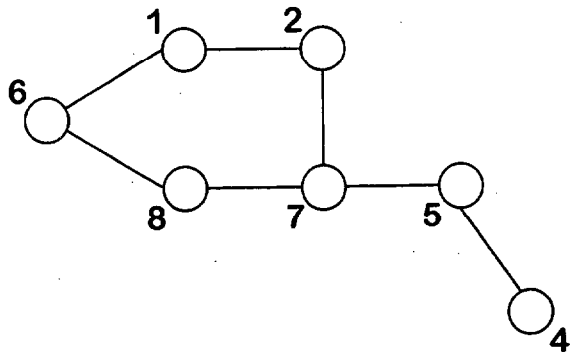
Nodos	Grado
1	2
2	2
3	1
4	1
5	3
6	2
7	3
8	2
9	1

Elegimos un nodo i de grado mínimo:

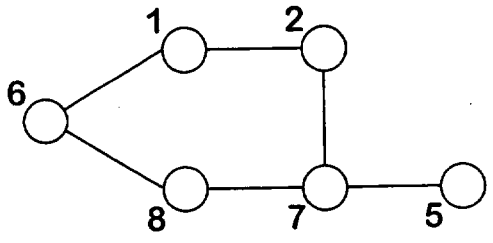
Sea $i=1=9$ $\delta(9)=1$ y eliminando dicho nodo y sus aristas incidentes queda el grafo de eliminación G_1:



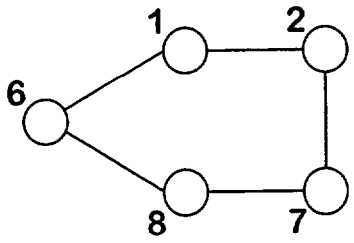
Sea $i=2=3$ $\delta(3)=1$



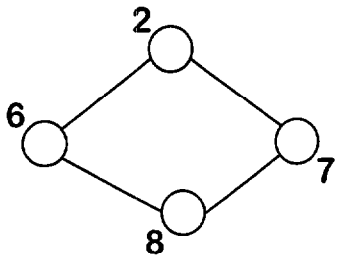
Sea $i=3=4$ $\delta(4)=1$



Sea i=4=5 $\delta(5)=1$

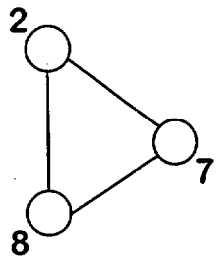


Sea i=5=1 $\delta(1)=2$



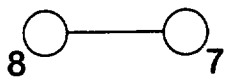
Al eliminar el nodo 1 se ha producido un relleno de huecos (efecto fill-in) en la posición 2-6.

Sea i=6=6 $\delta(6)=2$



Al eliminar el nodo 6 se ha producido un relleno de huecos (efecto fill-in) en la posición 2-8.

Sea i=7=2 $\delta(2)=2$



Sea i=8=7 $\delta(7)=1$

La sucesión G_1, G_2, G_3, G_4, G_5, G_6, G_7 y G_8 forman la sucesión de grafos de eliminación quedando la matriz A como sigue:

$$A = \begin{bmatrix} 9 & & & & & & & \\ & 3 & & & & & & \\ & & 4 & & & & & \\ & x & x & 5 & & & & \\ & & & & 1 & & & \\ & & & & x & 6 & & \\ & & & & x & \otimes & 2 & \\ & & & x & x & & & 7 \\ x & & & & & & \otimes & x & 8 \end{bmatrix}$$

Se observa en la nueva ordenación de la matriz A que se han producido dos rellenos de huecos señalados con (⊗).

Puede ocurrir que puede existir más de un nodo con el grado mínimo en un paso particular. Aquí rompemos los nudo arbitrariamente. Sin embargo, diferentes estrategias de ruptura de nudos nos dan diferentes versiones del algoritmo de grado mínimo, como veremos más adelante.

La idea que preside al algoritmo de grado mínimo es sencilla: como en cada etapa k del proceso de eliminación de Gauss, debido a manipulaciones inherentes al proceso que se efectúa en la submatriz activa, si la fila k tiene elementos no nulos a la derecha del elemento de la diagonal principal, al sumar un múltiplo de ésta fila a cualquiera de las filas k + 1 a n, donde se quiera hacer cero un elemento de la columna k por de bajo de la diagonal principal, se pueden producir elementos no nulos en esas filas, si se examina qué fila de la submatriz activa, l, tiene menor número de elementos distintos de cero y se intercambian las filas l y k y las columnas l y k, en esa submatriz activa se crearán el mínimo posible de elementos rellenos.

En la implementación de dicho algoritmo como se trabaja con grafos de eliminación, esto nos permite conocer al final del proceso qué nuevos elementos se harán distintos de cero al efectuar la correspondiente factorización.

5.2 Algoritmo de grado mínimo usando conjuntos accesibles.

El uso de los grafos de eliminación en el algoritmo de grado mínimo nos da un mecanismo mediante el cual seleccionamos el próximo nodo que será remunerado. Cada paso del algoritmo involucra una transformación del grafo, lo cual es la parte más cara del algoritmo en términos de implementación. Estas transformaciones pueden eliminarse si podemos dar un camino alternativo para calcular los grados de los nodos en el grafo de eliminación [22].

Con los conjuntos accesibles podemos aplicar el siguiente algoritmo:

Paso1. (Inicialización) $S \leftarrow \varnothing$. $\delta(x) \leftarrow |Ady(x)|$, para $x \in X$.

Paso 2. (Selección del grado mínimo). Seleccione un nodo $y \in X - S$ donde $\delta(y) = \underset{x \in X-S}{Min}\, \delta(x)$. Numere el nodo y próximo y fije $T \leftarrow S \cup \{y\}$.

Paso 3.(Actualización del grado). $\delta(u) \leftarrow |Acces(u,T)|$ para $u \in X\text{-}T$.

Paso 4. (Bucle o parada). Si $T = X$, parar. Si no fije $S \leftarrow T$ y vaya al Paso 2.

Este camino usa la estructura del grafo original a través de todo el proceso. Verdaderamente, el algoritmo puede ejecutarse con la estructura de adyacencia solamente de $G_0=(X, E)$.

Es apropiado apuntar aquí que en el paso de actualización del grado, no es necesario recalcular los tamaños de los conjuntos accesibles para cada nodo en X - T, puesto que la mayoría de ellos permanecen sin cambios. Esta observación se formaliza en el siguiente lema:

❑ **Lema :**

Sea S el conjunto de nodos eliminados e $y \notin S$ y $T = S \cup \{y\}$. Entonces:

$$Acces(x, T) = \begin{cases} Acces(x, S) & \text{para } x \notin Acces(y, S) \\ Acces(x; S) \cup Acces(y, S) - \{x, y\} & \text{si } x \in Acces(y, S) \end{cases}$$

Sean los grafos de la figura 5.2.0 :

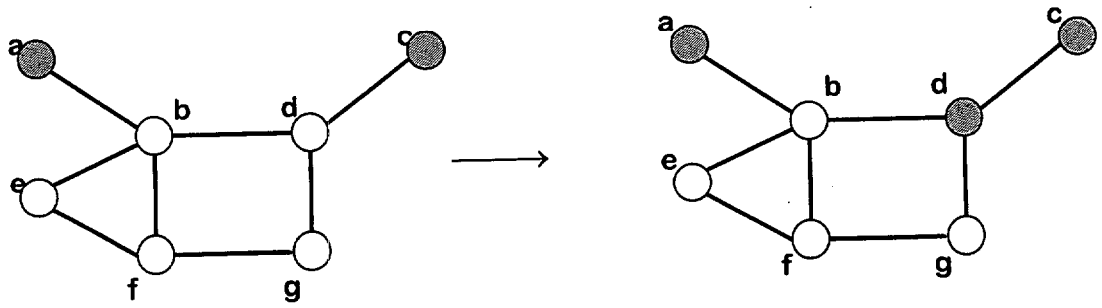


Fig.5.2.0

Si en la figura 5.2.0. tomamos como $S=\{a, c\}$ entonces tenemos que Acces(d, $S)=\{b, g\}$. Por tanto la eliminación del nodo d sólo afecta a los grados de los nodos b y g.

Mediante esta observación el paso 3 en el algoritmo queda como sigue:

Paso 3. (Actualización del grado)

$\delta(u) \leftarrow |Acces(u,T)|$, para $u \in$ Acces (y, S).

Como plantea el algoritmo, se numera un nodo cada vez que se ejecuta el lazo. Sin embargo, cuando se encuentra un nodo y de grado mínimo en el Paso 2, es posible a menudo que se detecte el subconjunto de los nodos que se remunerarán a continuación automáticamente sin ejecutar ninguna búsqueda de grado mínimo.

Comencemos el estudio introduciendo una relación de equivalencia. Consideremos la etapa de eliminación del proceso, donde S es el conjunto de los nodos eliminados. Se dice que dos nodos $x, y \in X - S$ son indistinguibles con respecto a la eliminación si:

$$Acces(x, S) \cup \{x\} = Acces (y, S) \cup \{y\}$$

En el siguiente ejemplo de la figura 5.2.1. el subconjunto S contiene 36 nodos sombreados (Es el grafo que ha quedado, cuando el algoritmo del grado mínimo se aplica al grafo).

Los nodos a,b y c son indistinguibles con respecto a la eliminación desde que Acces (a, S) ∪ {a}, Acces (b, S) ∪ {b} y Acces (c, S) ∪ {c} sean todos iguales a {a, b, c, d, e, f, g, h, j, k}.

En la figura existen dos grupos más que se pueden identificar como indistinguibles. Estos son {j, k} y {f, g}.

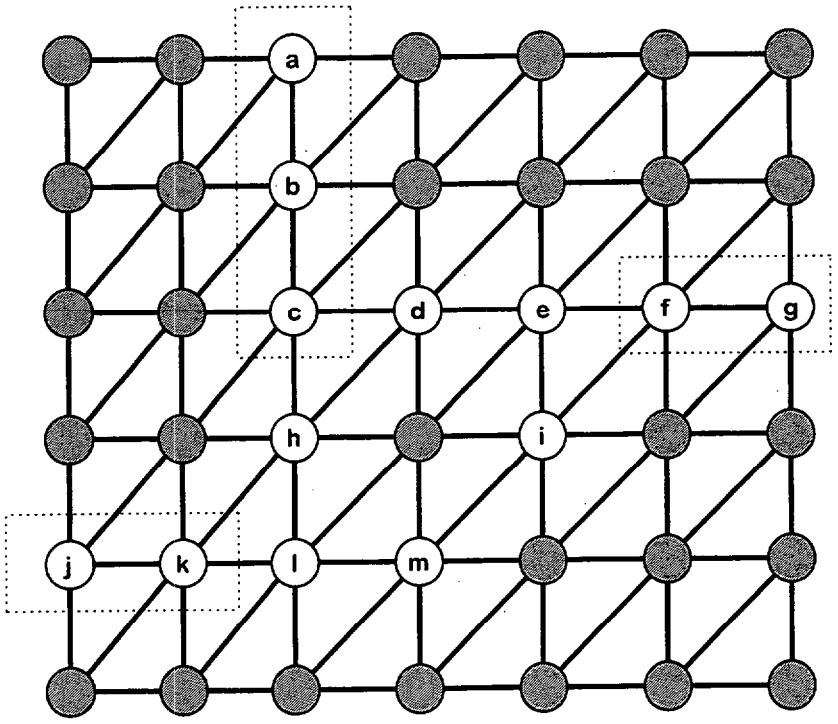


Fig. 5.2.1.

Estudiemos ahora la implicación de esta relación de equivalencia y su papel en el algoritmo de grado mínimo. Como veremos después, esta notación puede usarse para acelerar la ejecución del algoritmo de grado mínimo.

❑ **Teorema.**

Sea x, y ∈ X - S. Si Acces (x, S) ∪ {x} = Acces (y, S) ∪ {y}, entonces para todo X- {x, y} ⊃ T ⊃ S, se verifica que : Acces (x, T) ∪ {x} = Acces (y, T) ∪ {y}.

❍ **Demostración:**

Obviamente, x ∈ Acces (y, S) ⊂ Acces (y, T) ∪ T, de forma tal que x ∈ Acces (y, T). Ahora queremos demostrar que Acces (x, T) ⊂ Acces (y, T) ∪ {y}. Consideremos z ∈ Acces (x, T). Existe un camino $\{x, s_1, \ldots, s_t, z\}$ y éste camino obtenido verifica lo siguiente para el conjunto donde $\{s_1, \ldots, s_t\} \subset T$. Si todo $s_i \in S$, no hay nada que probar. Por otra parte, sea s_i el primer nodo en $\{s_1, \ldots, s_t\}$ que no en S, esto es: $s_i \in$ Acces (x, S) ∩ T.

Esto implica $s_i \in$ Acces (y, S) y por tanto z ∈ Acces (y, S). Entonces tenemos:

$$\text{Acces } (x, T) \cup \{x\} \subset \text{Acces } (y, T) \cup \{y\}.$$

❑ **Corolario.**

Sea x, y indistinguibles respecto al subconjunto S. Entonces, para T ⊃ S,

$$|\text{Acces } (x, T)| = |\text{Acces } (y, T)|$$

En otras palabras, si dos nodos se convierten en indistinguibles en alguna etapa de eliminación, permanecerán indistinguibles hasta que uno de ellos sea eliminado. De ésta manera, el siguiente teorema nos muestra que pueden ser eliminados juntos en el algoritmo de grado mínimo.

❑ **Teorema.**

Si dos nodos se vuelven indistinguibles en alguna etapa del algoritmo de grado mínimo, pueden ser eliminados juntos en el algoritmo.

❍ **Demostración:**

Sea x, y indistinguibles después de la eliminación del subconjunto S. Si asumimos que x se convierte en un nodo de grado mínimo después de que el conjunto T ⊃ S sea eliminado, tenemos que:

$$|\text{Acces } (x, T)| \leq |\text{Acces } (z, T)| \text{ para todo } z \in X - T.$$

$$|\text{Acces}(y, T \cup \{x\})| = |\text{Acces}(y, T) - \{x\}| = |\text{Acces}(y, T)| - 1$$
$$= |\text{Acces}(x, T)| - 1$$

Por tanto, para todo $z \in X - T \cup \{x\}$ se verifica que:

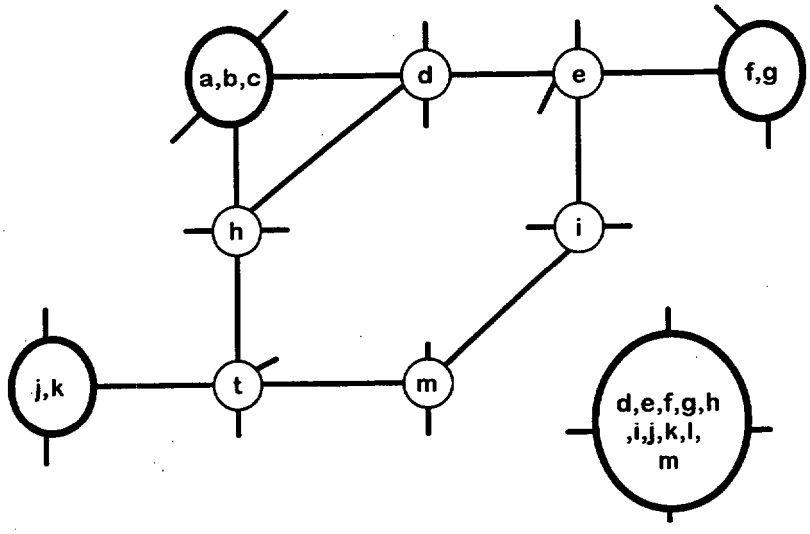
$$|\text{Acces}(y, T \cup \{x\})| \leq |\text{Acces}(z, T)| - 1 \leq |\text{Acces}(z, T - \{x\}|$$

De ésta manera, después de la eliminación del nodo x, el nodo y se convierte en un nodo de grado mínimo.

Estas observaciones pueden explotarse en la implementación del algoritmo de grado mínimo. Después de ejecutar la búsqueda de grado mínimo para determinar el próximo nodo $y \in X - S$ a eliminar, podemos enumerar inmediatamente después de y el conjunto de nodos indistinguibles de y.

Por lo tanto, en el paso de actualización del grado, puede reducirse el trabajo puesto que los nodos indistinguibles tienen el mismo grado en el grafo de eliminación. Una vez que se identifican los nodos como indistinguibles, pueden ser pegados y tratados como un supernodo simple.

En la siguiente grafo que viene de la figura 5.2.1 se muestran dos etapas en la eliminación donde los supernodos se forman de nodos indistinguibles. Después de la eliminación del conjunto indistinguible {a, b, c}, todos los nodos tienen conjuntos alcanzables idénticos de tal forma que se pueden mezclar en uno.



En general, identificar los nodos indistinguibles por medio de la definición consume mucho tiempo. A continuación presentamos un lema que es muy efectivo para la identificación de los nodos indistinguibles.

Sea G= (X, E) y S el conjunto de los nodos eliminados. Sea G (C_1) y G (C_2) dos componentes conexas en el subgrafo G(S), o sea C_1, $C_2 \in C(S)$.

❑ **Lema :**

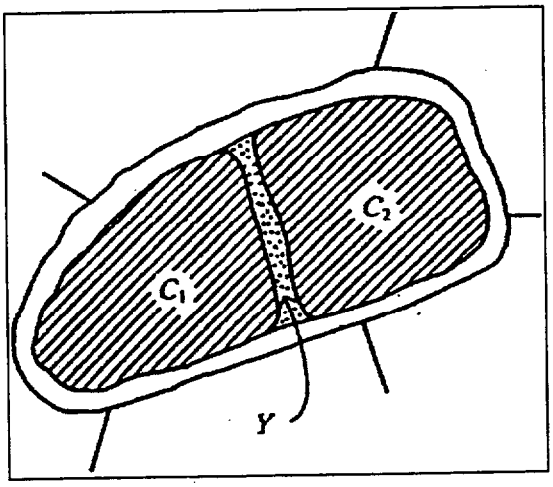
Sea R_1 = Ady (C_1), y R_2 = Ady (C_2). Si $y \in R_1 \cap R_2$. y $Ady(y) \subset R_1 \cup R_2 \cup C_1 \cup C_2$, entonces Acces (y, S) $\cup$ {y} = $R_1 \cup R_2$.

○ **Demostración**

Sea $x \in R_1 \cup R_2$. Si asumimos que $x \in R_1$ = Ady (C_1), como G (C_1) es una componente conexa en G (S), entonces podemos encontrar un camino de y a x a través de $C_1 \subset S$. Por lo tanto, $x \in$ Acces (y, S) $\cup$ {y}.

Por otra parte, $y \in R_1 \cup R_2$ por definición. Más aún, si $x \in$ Acces (y, S), existe un camino de y a x a través de S, dado por y, s_1 ,.., s_t, x.

Si t =0, entonces $x \in$ Ady (y) - $S \subset R_1 \cup R_2$. Por otra parte, si t > 0, $s_1 \in$ Ady (y) $\cap$ $S \subset C_1 \cup C_2$. Según esto tenemos que $\{s_1, ..., s_t\}$ es un subconjunto tanto de C_1 como de C_2 de forma tal que $x \in R_1 \cup R_2$. Así Acces (y, S) $\cup$ {y} $\subset R_1 \cup R_2$.



Sea C_1 , C_2 , R_1 y R_2 los conjuntos definidos anteriormente, entonces los nodos en $Y = \{y \in R_1 \cap R_2 \,/\, Ady(y) \subset R_1 \cup R_2 \cup C_1 \cup C_2\}$ son indistinguibles con respecto al conjunto eliminado S.

❑ **Corolario:**

Para $y \in Y$, $|Acces\ (y, S)| = |R_1 \cup R_2| - 1$, se tiene que puede usarse para mezclar nodos indistinguibles en la intersección de don conjuntos accesibles R_1 y R_2. La prueba puede hacerse simplemente inspeccionando el conjunto adyacente de nodos en la intersección $R_1 \cap R_2$.

Esta noción de nodos indistinguibles puede aplicarse al algoritmo de grado mínimo. El nuevo algoritmo realizado puede exponerse como sigue:

Paso 1. (Inicialización) $S \leftarrow \varnothing$.

$\delta(x) = |Ady\ (x)|$, para $x \in X$

Paso 2. (Selección). Seleccionar un nodo $y \in X - S$ tal que:

$$\delta(y) = \min_{x \in X-S} \delta(x)$$

Paso 3. (Eliminación). Numere los nodos en:

$Y = \{x \in X - S \,/\, x$ es indistinguible desde $y\}$ en el próximo ordenamiento.

Paso 4. (Actualización del grado). Para $u \in Acces\ (y, S) - Y$

$$\delta\ (u) = |Acces\ (u, S \cup Y)|$$

e identificar los nodos indistinguibles en el conjunto Acces (y, S) - Y

Paso 5. (Bucle o Parar). Fije $S \leftarrow S \cup Y$. Si $S = X$, Parar. Si no , ir al Paso 2.

5.3 Implementación del Algoritmo de grado mínimo.

La implementación del algoritmo de grado mínimo que presentamos aquí, incorpora la noción de nodos indistinguibles como se describió en las secciones previas. Los nodos identificados como indistinguibles se mezclan para formar un supernodo. Serán tratados esencialmente como un nodo en el algoritmo. Ellos muestran la misma adyacencia, el mismo grado, y pueden eliminarse juntos en el algoritmo. En la implementación, este supernodo puede referenciarse por un representante del conjunto.

El algoritmo requiere la determinación de los conjuntos accesibles para la actualización del grado. El modelo de grafo cociente se usa con el propósito de mejorar la eficiencia total del algoritmo. En efecto, los nodos conectados que son eliminados, se mezclan y se utiliza su representación en la secuencia del grafo cociente.

Debe enfatizarse que la idea de cociente (unir en supernodos) se aplica aquí en dos contextos diferentes.

a) Eliminación de los nodos conectados para facilitar la determinación de los conjuntos accesibles.

b) la no eliminación de los nodos indistinguibles para acelerar la eliminación.

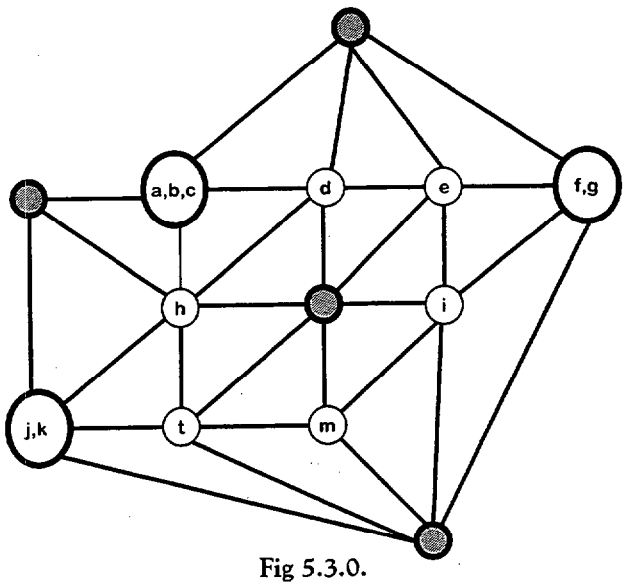


Fig 5.3.0.

En la figura 5.3.0 se observa los pasos a y b señalados. Los nodos que están vacíos son nodos eliminados mientras que el resto permanecen como nodos indistinguibles.

Ahora describimos un conjunto de subrutinas que implementan el algoritmo del grado mínimo .

El grafo G=(X, E) se almacena usando los vectores (ADYACY, XADJ) y el número de variables por NEQNS. EL ordenamiento de grado mínimo resultante se almacena en el vector PERM, mientras que INVP nos da el ordenamiento inverso.

Este conjunto de subrutinas requiere algunos vectores de trabajo para implementar el modelo de grafo cociente y la idea de nodos indistinguibles.

Los grado de los nodos en el grafo de eliminación se guardan en el arreglo DEG. El valor de DEG para nodos que se eliminan en el conjunto es -1.

En la representacion de la secuencia de grafos cocientes , los nodos conectados se mezclan para formar el supernodo. Es suficiente seleccionar un representante del supernodo, si G (C) es una componente conexa, siempre seleccionamos el nodo $x \in C$ como último que se elimina de los representantes de C, esto implica que los nodos restantes en C pueden ignorarse en los sucesivos grafos cocientes.

El vector MARKER se usa para marcar aquellos nodos que pueden ignorarse en la estructura de adyacencia. Los valores de MARKER para cada nodo se fijan en -1 .También este vector se usa temporalmente para facilitar la generación de los conjuntos accesibles.

Se usan dos arreglos más, QSIZE y QLINK para completar la especificación de los supernodos indistinguibles.Si el nodo y es representante, el número de nodos en éste supernodo está dado por QSIZE (i) y los nodos están dados por :i, QLINK (i), QLINK(QLINK (i)), ...

La figura 5.3.1 muestra el uso de los vectores QSIZE, QLINK y MARKER.

Los nodos {2,5,8} forman un supernodo indistinguible representado por el nodo 2. Así los valores de MARKER de 5 y 8 son -1. Por otra parte {3,6,9} forman un supernodo eliminado. Su representante es el nodo 9 de forma tal que MARKER(3) y MARKER (9) son -1.

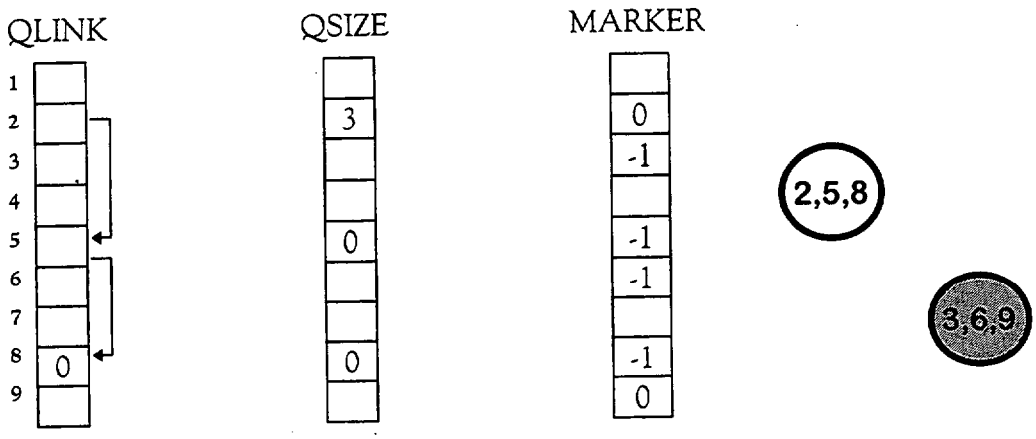


Fig 5.3.1

5.4 Evolución del algoritmo de grado mínimo.

5.4.0 Introducción.

Consideremos un sistema de ecuaciones n x n , A x = b donde la matriz A es simétrica , definida positiva y sparse .

Como sabemos cuando se factoriza A por Cholesky ésta sufre un relleno, o sea elementos que en A son ceros, en el factor L son distintos de cero, éste es el efecto fillin, como hemos descrito anteriormente.

Antes de factorizar la matriz A debemos hacer un ordenamiento de dicha matriz para minimizar el relleno o sea minimizar el efecto fill-in .

Como sabemos el algoritmo de Markowitz comienza con una matriz dada y a cada paso de la eliminación Gaussiana, las filas y las columnas se permutan para minimizar el producto de no ceros fuera de la diagonal en la fila y columna pivotes. De esta forma minimizamos los cálculos a realizar en cada paso de la eliminación Gaussiana.

Esta estrategia de minimizacion local no da en general un mínimo global, Tinney y Walker emplearon una estrategia similar para resolver grandes sistemas sparse, en el análisis de sistemas de potencia , éstos sistemas tienen estructura simétrica y no necesitan intercambios para lograr la estabilidad numérica , dado que los pivotes se toman de la diagonal de A [23].

Rose desarrolló un modelo empleando grafos y junto con la estrategia de Markowitz y Tinney, desarrolló el algoritmo heurístico del grado mínimo.

El algoritmo del grado mínimo sólo trabaja con la estructura de la matriz A y simula los n pasos de la eliminación Gaussiana .En cada paso se intercambia una fila y su correspondiente columna en la parte de la matriz que queda por factorizar de tal forma que el número de no ceros en la fila y columna pivotes se minimice.

Después de n pasos, se ha simulado toda la factorización y el orden en que se seleccionaron las filas y columnas pivote, es el nuevo ordenamiento.

Como sabemos el ordenamiento de grado mínimo puede ser descrito en términos de grafos de eliminación .La selección de un nodo de grado mínimo se selecciona en el nuevo grafo de eliminación .

El grado de un nodo puede cambiar después de la transformación del grafo debido a la eliminación de bordes que inciden en el nodo eliminado.

El algoritmo de grado mínimo básico no está especificado completamente ya que en etapa de selección del nodo puede haber muchos nodos de grado mínimo. Lo que se hace es elegir el próximo nodo a eliminar de una forma arbitraria.

La calidad resultante del orden de grado mínimo depende mucho de la selección correcta del conjunto de nodos de grado mínimo. Existen varias estrategias para la buena elección del nodo siguiente en la eliminación, como veremos a continuación.

5.4.1 Calidad del orden de grado mínimo.

El orden de grado mínimo en una estructura de árbol será aquel que tenga un orden perfecto, o sea sin ceros y el ordenamiento será de mínimo relleno de huecos.

En la descripción del algoritmo de grado mínimo básico se observa que la transformación del grafo de eliminación $G \leftarrow G_y$ es un paso central en la implementaron de dicho algoritmo. En general, éste paso de eliminación crea rellenos de huecos dentro de los nodos adyacentes al nodo y.

También se tiene que los grados de los nodos adyacentes a y pueden cambiar, por lo que se requiere un recálculo de sus grados en la preparación del paso de selección del próximo nodo. Como se sabe la actualización del grado es el paso que más tiempo consume en el algoritmo.

5.4.2 Eliminación de masa.

En problemas de elementos finitos se observa que en la eliminación de un nodo y de grado mínimo, a veces hay un subconjunto de nodos adyacentes a y que pueden eliminarse inmediatamente después de y o sea que si se selecciona y como el nodo de grado mínimo en el grafo G, entonces el subconjunto:

$Y = \{z \in Ady(y) / \delta_{G_y}(z) = \delta_G(y) - 1\}$ puede seleccionarse la próxima vez en cualquier orden.

Esto permite evitar algunas transformaciones del grafo y pasos de actuación del grado, ya que nos da un conjunto de nodos de grado mínimo que pueden seleccionarse como próximos, ya que al hacer la transformación $G \leftarrow G_y$ y actualizar el grado, podemos eliminar nodos en $Y \cup \{y\}$ simultáneamente.

Esto implica que el grafo de eliminación y la actualización del grado sólo necesita hacerse una vez para todo el conjunto en vez de $|Y \cup \{y\}|$. Debido a esto hay también un ahorro de tiempo en la selección del nodo siguiente.

Como ejemplo de la eliminación de masa tomemos un modelo de un problema de una malla regular 5 x 5 usando 9 puntos de conectividad.

Supongamos que una vez aplicado el algoritmo del grado mínimo a la malla tenemos estas dos ordenaciones distintas:

4	8	11	7	3
20	21	22	16	15
10	14	23	13	9
6	18	24	25	5
2	17	12	19	1

4	7	21	12	3
11	16	22	15	8
20	19	23	18	17
6	14	24	13	9
2	10	25	5	1

Después de haber eliminado 14 nodos durante el ordenamiento, nodos que están marcados con • queda :

•	•	•	•	•
20	21	22	16	15
•	•	23	•	•
•	18	24	25	•
•	17	•	19	•

El conjunto de nodos adyacentes por ejemplo a $y=15$ en el grafo de eliminación es $\{16, 19, 20, 21, 22, 23, 24, 25\}$. Estos son nodos eliminados que pueden alcanzarse desde el nodo y vía nodos eliminados, como se observa el nodo $y=15$ tiene de grado mínimo que es 8 y el nodo $z=16$ puede eliminarse al mismo tiempo que y luego el conjunto de nodos $\{y, z\}$ pueden eliminarse juntos.

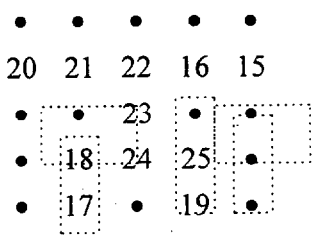
5.4.3 Nodos indistinguibles.

Sea y un nodo de grado mínimo en el grafo G y sea z un nodo tal que $z \in Ady_G(y)$, entonces se tiene que : $Ady_{G_y}(z) = (Ady_G(y) \cup Ady_G(z) - \{y, z\}$.

Decimos que dos nodos u y v son indistinguibles en G(y) si $Ady_G(u) \cup \{u\} = Ady_G(v) \cup \{v\}$. EL conjunto $Ady\{u\} \cup \{u\}$ se refiere a la vecindad de u , por lo tanto dos nodos con vecindades iguales son indistinguibles.

Si dos nodos son indistinguibles en G también son indistinguibles en Gy, es obvio que si dos nodos son indistinguibles entonces sus grados son los mismos, también se tiene que si dos nodos se convierten en indistinguibles en alguna etapa de la eliminación, entonces tendrán la misma vecindad y por lo tanto el mismo grado, luego esto implica que pueden ser eliminados juntos siempre y cuando uno de ellos se seleccione para la eliminación.

A medida que se avanza en la aplicación del algoritmo del grado mínimo pueden aparecer nodos indistinguibles juntos y ser tratados como tales. De esta forma, sólo necesitamos considerar una representacion para cada grupo de nodos indistinguibles.En el ejemplo anterior, después de realizar esta operación quedan cinco supernodos.



Aquí tenemos que los once nodos que quedan pueden dividirse en cinco grupos de nodos indistinguibles como son: {15, 16}, {17, 18}, {19, 25}, {20, 21} y {22, 23, 24}. Se observa que ahora se puede trabajar con un grafo de cinco super nodos.

5.4.4 Actualización de grado incompleta.

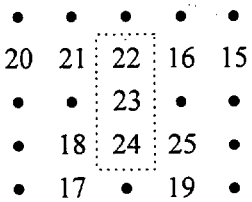
El uso de los nodos indistinguibles ayuda no solo a la eliminación sino también al tiempo de actualización del grado, de ésta manera mezclando los nodos indistinguibles, solo necesitamos recalcular los grados de los nodos representativos.

Esta técnica acelera al algoritmo del grado mínimo ya que se evita el cálculo de los grados de los nodos que no son de grado mínimo.

Dados dos nodos u y v de un grafo G, diremos que el nodo v está señalado por el nodo u si se verifica que : $Ady_G(u) \cup \{u\} \leq Ady_G(v) \cup \{v\}$. De esta manera si el nodo v está señalado por u en G, entonces también está señalado por u en G(y), de esta manera si un nodo v está señalado por u en el proceso de eliminación, el nodo u puede ser eliminado antes que v en el algoritmo de ordenamiento de grado mínimo.

Luego si v se convierte en señalado por u en alguna etapa durante la eliminación, no es necesario actualizar el grado de v hasta que el nodo u sea eliminado, luego usando la propiedad de señalamiento la actualización del grado se salta para aquellos nodos que no participan en la próxima ronda de selección del grado mínimo.

Volviendo al ejemplo anterior:



notamos que los nodos en {22, 23, 24} están todos señalados por cada uno de los nodos que permanecen, que son : {15, 16, 17, 18, 19, 20, 21, 25}, luego no es necesario recalcular los grados de los nodos 22, 23 y 24 hasta que todos los otros sean eliminados. Esta técnica es muy potente para la realización del algoritmo de ordenamiento.

5.4.5 Eliminación múltiple.

La actualización del grado incompleta también puede ser considerada como una técnica de actualización del grado retardada. La importancia de la eliminación múltiple extiende la idea al retardo de la actualización del grado, o sea en lugar de realizar un paso de actualización después de cada selección de nodo de grado mínimo y la correspondiente transformación del grafo de eliminación, la técnica de eliminación múltiple pospone el paso de actualización a una etapa posterior. Esta técnica se puede suponer como un concepto opuesto a la eliminación de masa estudiada anteriormente. La base para la eliminación múltiple es la siguiente :

Esta usa la observación de que en la eliminación del nodo y del grafo G, la estructura asociada con los nodos que no están en $Ady_G(y)$ permanece intacta. La idea es suspender la actualización para nodos en $Ady_G(y)$ y seleccionar un nodo con el mismo grado que y en el subgrafo G que queda o sea en $G-(Ady_G(y)\cup\{y\}$.Este proceso se repite hasta que no existan nodos en el subgrafo que queda con grado $=\delta$ G(y), entonces se realiza un paso de actualización.

En realidad lo que se hace antes de cada paso de actualización es seleccionar un conjunto independiente de nodos con grado mínimo.

Esta modificación no siempre produce un ordenamiento de grado mínimo genuino, pero raras veces produce un ordenamiento inferior .

Para entender la eliminación múltiple consideremos la malla 5 x 5 con nueve puntos de conectividad, donde se han eliminado ocho nodos que están marcados por •.

•	•	11	•	•
20	21	22	16	15
10	14	23	13	9
•	18	24	25	•
•	17	12	19	•

En esta etapa el orden mínimo es cinco. El conjunto {9, 10, 11, 12} forma un conjunto independiente con grado cinco. Eliminándolos entonces todos antes de una actualización, ayudará a reducir el tiempo de la misma.

Necesitamos actualizar el grado de los nodos en {13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25} solo una vez antes de la eliminación de {9, 10, 11, 12}.

5.4.6 Grado externo.

El grado externo de un nodo se define como el número de nodos adyacentes a él que no son indistinguibles.

Liu sugiere el uso del grado externo en vez del verdadero al aplicar el algoritmo del grado mínimo. La eliminación de un nodo de grado mínimo implica la formación del ciclo más pequeño posible debido a al eliminación.

•	•	•	•	•
20	21	22	16	15
•	•	23	•	•
•	18	24	25	•
•	17	•	19	•

En la figura tenemos que los nodos 15 y 16 son indistinguibles, y el grado externo del nodo 15 es siete(siendo su verdadero grado ocho).Los resultados experimen-

tales muestran que usando el grado externo se llega a una reducción en el número de no ceros en la matriz del factor.

5.4.7 Grado mínimo con preordenamiento.

Si al aplicar el algoritmo de grado mínimo se tiene mucho cuidado al seleccionar los nodos que tengan grado mínimo, puede haber una ganancia muy significativa en términos de almacenamiento y minimización de las operaciones aritméticas.

Una manera de lograr este objetivo es fijar el ordenamiento inicial reorganizando la estructura de adyacencia antes de pasarla a la subrutina de ordenamiento de grado mínimo. Un ordenamiento inicial que da muy buenos resultados, es aplicar al principio el ordenamiento inverso de CM. La estrategia general es :

$$A \xrightarrow{RCM} \tilde{A} = P_r A P_r^t \xrightarrow{GM} P\tilde{A}P^t$$

Donde P_r es el ordenamiento inverso de Cuthill-McKee (RCM) en la matriz dada A y P es el ordenamiento de grado mínimo (GM) en la matriz permutada $\tilde{A} = P_r A P_r^t$. Este reordenamiento inicial tiene efectos muy destacables en problemas de mallas ya que el orden mínimo resultante es el mismo, independientemente de como hayamos permutado la matriz A. El algoritmo inverso de Cuthill-McKee elimina la aleatoriedad en los arreglos de filas y columnas antes de presentar la matriz al algoritmo del grado mínimo.

5.5 Factorización Simbólica.

Como su nombre implica, la factorización simbólica es el proceso de simular la factorización numérica de una matriz dada A, para así obtener la estructura del nonulo de su factor L. El problema puede ser convenientemente estudiado usando la teroría de aproximación de grafos [24].

Sea $G^{\alpha} = (X^{\alpha}, E)$ un grafo ordenado, donde $|X^{\alpha}| = n$ y para conveniencia sea $\alpha(i) = x_i$. La factorización simbólica puede ser vista como determinaciones de los conjuntos Acces de la forma Acces $(x_i, \{ x_1, \ldots, x_{i-1} \})$ para i, = 1,..., n.

Definamos $S_i = \{x_1, \ldots, x_i\}$ como el conjunto de nodos eliminados, entonces se tiene el siguiente lema:

$$\text{Acces}(x_i, S_{i-1}) = \text{Ady}(x_i) \cup \left(\bigcup_k \{\text{Acces}(x_k, S_{k-1}) \ / \ x_i \in \text{Acces}(x_k, S_{k-1})\}\right) - S_i$$

O Demostración.

Sea $j > i$. Entonces se tiene que:

$x_j \in \text{Acces}(x_i, S_{i-1})$ si $\{x_i, x_j\} \in E(F)$, si $\{x_i, x_j\} \in E(F) =$ y $\{x_i, x_j\} \in E(F)$ para algún $k < \min\{ i,j \}$, si $x_j \in \text{Adj}(x_i)$ o $x_i, x_j \in \text{Alcan}(x_k, S_{k-1})$ para algún k.

Por lo tanto podemos escribir un algoritmo para encontar los conjuntos alcanzables y de esta manera hallar la estructura del factor L. Dicho algoritmo lo describimos como sigue:

Primer paso: (Inicialización) Para k = 1,..., n.

$\text{Acces}(x_k, S_{k-1}) \leftarrow \text{Adj}(x_k) - S_{k-1}$

Segundo paso: (Factorización simbólica)

Para k = 1,..., n.

si $x_i \in \text{Acces}(x_k, S_{k-1})$ entonces:

$\text{Alcan}(x_i, S_{i-1}) \leftarrow \text{Acces}(x_i, S_{i-1}) \cup \text{Acces}(x_k, S_{k-1}) - S_i$

La ilustración de este algoritmo está desrita en la siguiente figura 5.5.0 que simula esencialmente la factorización entera.

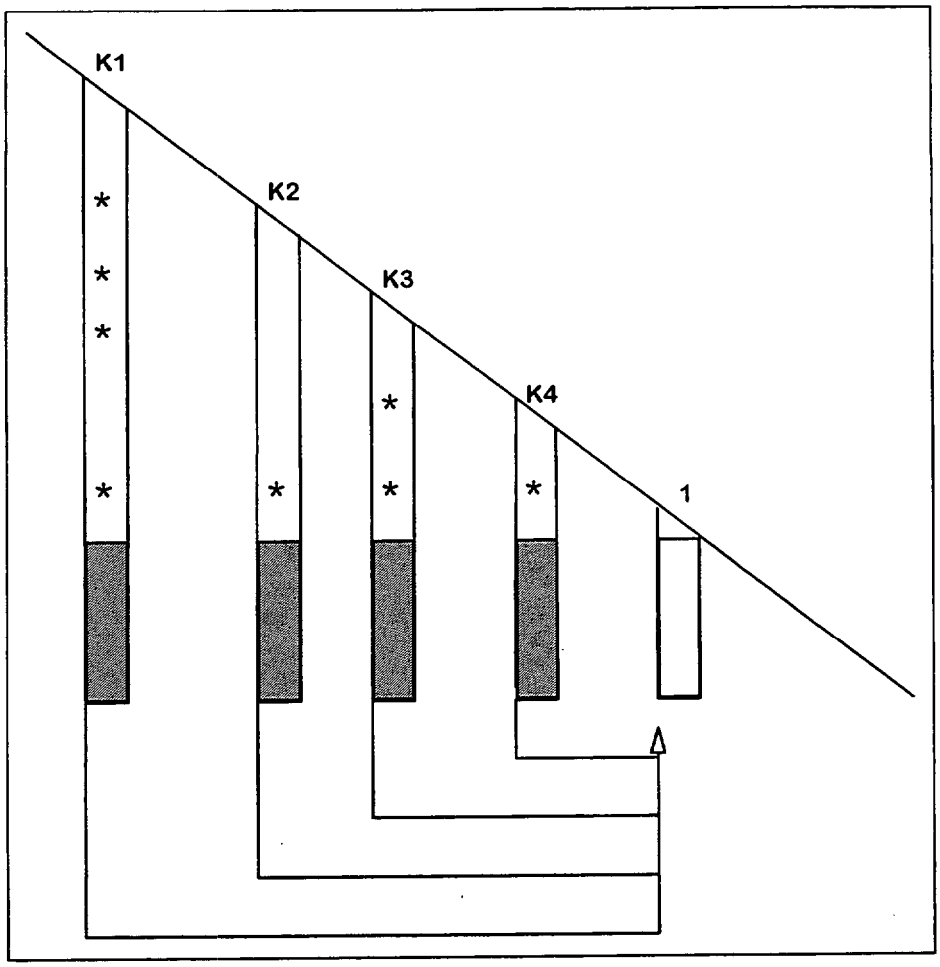


Fig 5.5.0

A continuación optimizaremos este algoritmo. Consideremos la etapa dónde el conjunto $S_{i-1} = \{x_1, \ldots\ldots, x_{i-1}\}$ ha sido eliminado. Para analizar este punto, asumimos que x_i tiene dos componentes conexas en $G(S_{i-1})$ adyacente a él. Sean las componente conexas C_1 y C_2 segun indicamos en la figura 5.5.1:

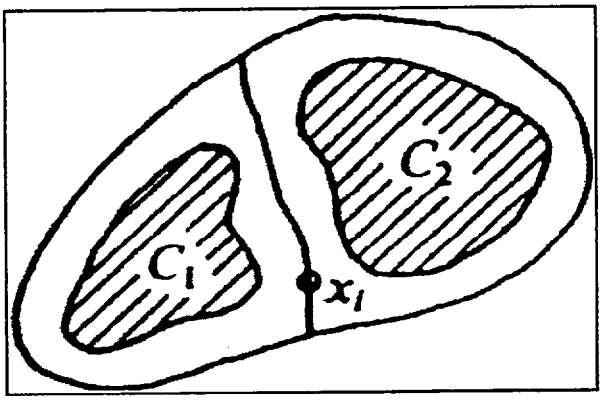


Fig: 5.5.1

En este caso, tenemos que:

$$\text{Acces}(x_i, S_{i-1}) = \text{Adj}(x_i) \cup \text{Adj}(C_1) \cup \text{Adj}(C_2) - S_i$$

Si elegimos dos representantes x_{r_1} y x_{r_2}, de C_1 y C_2 respectivamente entonces $\text{Adj}(C_1) = \text{Acces}(x_{r_1}, S_{r_1-1})$ y $\text{Adj}(C_2) = \text{Acces}(x_{r_2}, S_{r_2-1})$.

De esta manera, el conjunto de Acces puede ser escrito como:

$$\text{Acces}(x_i, S_{i-1}) = \text{Adj}(x_i) \cup \text{Acces}(x_{r_1}, S_{r_1-1}) \cup \text{Acces}(x_{r_2}, S_{r_2-1}) - S_i$$

Así, en vez de tener muchos conjuntos alcanzables podemos seleccionar sus representantes. Esta selección se puede llevar a cabo por la siguiente observación:

Para k = 1, ..., n, definimos:

$$m_k = \min\{j \mid x_j \in \text{Acces}(x_k, S_{k-1}) \cup \{x_k\}\}$$

En términos de la matriz, es el primer nocero en el vector columna L_{*k} excluyendo el componente diagonal.

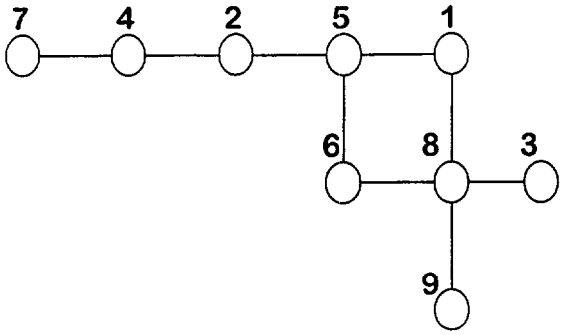
Por lo tanto: $\text{Acces}(x_k, S_{k-1}) \subset \text{Acces}(x_{m_k}, S_{m_k-1}) \cup \{x_{m_k}\}$ y para cualquier $x_i \in \text{Acces}(x_k, S_{k-1})$, entonces $k < m_k \leq i$. Si $i = m_k$, no hay nada que probar. De otra manera, se tiene que: $x_i \in \text{Acces}(x_{m_k}, S_{m_k-1})$.

Para $x_i \in \text{Acces}(x_k, S_{k-1})$ y $i > m_k$, es redundante considerar $\text{Acces}(x_k, S_{k-1})$, pues todos los nudos alcanzables vía x_k pueden ser encontrados en $\text{Acces}(x_{m_k}, S_{m_k-1})$. Así, es suficiente unir los conjuntos alcanzables de algunos nudos representativos.

Según esto se tiene:

$$\text{Acces}(x_i, S_{i-1}) = \text{Adj}(x_i) \cup \left(\bigcup_k \{\text{Acces}(x_k, S_{k-1}) \mid m_k = i\} \right) - S_i$$

Como ejemplo tomamos el grafo siguiente:



k	m_k	$Acces(x_k, S_{k-1})$
1	5	x_5, x_8
2	4	x_4, x_5
3	8	x_8
4	5	x_5, x_7

Consideremos la determinación de Acces (x_5, S_4) en el grafo anterior. Vemos que los conjuntos :

$$\text{Acces } (x_1, S_0)$$

$$\text{Acces } (x_2, S_1)$$

y

$$\text{Acces } (x_4, S_3)$$

tienen que ser unidos con $Adj(x_5)$.

Según lo anterior es suficiente considerar Acces (x_1, S_0) y Acces (x_4, S_3).

Para $m_2 = 4$, tenemos que Acces $(x_2, S_1) = \{x_4, x_5\} \subset$ Acces $(x_4, S_3) \cup \{x_4\}$.

La factorización simbólica algorítmica puede ser ahora descrita por el siguiente algoritmo:

Paso 1: (Inicialización)

Para $k = 1, \ldots, n$,

$$\text{Acces } (x_k, S_{k-1}) = \text{Adj } (x_k) - S_k$$

Paso 2: (Factorización simbólica)

Para $k = 1, \ldots, n$,

si $\text{Acces}(x_k, S_{k-1}) \neq \varnothing$ entonces

$$m = min\{j \mid x_j \in \text{Acces}\,(x_k, S_{k-1})\}$$

$$\text{Acces}\,(x_m, S_{m-1}) \leftarrow \text{Acces}\,(x_m, S_{m-1}) \cup$$

$$(\text{Access}\,(x_k, S_{k-1}) - \{x_m\}.$$

A continuación describimos la subrutina que ejecuta la factorización simbólica, esta fué diseñada por Eisenstat, et al. (1981), y puede encontrarse en Yale Sparse Matrix Package. Esencialmente implementa el algoritmo perfeccionado como se describió anteriormente, con una reorganización del orden donde los conjuntos alcanzables están juntos.

Paso 1: (Inicialización) Para $i=1,\ldots,n$, $R_i = \varnothing$

Paso 2: (Factorización simbólica)

Para $i=1,\ldots,n$,

$\text{Acces}\,(x_i, S_{i-1}) = \text{Adj}\,(x_i) - S_i$

Para $k \in R_i$ hacer

$\text{Acces}\,(x_i, S_{i-1}) \leftarrow \text{Acces}\,(x_i, S_{i-1}) \cup$

$\text{Acces}\,(x_k, S_{k-1}) - S_i$

$m = min\{j \mid x_j \in \text{Acces}\,(x_i, S_{i-1})\}$

$R_m \leftarrow R_m \cup \{x_i\}$

En este algoritmo, el conjunto R_i, es utilizado para acumular representantes en los cuáles sus conjuntos alcanzables afecta x_i.

Si hay sólo un $m_k = i$, y $\text{Adj}\,(x_i) - S_i \subset \text{Acces}\,(x_k, S_{k-1})$ entonces:

$$\text{Acces}\,(x_i, S_{i-1}) = \text{Acces}\,(x_k, S_{k-1}) - \{x_i\}.$$

La subrutina SMBFCT tiene como entrada el grafo de la matriz guardada en el par (ADYACY, XADJ), junto con el vector de permutación PERM y su inversa INVP.

Tambien la factorización simbólica se pude hacer mediante la observación de que tanto la matriz A como la matriz L tiene el mismo perfil, por lo tanto si consideramos una matriz simétrica definida positiva $A=(a_{ij})$ y su factorización de Cholesky $A=LL^t$, en donde $L=(l_{ij})$ es triangular inferior y con entradas diagonales positivas ,podemos estudiar la estructura simbólica de L conocida la estructura de A.

Si la matriz A es densa, será necesario calcular las $(n^2+n)/2$ entradas de la matriz L. Pero si la matriz A es grande y sparse, muchas entradas de la matriz triangular serán nulas, y entonces sería interesante conocer la estructura simbólica de L antes de comenzar la factorización numérica al objeto de reducir el coste computacional de dicha factorización.

Una solución parcial sencilla de este problema sería el observar que las matrices A y L tienen el mismo perfil, es decir, que la primera entrada no nula de cada fila ocupa el mismo lugar en L y A, pero en la matriz L habrá probablemente entradas posteriores a la primera que , aún siendo nulas en A, no lo son en ella.

Comenzamos hallando LL^t e identificando coeficientes:

$$l_{11}=\sqrt{a_{11}}$$
$$l_{21}=a_{21}/l_{11}$$
$$l_{31}=a_{31}/l_{11}$$
.....

En general: $l_{i1}=a_{i1}/l_{11}$, lo que nos indica que las entradas no nulas de la primera columna de la matriz L están en las mismas posiciones que las de la primera columna de la matriz A, ya que $l_{11}\neq 0$. Podemos decir que las primeras columnas de A y L tienen la misma estructura simbólica o lógica.

Si seguimos adelante:

$$l_{32}=(a_{32}-l_{31}.\ l_{21})\ /\ l_{22}$$

$$l_{42}=(a_{42}-l_{41}.\ l_{21})\ /\ l_{22}$$

$$l_{43}=(a_{43}-l_{41}.\ l_{31}-l_{42}.\ l_{32})\ /\ l_{33}$$

.....

En general:

$$l_{ij}=(a_{ij}-\sum_{k=1}^{j-1} l_{ik}\ .\ l_{jk})\ /\ l_{jj}$$

Por lo tanto la entrada l_{ij} de L será igual a cero si lo es la entrada aij de A y, además, es cero uno de los dos factores $l_{ik}.\ l_{jk}$ en cada uno de los k sumandos de $\sum_{k=1}^{j-1} l_{ik}\ .\ l_{jk}$.

Vamos a verlo en un caso práctico. Sea la matriz A de la figura 5.5.2, simétrica y definida positiva:

$$A=\begin{bmatrix} 1 & & & & & & & & & \\ x & 2 & & & & & SIM. & & & \\ & x & 3 & & & & & & & \\ & & & 4 & & & & & & \\ x & x & & & 5 & & & & & \\ & & x & & x & 6 & & & & \\ & & & x & & x & 7 & & & \\ x & & & & & & & 8 & & \\ & & & & x & x & x & x & 9 & \\ & & & & & & x & & x & 10 \end{bmatrix}.$$

Fig 5.5.2

Los vectores ADYACI y XADJ para la matriz A son :

ADYACI: 2,5,8,1,3,5,2,6,7,1,2,6,9,3,5,7,9,4,6,9,10,1,9,5,6,7,8,10,7,9

	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑		↑	↑	
XADJ:	1	4	7	9	10	14	18	22	24			29	31	
nodo:	1	2	3	4	5	6	7	8	9			10		

La entrada $l_{32} \neq 0$, por ser $a_{32} \neq 0$, ya que en ADYACI vemos que en la lista de adyacencia del nodo 2 figura el nodo 3 (en la posición número 5 de ADYACI).

La entrada $l_{42}=0$, ya que $a_{42}=0$ y además $l_{21}=0$ o $l_{41}=0$.

La entrada $l_{43}=0$, ya que $a_{43}=0$ y además $l_{31}=0$ y $l_{43}=0$.

La entrada $l_{53} \neq 0$, ya que $a_{53} \neq 0$.

La entrada $l_{54}=0$, ya que $a_{54}=0$ y además $l_{41}=l_{42}=l_{43}=0$.

Y, procediendo de este modo, obtenemos la estructura simbólica de L, de la matriz A de la figuira 5.5.2:

$$L = \begin{bmatrix} x & & & & & & & & & & \\ x & x & & & & & & & & & \\ & x & x & & & & & & & & \\ & & & x & & & & & & & \\ x & x & x & & x & & & & & & \\ & & x & & x & x & & & & & \\ & & & x & & x & x & & & & \\ x & x & x & & x & x & x & x & & & \\ & & & & x & x & x & x & x & & \\ & & & & & & x & x & x & x \end{bmatrix}$$

En efecto, la lista de adyacencia en L del nodo 1, $Adj_L(1)$ está formada por los nodos 2, 5 y 8:

$$Adj_L(1) = Adj_A(1) = \{2,5,8\}.$$

Por tanto, la lista de adyacencia en L del nodo 2, $Ady_L(2)$, será igual a $Ady_A(2)$ unida con la lista de adyacencia en L del nodo 1, salvo los nodos 1 y 2:

$$Ady_L(2) = Ady_A(2) \cup Ady_A(1) - \{1,2\} = \{1,3,5\} \cup \{2,5,8\} - \{1,2\} = \{3,5,8\}.$$

La lista de adyacencia en L del nodo 3, será igual a $Ady_A(3)$ unida con $Ady_L(2)$. No tomamos la $Ady_L(1)$ puesto que $l_{31}=0$. Por tanto:

$$Ady_L(3) = Ady_A(3) \cup Ady_L(2) - \{1,2,3\} = \{6\} \cup \{3,5,8\} - \{1,2,3\} = \{5,6,8\}.$$

La lista de adyacencia en L del nodo 4:

$$Ady_L(4) = Ady_A(4) \cup \varnothing = \{7\}.$$

La lista de adyacencia en L del nodo 5:

$$Ady_L(5) = Ady_A(5) \cup Ady_L(1) \cup Ady_L(2) \cup Ady_L(3) - \{1,2,3,4,5\} = \{6,9\} \cup \{2,5,8\} \cup \{3,5,8\} \cup \{5,6,8\} - \{1,2,3,4,5\} = \{6,8,9\}.$$

La lista de adyacencia en L del nodo 6:

$$Ady_L(6) = Ady_A(6) \cup Ady_L(3) \cup Ady_L(5) - \{1,2,3,4,5,6\} = \{3,5,7,9\} \cup \{5,8\} \cup \{6,8\} - \{1,2,3,4,5,6\} = \{7,8,9\}.$$

La lista de adyacencia en L del nodo 7:

$$Ady_L(7) = Ady_A(7) \cup Ady(4) \cup Ady(6) - \{1,2,3,4,5,6,7\} = \{4,6,9,10\} \cup \{7\} \cup \{3,5,7,8,9\} - \{1,2,3,4,5,6,7\} = \{8,9,10\}.$$

En general, la lista de adyacencia del nodo i en L será:

$$Ady_L(i) = Ady_A(i) \cup \left[\bigcup_{j<i} Ady_L(j) / l_{ij} \neq 0 \right] - \{k / k \leq i\}$$

Los vectores ADYACI y XADJ para L^t serán:

ADYACI:	2,5,8	3,5,8	5,6,8	7	6,8,9	7,8,9	8,9,10	9,10	10	
	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
XADJ:	1	4	7	10	11	14	17	20	22	23
nodo:	1	2	3	4	5	6	7	8	9	

Hemos obtenido ADYACI para el grafo dirigido correspondiente a la matriz triangular superior L^t.

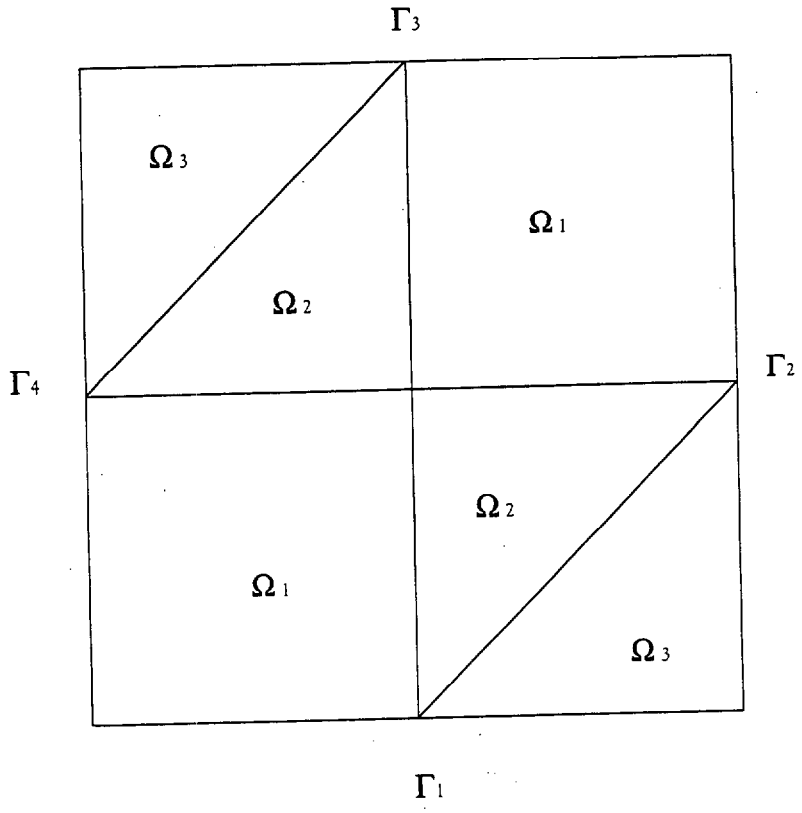
Aplicaciones

En éste capítulo aplicamos los distintos algoritmos de reordenamiento que hemos visto en los capítulos anteriores. Lo que se pretende no es solo obtener la solución del sistema, sino comprobar la efectividad de las distintas técnicas de reordenamiento y la disminución del efecto fill-in.

A los distintos problemas que luego mencionaremos, se les ha aplicado en primer lugar el algoritmo inverso de CM, luego el algoritmo de grado mínimo y por último el algoritmo de grado mínimo de eliminación múltiple.

En la aplicación de los elementos finitos para obtener el sistema de ecuaciones, todos los problemas son referidos a un dominio de geometría simple, de cuadrado de lado la unidad.

Sea el dominio $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$ de frontera $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$



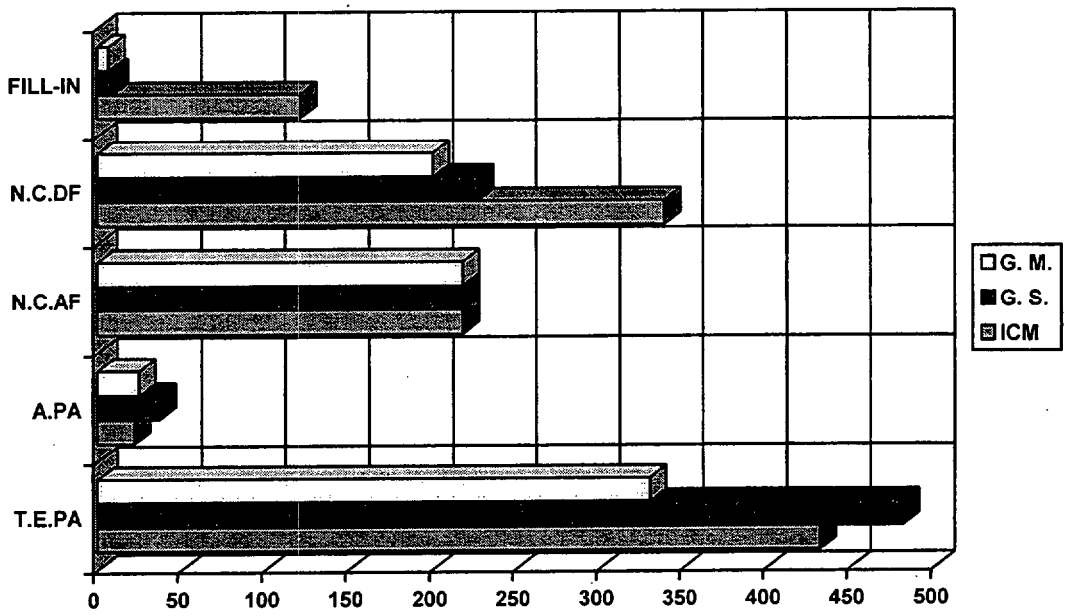
El código Neptuno, además de genenarnos el sistema de ecuaciones, nos permite obtener el fichero de salida que luego será utilizado para representar el mallado correspondiente y la solución del problema.

A título ilustrativo, hemos desarrollado algunos ejemplos de sistemas sparse, con distintas dimensiones, que hemos resuelto por las técnicas de renumeración usadas en esta Tesis, que son: el algoritmo inverso de CM, grado mínimo y grado mínimo múltiple, cuyos resultados son los que se reflejan en las siguientes tablas:

TABLA EJEMPLO NÚMERO 1

Número de ecuaciones:		40
Número de entradas de A:		1600
Número de no-ceros de A:		476
Dispersidad de A:		71%
Tamaño de la envoltura de A:		524
Ancho de banda de A:		37
Dimensión del vector de adyacencia:		436
ICM	Tamaño de la envoltura de PA:	432
	Ancho de banda de PA:	22
	Número de ceros antes de factorizar:	218
	Número de ceros después de factorizar:	339
	Fill-in:	121
GRAMIN SIMPLE Y GRAMIN MULTIPLE*	Tamaño de la envoltura de PA:	481
	Ancho de banda de PA:	37
	Número de ceros antes de factorizar:	218
	Número de ceros después de factorizar:	227
* Reducción del tiempo de ejecución.	Fill-in:	9

EJEMPLO NÚMERO 1

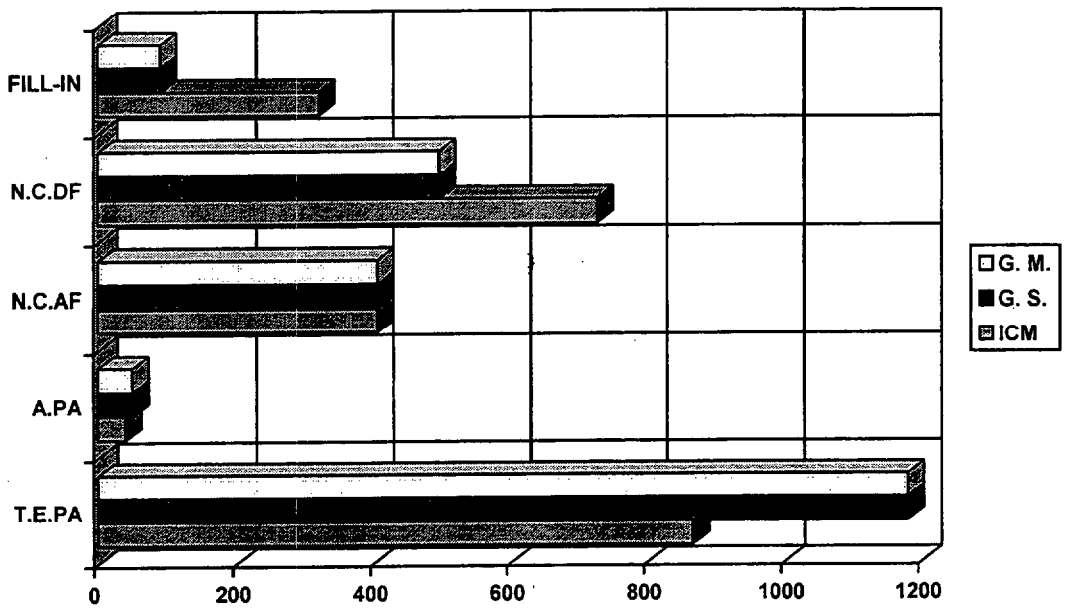


En el eje de abscisa está el número de elementos no nulos.

FILL-IN: ELEMENTOS DE RELLENO DESPUÉS DE FACTORIZAR.
N.C.AF: NUMEROS DE CEROS ANTES DE FACTORIZAR .
N.C.DF: NÚMEROS DE CEROS DESPUÉS DE FACTORIZAR.
A.PA: ANCHO DE BANDA DE P.A.
TE.PA: TAMAÑO DE LA ENVOLTURA DE P.A.

TABLA EJEMPLO NÚMERO 2		
Número de ecuaciones:		60
Número de entradas de A:		3600
Número de no-ceros de A:		872
Dispersidad de A:		76%
Tamaño de la envoltura de A:		1290
Ancho de banda de A:		57
Dimensión del vector de adyacencia:		812
ICM	Tamaño de la envoltura de PA:	866
	Ancho de banda de PA:	40
	Número de ceros antes de facto1rizar:	406
	Número de ceros después de factorizar:	726
	Fill-in:	320
GRAMIN SIMPLE Y GRAMIN MULTIPLE*	Tamaño de la envoltura de PA:	1180
	Ancho de banda de PA:	50
	Número de ceros antes de factorizar:	406
	Número de ceros después de factorizar:	496
* Reducción del tiempo de ejecución.	Fill-in:	90

EJEMPLO NÚMERO 2

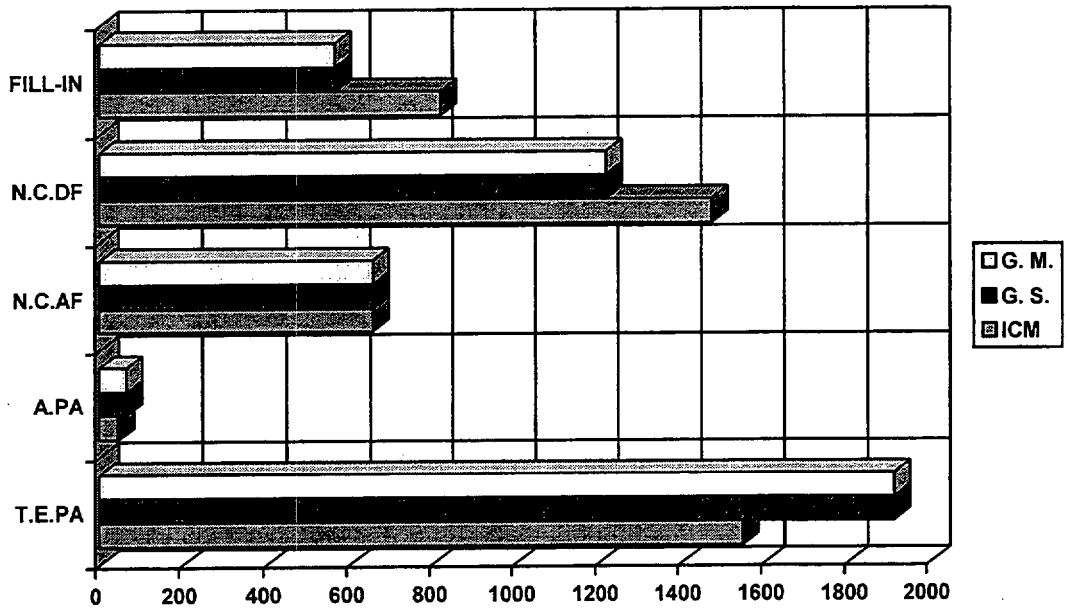


En el eje de abscisa está el número de elementos no nulos.

FILL-IN: ELEMENTOS DE RELLENO DESPUÉS DE FACTORIZAR.
N.C.AF: NUMEROS DE CEROS ANTES DE FACTORIZAR .
N.C.DF: NÚMEROS DE CEROS DESPUÉS DE FACTORIZAR.
A.PA: ANCHO DE BANDA DE P.A.
TE.PA: TAMAÑO DE LA ENVOLTURA DE P.A.

TABLA EJEMPLO NÚMERO 3		
Número de ecuaciones:		70
Número de entradas de A:		4900
Número de no-ceros de A:		1378
Dispersidad de A:		72%
Tamaño de la envoltura de A:		1940
Ancho de banda de A:		65
Dimensión del vector de adyacencia:		1308
ICM	Tamaño de la envoltura de PA:	1547
	Ancho de banda de PA:	47
	Número de ceros antes de factorizar:	654
	Número de ceros después de factorizar:	1473
	Fill-in:	819
GRAMIN SIMPLE Y GRAMIN MULTIPLE*	Tamaño de la envoltura de PA:	1913
	Ancho de banda de PA:	66
	Número de ceros antes de factorizar:	654
	Número de ceros después de factorizar:	1218
* Reducción del tiempo de ejecución.	Fill-in:	564

EJEMPLO NÚMERO 3



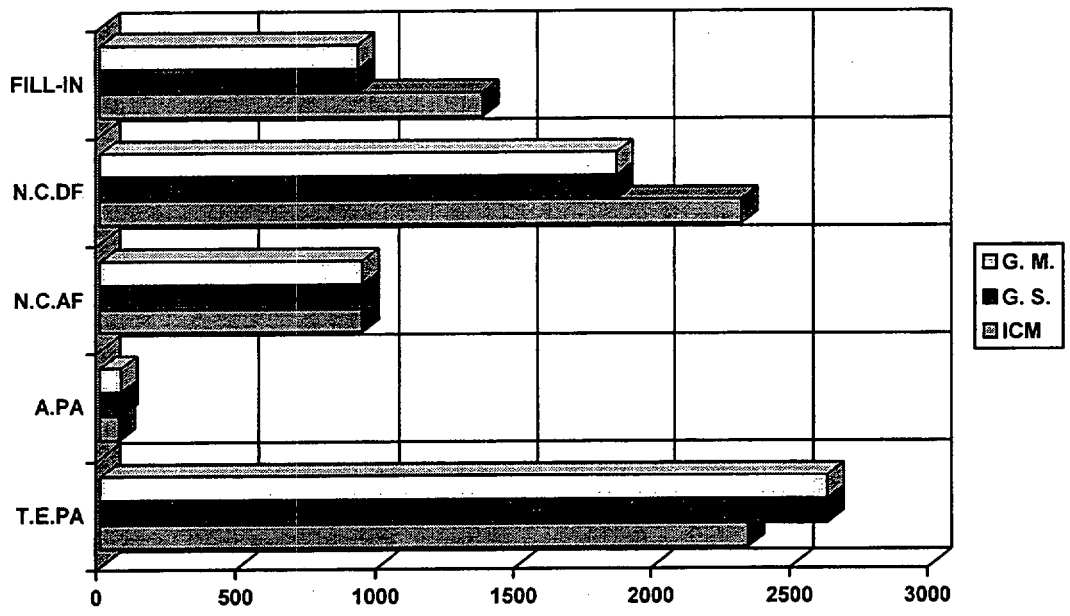
En el eje de abscisa está el número de elementos no nulos.

FILL-IN: ELEMENTOS DE RELLENO DESPUÉS DE FACTORIZAR.
N.C.AF: NUMEROS DE CEROS ANTES DE FACTORIZAR .
N.C.DF: NÚMEROS DE CEROS DESPUÉS DE FACTORIZAR.
A.PA: ANCHO DE BANDA DE P.A.
TE.PA: TAMAÑO DE LA ENVOLTURA DE P.A.

TABLA EJEMPLO NÚMERO 4

Número de ecuaciones:		80
Número de entradas de A:		6400
Número de no-ceros de A:		1960
Dispersidad de A:		70%
Tamaño de la envoltura de A:		2564
Ancho de banda de A:		76
Dimensión del vector de adyacencia:		1880
ICM	Tamaño de la envoltura de PA:	2337
	Ancho de banda de PA:	68
	Número de ceros antes de factorizar:	940
	Número de ceros después de factorizar:	2314
	Fill-in:	1374
GRAMIN SIMPLE Y GRAMIN MULTIPLE*	Tamaño de la envoltura de PA:	2623
	Ancho de banda de PA:	76
	Número de ceros antes de factorizar:	940
	Número de ceros después de factorizar:	1863
* Reducción del tiempo de ejecución.	Fill-in:	923

EJEMPLO NÚMERO 4



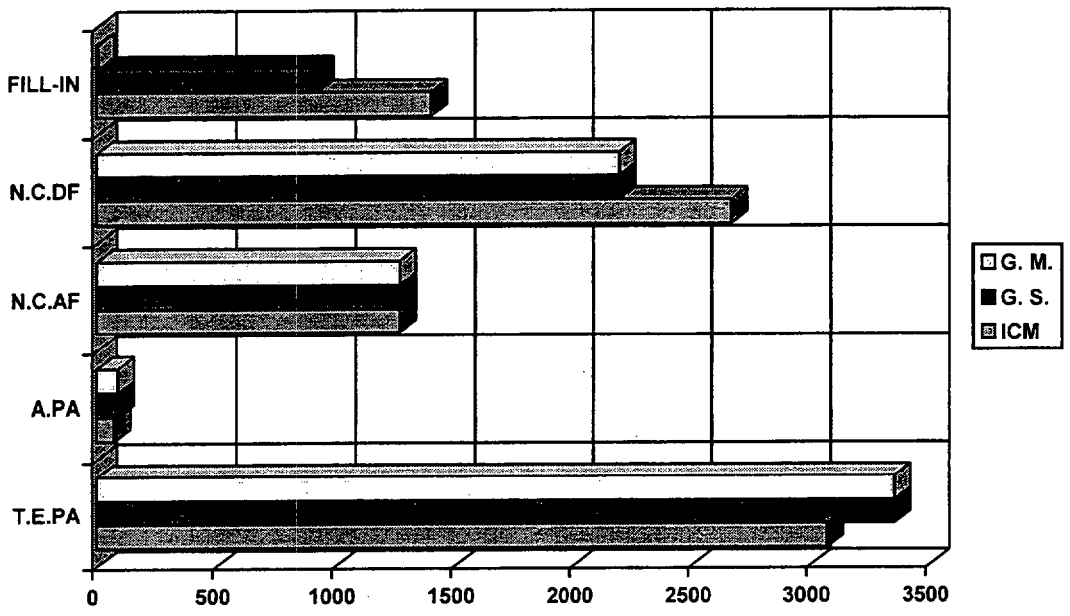
En el eje de abscisa está el número de elementos no nulos.

FILL-IN: ELEMENTOS DE RELLENO DESPUÉS DE FACTORIZAR.
N.C.AF: NUMEROS DE CEROS ANTES DE FACTORIZAR .
N.C.DF: NÚMEROS DE CEROS DESPUÉS DE FACTORIZAR.
A.PA: ANCHO DE BANDA DE P.A.
TE.PA: TAMAÑO DE LA ENVOLTURA DE P.A.

TABLA EJEMPLO NÚMERO 5

Número de ecuaciones:		90
Número de entradas de A:		8100
Número de no-ceros de A:		2630
Dispersidad de A:		68%
Tamaño de la envoltura de A:		3220
Ancho de banda de A:		87
Dimensión del vector de adyacencia:		2540
ICM	Tamaño de la envoltura de PA:	3069
	Ancho de banda de PA:	73
	Número de ceros antes de factorizar:	1270
	Número de ceros después de factorizar:	2670
	Fill-in:	1400
GRAMIN SIMPLE Y GRAMIN MULTIPLE*	Tamaño de la envoltura de PA:	3352
	Ancho de banda de PA:	87
	Número de ceros antes de factorizar:	1270
	Número de ceros después de factorizar:	2193
* Reducción del tiempo de ejecución.	Fill-in:	923

EJEMPLO NÚMERO 5



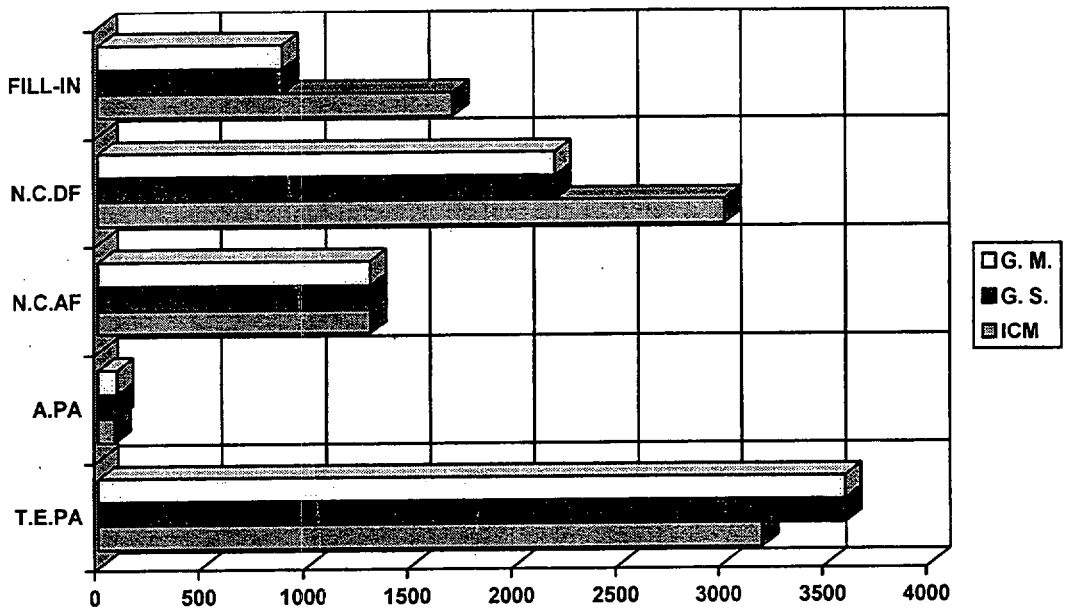
En el eje de abscisa está el número de elementos no nulos.

FILL-IN: ELEMENTOS DE RELLENO DESPUÉS DE FACTORIZAR.
N.C.AF: NUMEROS DE CEROS ANTES DE FACTORIZAR .
N.C.DF: NÚMEROS DE CEROS DESPUÉS DE FACTORIZAR.
A.PA: ANCHO DE BANDA DE P.A.
TE.PA: TAMAÑO DE LA ENVOLTURA DE P.A.'

TABLA EJEMPLO NÚMERO 6

Número de ecuaciones:		95
Número de entradas de A:		9025
Número de no-ceros de A:		2715
Dispersidad de A:		70%
Tamaño de la envoltura de A:		3524
Ancho de banda de A:		89
Dimensión del vector de adyacencia:		2620
ICM	Tamaño de la envoltura de PA:	3193
	Ancho de banda de PA:	80
	Número de ceros antes de factorizar:	1310
	Número de ceros después de factorizar:	3015
	Fill-in:	1705
GRAMIN SIMPLE Y GRAMIN MULTIPLE*	Tamaño de la envoltura de PA:	3592
	Ancho de banda de PA:	93
	Número de ceros antes de factorizar:	1310
	Número de ceros después de factorizar:	2197
* Reducción del tiempo de ejecución.	Fill-in:	887

EJEMPLO NÚMERO 6



En el eje de abscisa está el número de elementos no nulos.

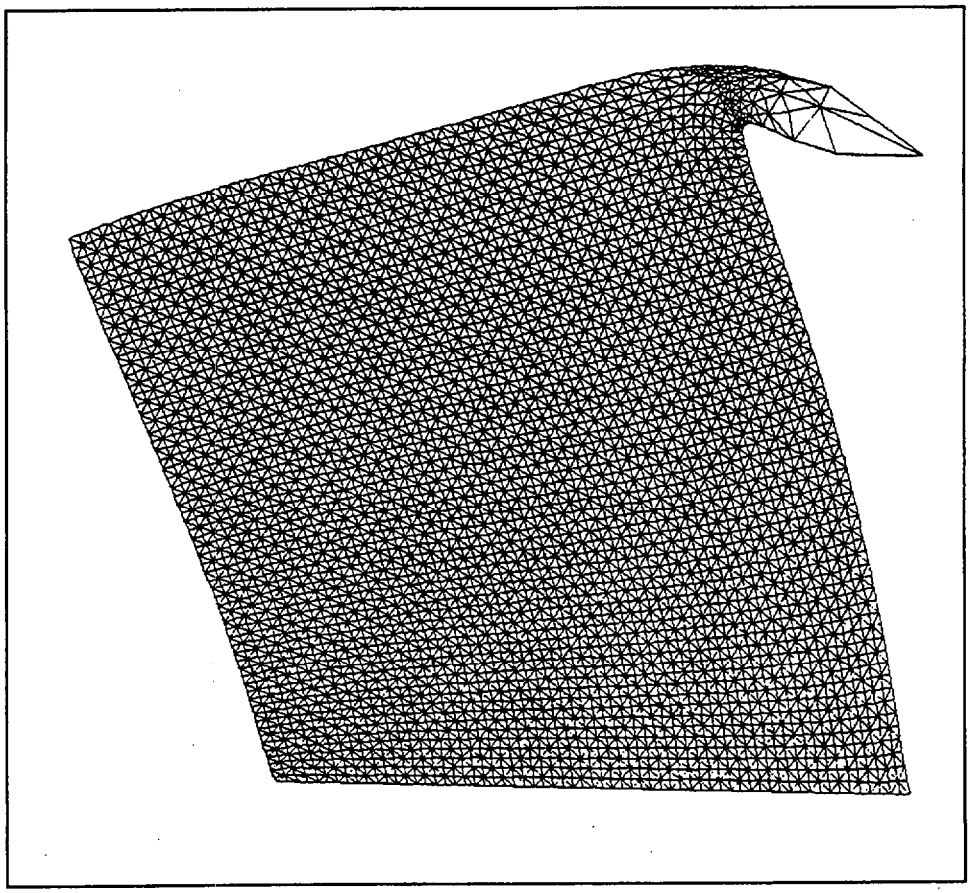
FILL-IN: ELEMENTOS DE RELLENO DESPUÉS DE FACTORIZAR.
N.C.AF: NUMEROS DE CEROS ANTES DE FACTORIZAR .
N.C.DF: NÚMEROS DE CEROS DESPUÉS DE FACTORIZAR.
A.PA: ANCHO DE BANDA DE P.A.
TE.PA: TAMAÑO DE LA ENVOLTURA DE P.A.

En todos ellos, y en muchos otros que hemos ejecutado, se ha observado la bondad del método del grado mínimo en la reducción del efecto fill-in, que redunda en disminuciones de almacenamiento y tiempo de ejecución.

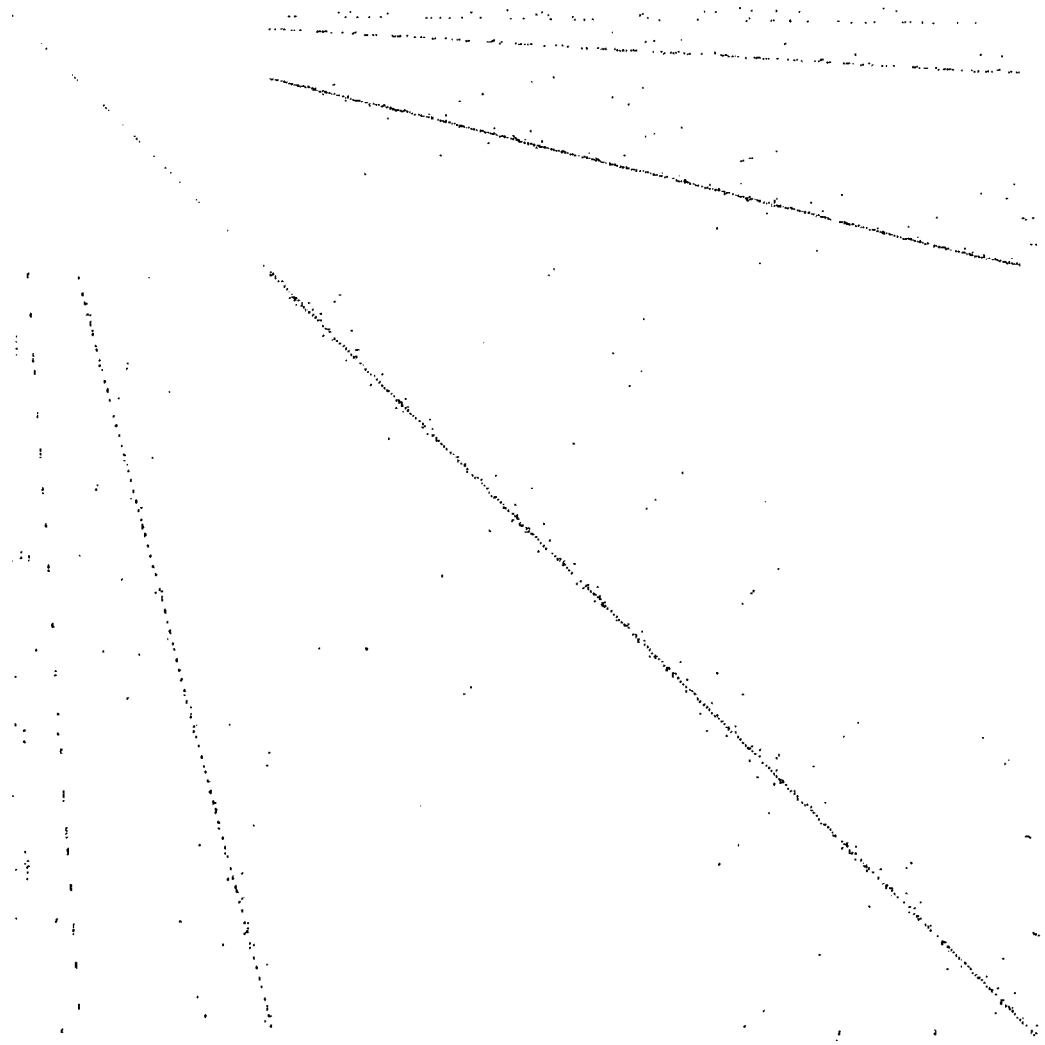
Además en el grado mínimo múltiple se obtiene una reducción del tiempo de ejecución .Por otra parte, en todos los casos hemos aplicado refinamiento iterativo, el cual no ha sido necesario en algunos ejemplos a los que se les ha renumerado prevismente por grado mínimo.

En el caso del problema de deformacioón de una barra elástica delgada Ω, que está sometida a una carga constante en Γ_2 ($L_2 = -1$ kgs/m) y por otra que varia linealmente en Γ_3,($L_3 = -x/25$ kgs/m). Imponemos la condición de inmovilidad en Γ_1 y en el punto (1, 1). Las fuerzas de volumen se consideran nulas.

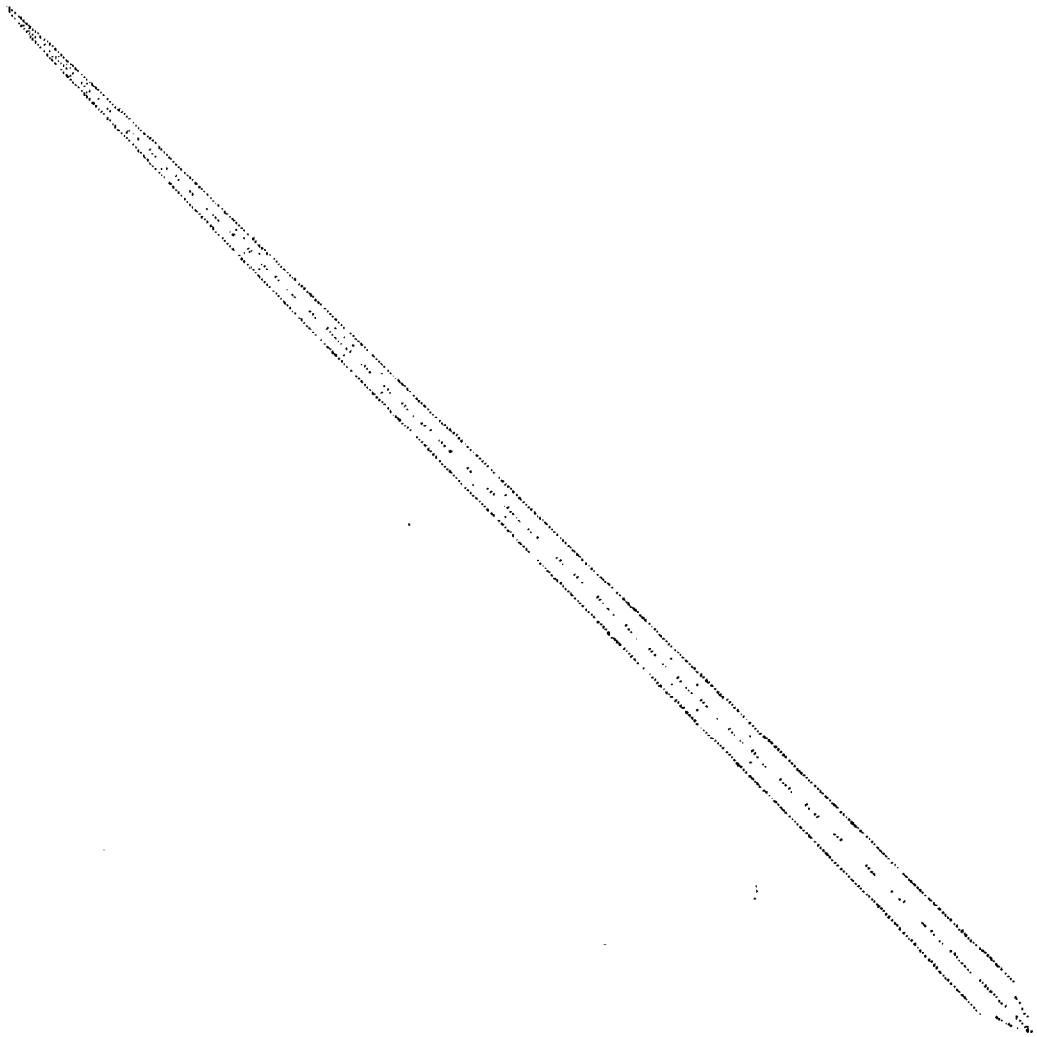
El sistema que queda después de discretizar por elementos finitos es un sistema de 8.434 ecuaciones cuyo mallado viene dado por la figura siguiente:



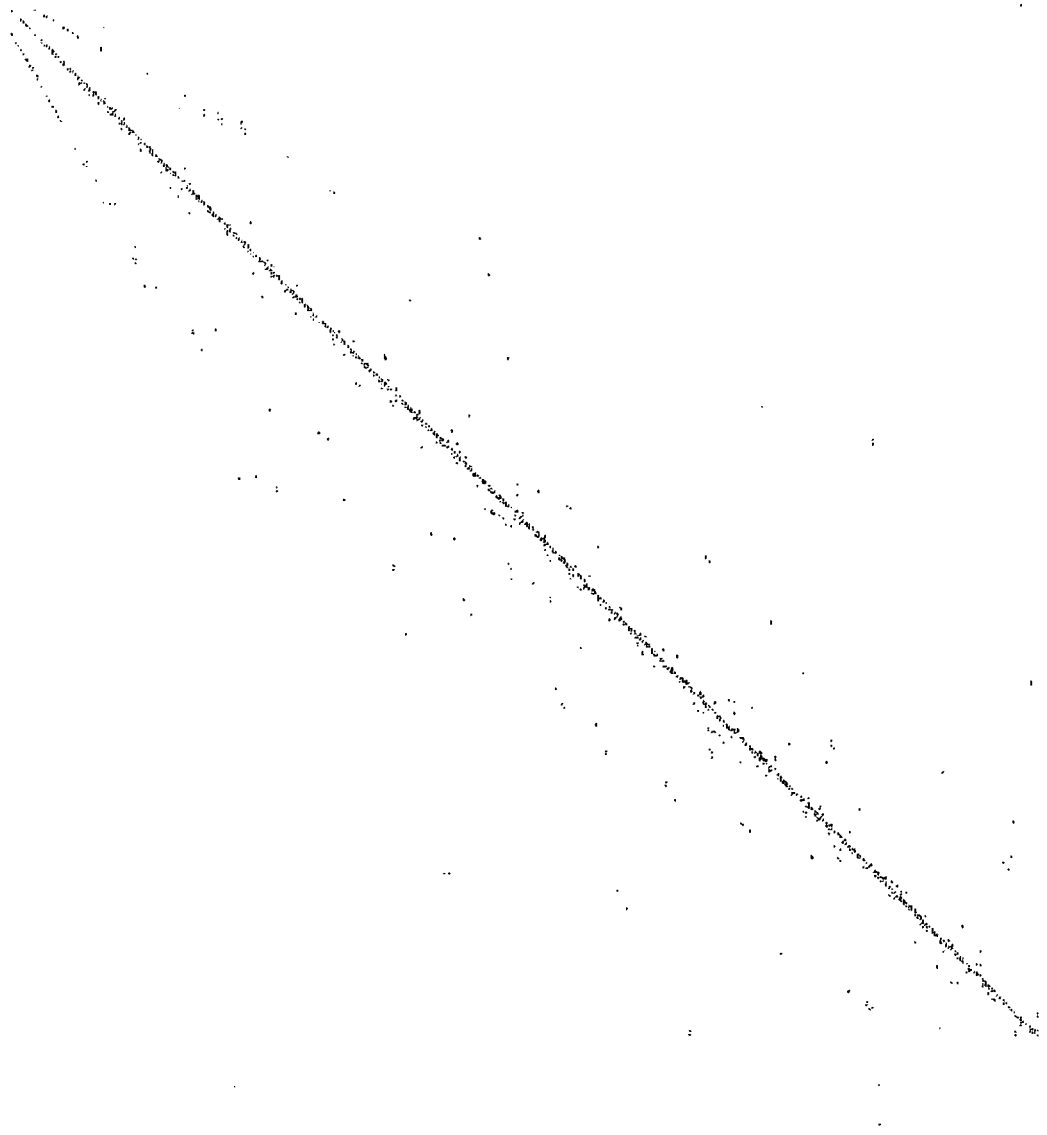
La figura siguiente es la de la matriz A del sistema sin reordenar:



La figura que sigue es la de la matriz después de pasar el algoritmo de ICM:



Por último tenemos la figura de la matriz reordenada mediante el algoritmo de grado mínimo.



Se observa que la figura correspondiente al algoritmo de grado mínimo presenta un ancho de banda mayor que la del algorimto de ICM, pero se ha reducido considerablemente el efecto fill-in y el tiempo de ejecución del sistema.

CONCLUSIONES

A las concluiones que hemos llegado en el estudio realizado en sistemas lineales sparse con matrices siméstricas y definidas positivas ayudándonos de la teoría de grafos son las siguientes:

❶ La mejora, en cuanto a almacenamiento y tiempo de cáculo, de los métodos directos tras aplicar técnicas de renumeración.

❷ Aunque el algoritmo inverso de C.M. controla de alguna manera el aumento del efecto fill-in, éste es óptimamente reducido por aplicación del algoritmo del grado mínimo.

❸ El algoritmo de grado mínimo con eliminación múltiple, controla y reduce de manera similar el efecto fill-in, con la ventaja de reducir el tiempo de ejecución.

❹ Así mismo, el refinamiento iterativo tras la resolución del sistema es prescindible en muchos casos que se reordenan por grado mínimo.

❺ En la aplicación del algoritno del grado mínimo, hay una ganancia muy significativa en almacenamiento y tienpo de ejecución, si antes de pasar el algoritmo de grado mínimo hacemos un preordenamiento de la matriz aplicando el algoritmo inverso de C.M.

❻ Estas técnicas son tambien válidas para métodos iterativos, previa ordenación por grado mínimo simple y múltiple y posterior resolución iterativa.

❼ Asimismo, también son válidas para utilización de precondicionadores después de la reordenación.

PERSPECTIVAS FUTURAS

A continuación exponemos varias cuestiones que constituyen vias futuras de investigación y que están relacionadas con los tópicos que se han tratado en esta Tesis:

❶ Desarrollar todas estas técnicas de forma amplia y completas para matrices no definidas positivas, no simétricas, ni aun de perfil simétrico.

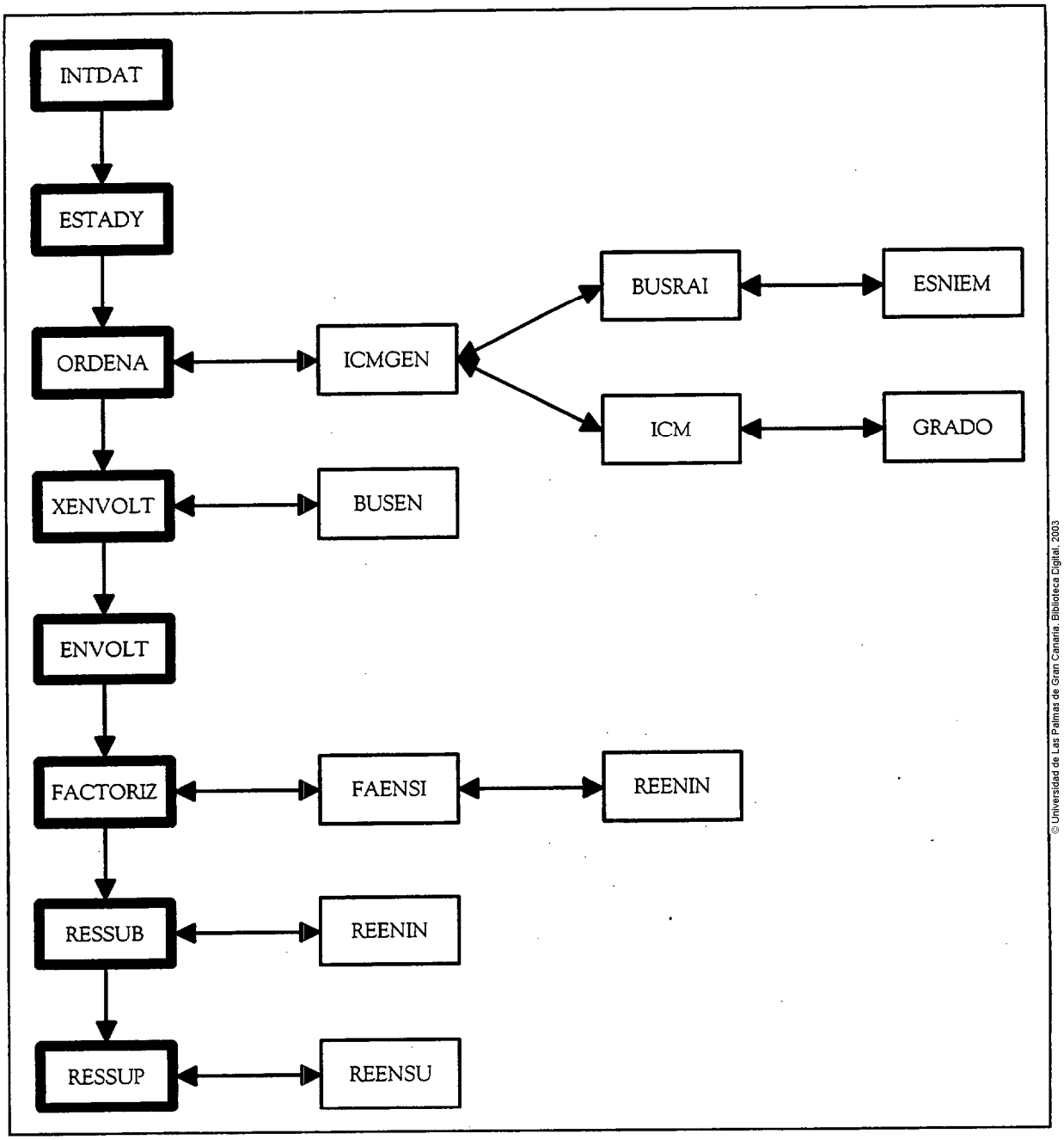
❷ Asimismo, desarrollar algoritmos combinados en los métodos One-way, Nested Dissection y el Go-Away, recientemente diseñado por profesores del Departamento de Matemáticas.

❸ En general, aplicar todas estas técnicas de renumeración a los sistemas que surgan como consecuencia de discretización de ecuaciones diferenciales.

❹ También tenemos previsto aplicar estas técnicas a problemas no lineales y evolutivos.

❺ Además las distintas técnicas de ordenamiento que hemos desarrollado en este trabajo las podemos aplicar en otros campos distintos a las matemáticas como pueden ser, en la psicometría, en el escalamiento de sujetos y estímulos respecto a una dimensión determinada en el contínuo escalar. También las técnicas de reordenamiento las podríamos aplicar en los modelos de ecuaciones simultáneas de econometría.

RELACIÓN DE CONTROL ENTRE LOS PROGRAMAS Y SUBRUTINAS DEL ALGORITMO ICM



Descripción de las principales subrutinas

❑ Subrutina INTDAT.

La subrutina intdat guarda en archivos, los valores de las entradas de la diagonal principal de la matriz del sistema A, las entradas de las matrices L y U.

❑ Subrutina ESTADY.

Esta subrutina determina la estructura de adyacencia del grafo asociado a la matriz A del sistema a resolver.

❑ Subrutina ORDENA.

Esta subrutina proviene del acoplamiento de las subrutinas ICMGEN, BUSRAI, ESNIEN, ICM y GRADO. Su misión es obtener el nuevo ordenamiento del grafo inicial asociado a A, o sea el del grafo asociado a la matriz PAP' .

❑ Subrutina ESNIEN.

Genera un nivel de estructura de la componente conexa determinada por los parámetros de entrada RAIZ, MASC, XADJ y ADYACI.

❑ Subrutina BUSRAI.

Su misión es hallar un nodo pseudo-periférico de una componente conexa de un grafo teniendo como parámetros de entrada en la componente conexa RAIZ, MASC, XADJ y ADYACI.

❑ Subrutina GRADO.

Esta subrutina calcula los grados de los nodos en una componente conexa de un grafo ,teniendo como parámetros de entrada en la componente conexa RAIZ, MASC, XADJ y ADYACI.

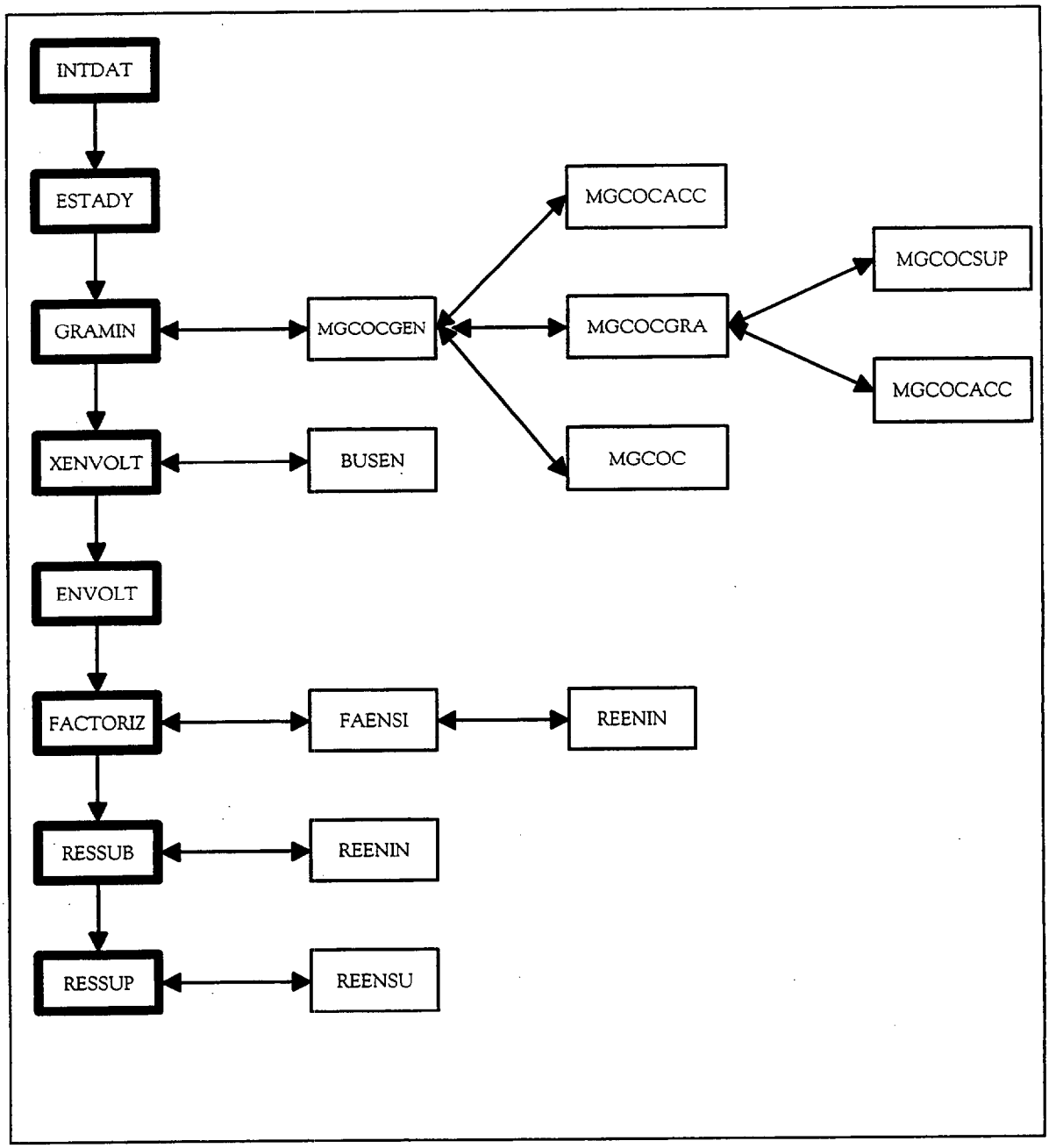
❑ Subrutina ICM.

Su misión es aplicar el algoritmo ICM para una componente conexa de un subgrafo.

❑ Subrutina ICMGEN.

Esta subrutina halla el ordenamiento ICM de un grafo general inconexo, haciendo una llamada a la subrutina ICM para enumerar cada componente conexa.

RELACIÓN DE CONTROL ENTRE LOS PROGRAMAS Y SUBRUTINAS DEL ALGORITMO DEL GRADO MÍNIMO SIMPLE



Descripción de las principales subrutinas del algoritmo del grado mínimo simple.

❑ Subrutina MGCOCGEN.

Su propósito es determinar el ordenamiento del grafo general inconexo. Los parámetros de entrada han de ser el número de ecuaciones (NUMECU) y la estruc-

tura de adyacencia (ADYACI, XADJ). Los parámetros de salida serán los vectores PERM e INVP, que determinan el ordenamiento de grado mínimo y su inverso respectivamente.

La estructura de adyacencia será empleada por la subrutina que almacena la secuencia de las estructuras del grafo cociente. Cuando MGCOCGEN haya hallado un nodo de grado mínimo entonces el conjunto accesible de este nodo a través de los supernodos eliminados mediante llamada a la subrutina MGCOCACC.

Posteriormente, mediante llamada a la subrutina MGCOCGRA, los nodos del conjunto accesible actualizan su grado.

Antes que el programa proceda a hallar el siguiente nodo de grado mínimo, se llamará a la subrutina MGCOC para que se ejecute la transformación del grafo cociente.

❑Subrutina MGCOCACC.

Esta subrutina determinará el conjunto accesible de un nodo RAIZ dado a través del conjunto de nodos eliminados. Supondremos que la estructura de adyacencia se almacena en el formato del grafo cociente.

❑ Subrutina MGCOC.

Esta subrutina ha de ejecutar la transformación del grafo cociente sobre la estructura de adyacencia.

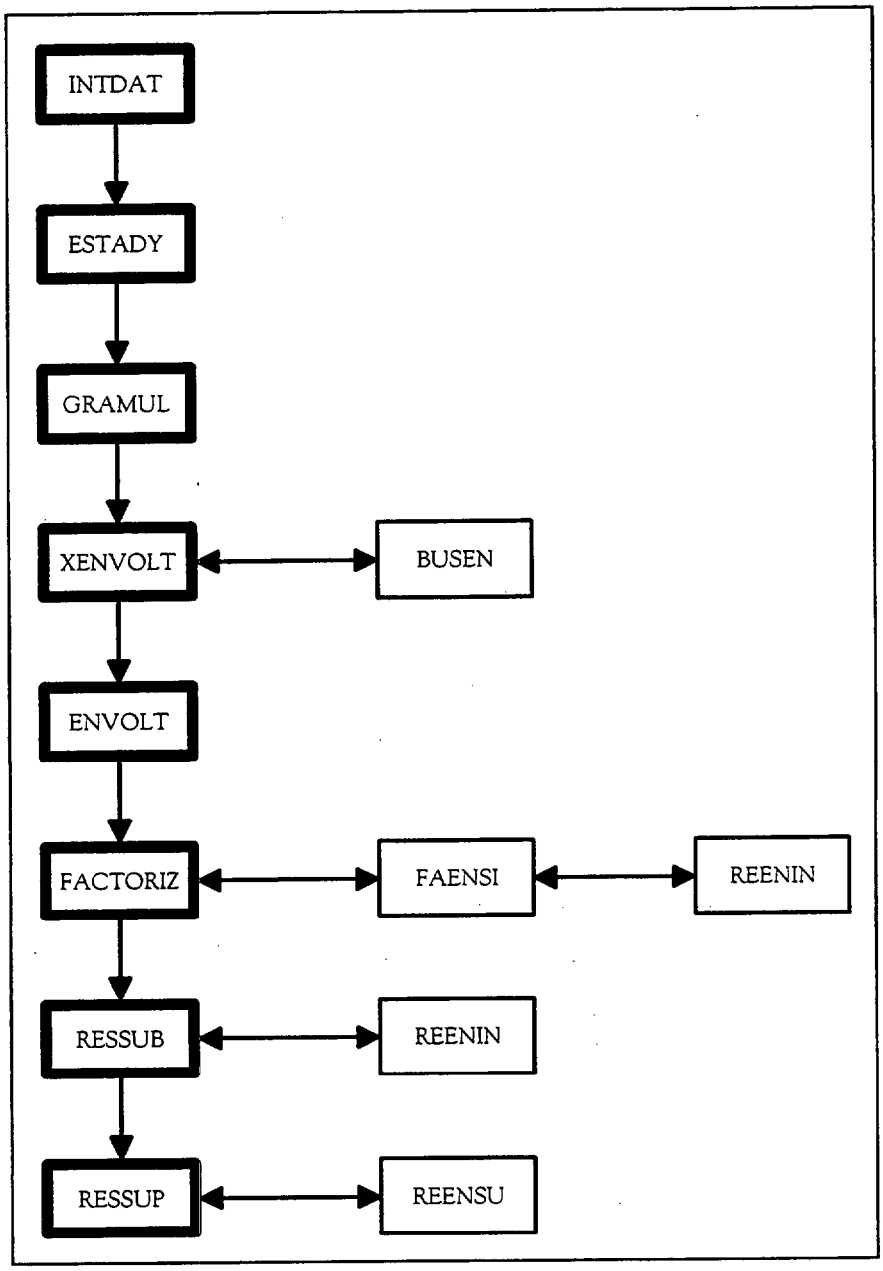
❑ Subrutina MGCOCGRA.

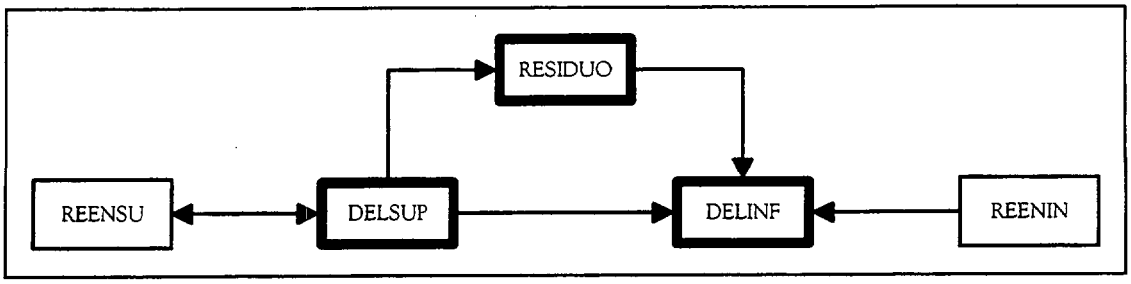
Esta subrutina ejecuta el paso de la actualización de grado en el algoritmo de grado mínimo.

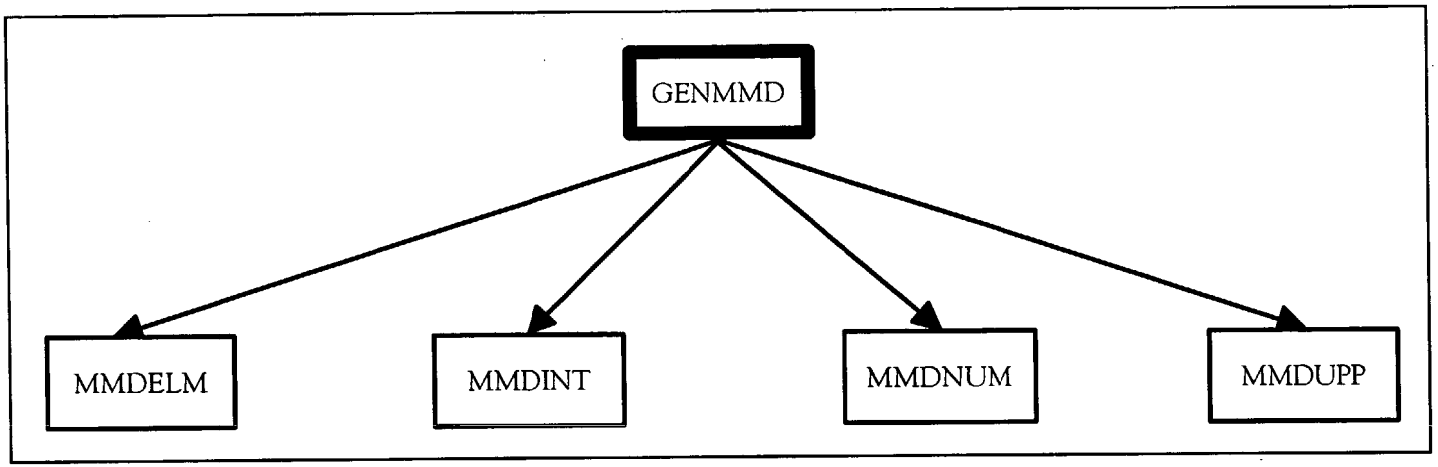
❑ Subrutina MGCOCSUP.

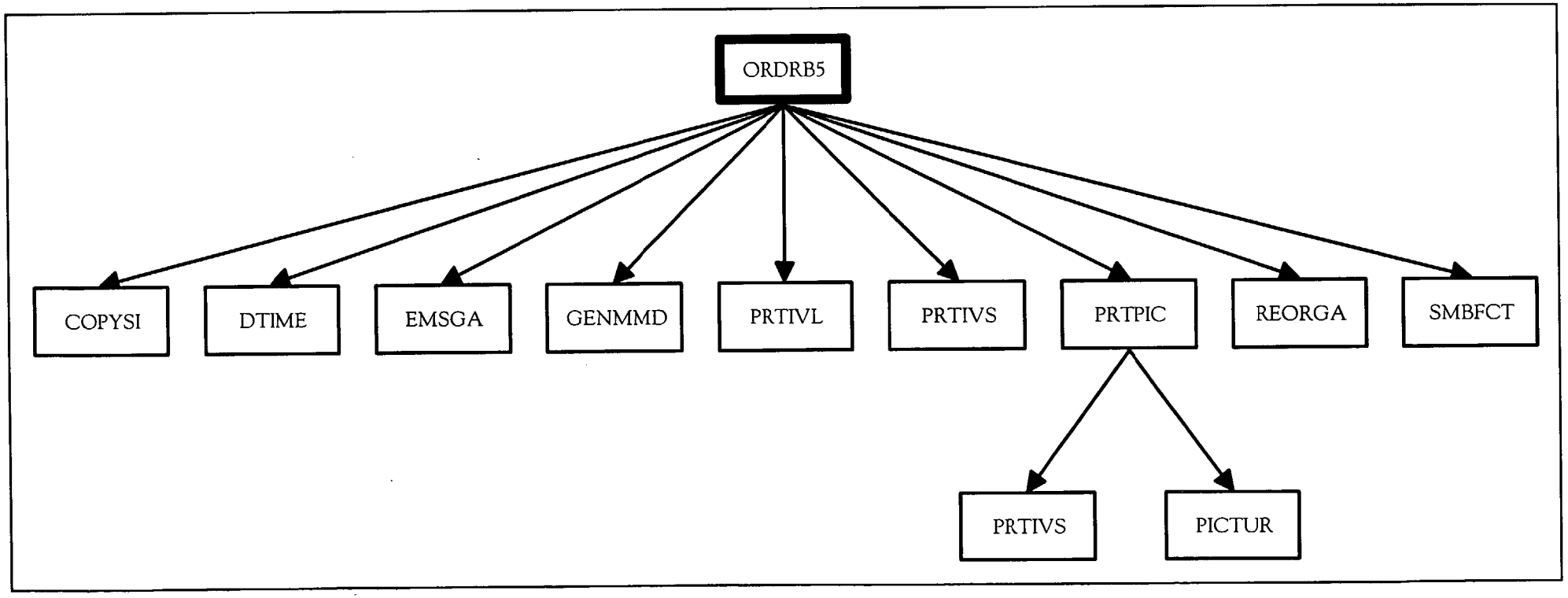
Esta subrutina fusiona nodos indistinguibles en el algoritmo de ordenamiento de grado mínimo.

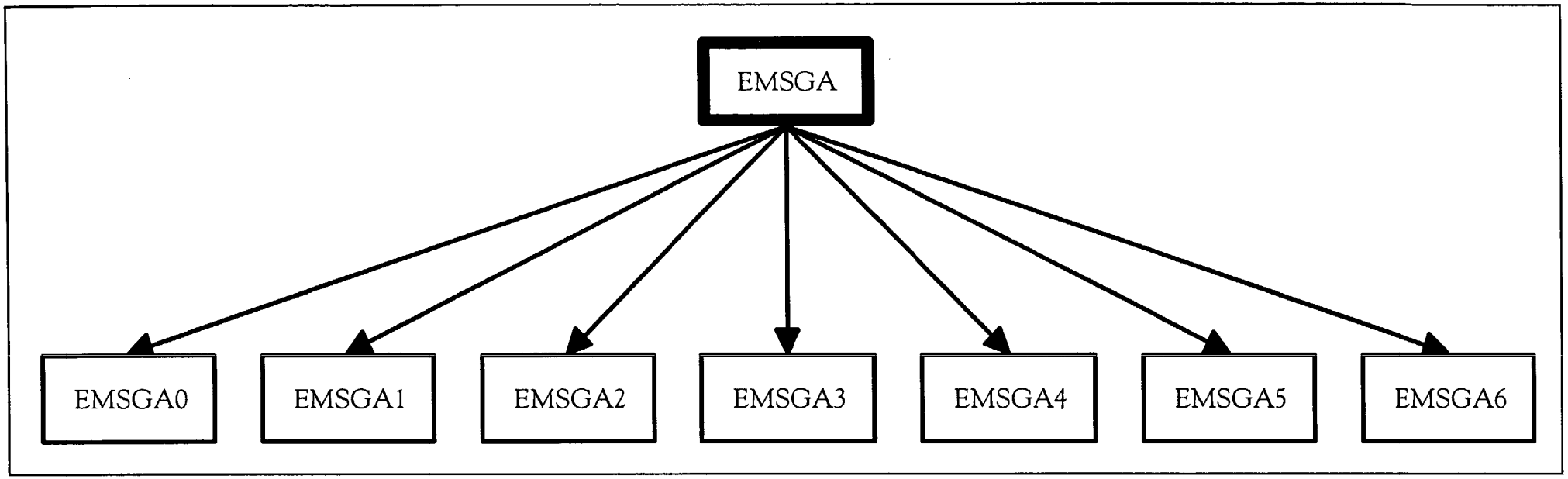
RELACIÓN DE CONTROL ENTRE LOS PROGRAMAS Y SUBRUTINAS DEL ALGORITMO DE GRADO MÍNIMO MULTIPLE

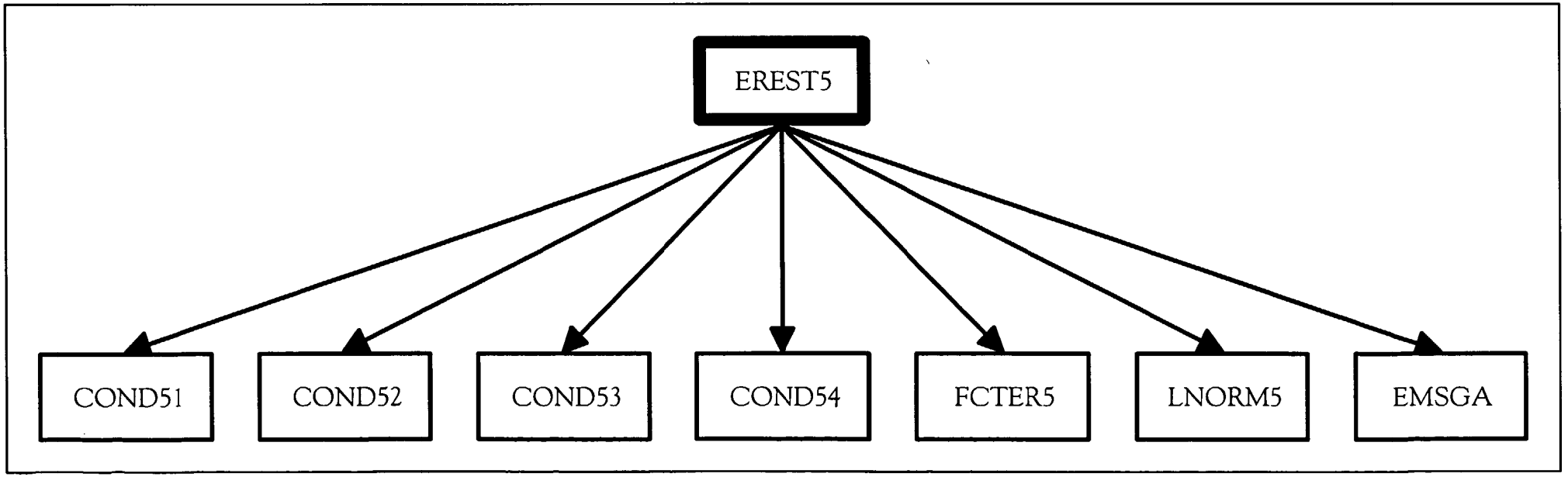


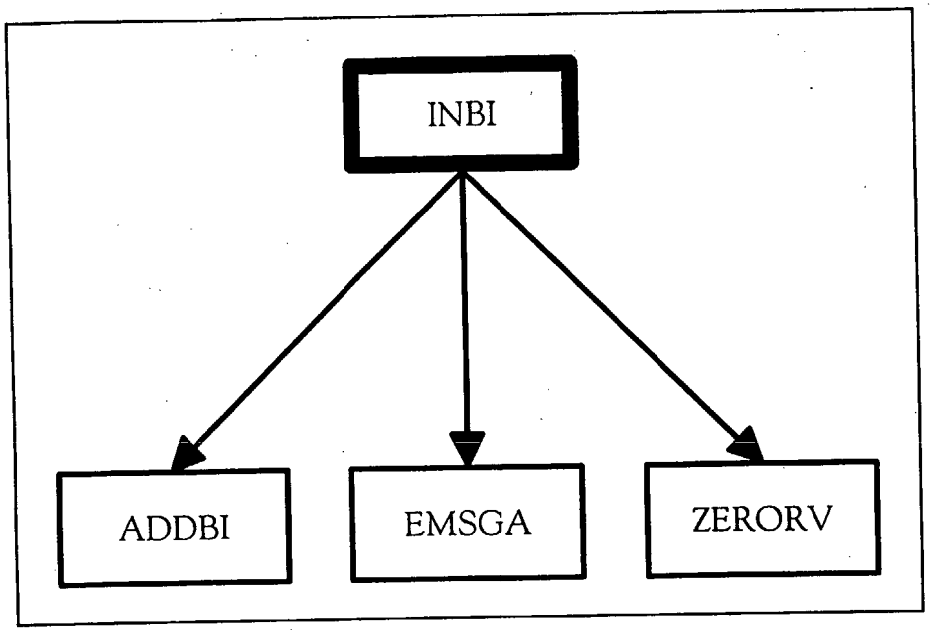


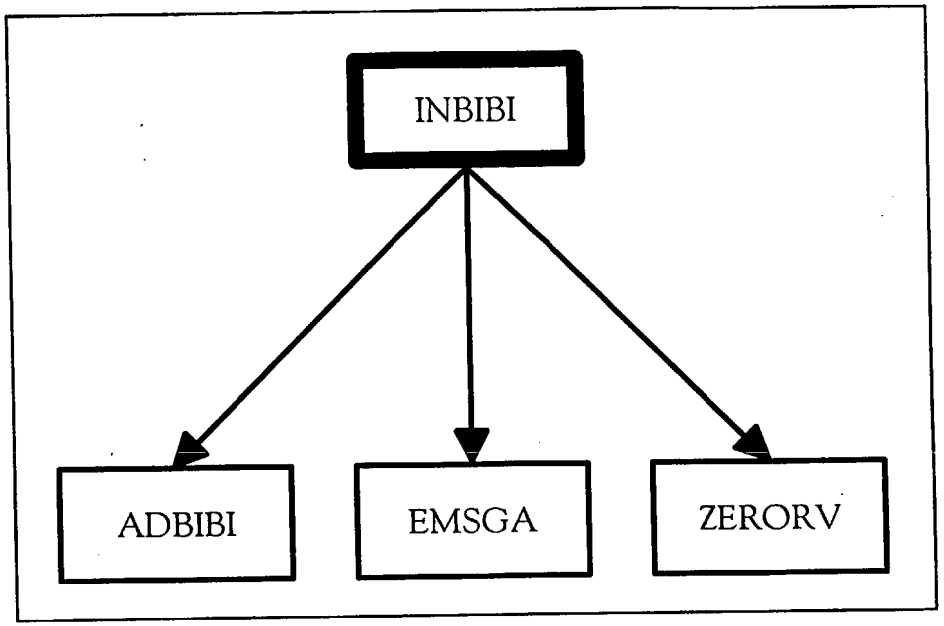


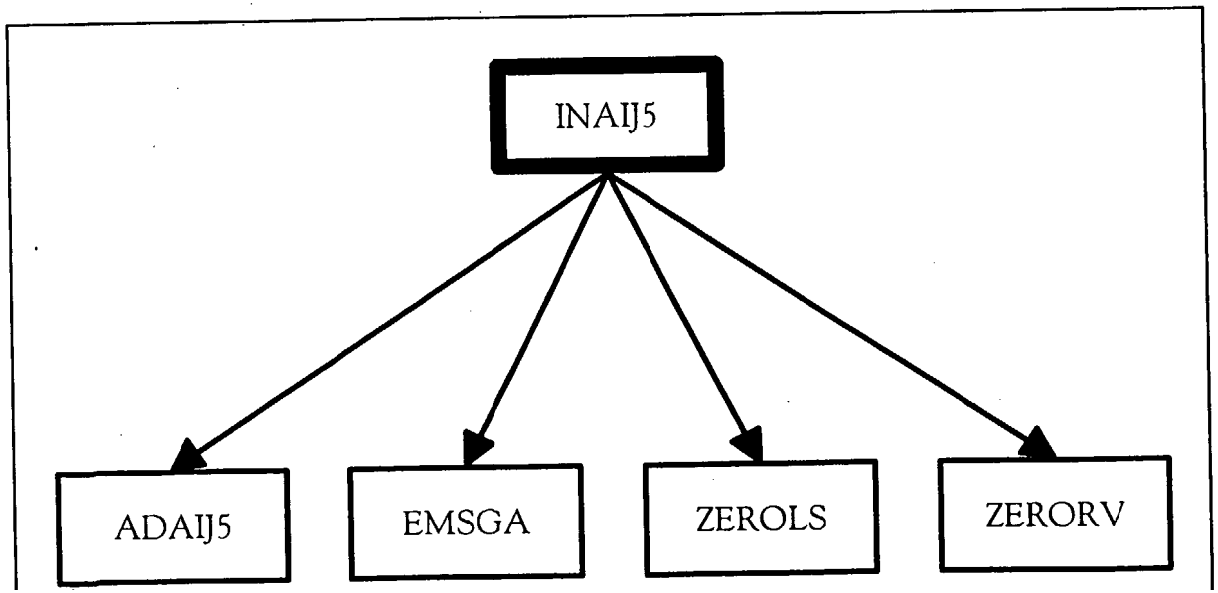


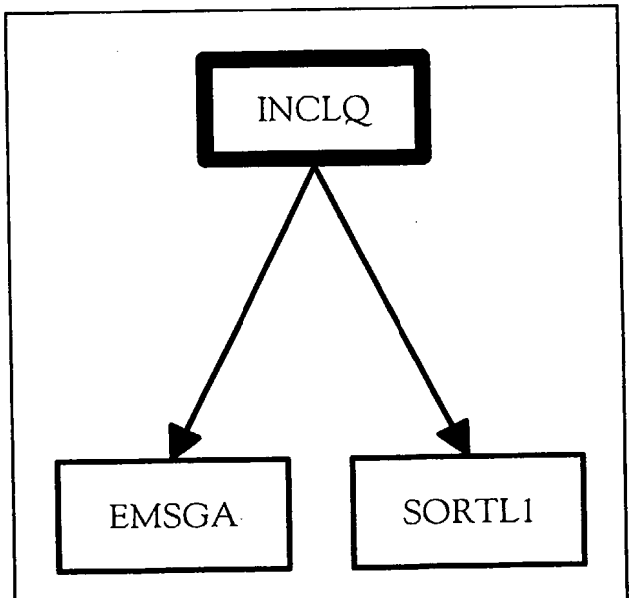












Descripción de las principales subrutinas del algoritmo del grado mínimo múltiple.

❑ **GENMMD.** Esta subroutina implementa el algoritmo del grado mínimo. Utiliza la representación implícita de los grafos de eliminación por grafos cocientes y la noción de nodos indistinguibles. También implementa las modificaciones por eliminación múltiple y grado mínimo externo.

❑ **MMDELM.** Esta subroutina elimina el nodo de grado mínimo de la estructura de adyacencia la cual se almacena en el formato de grafo cociente. También transforma la representación de grafo cociente del grafo de eliminación.

❑ **MMDINT.** Esta subroutina ejecuta la inicialización para la versión de eliminación múltiple del algoritmo de grado mínimo.

❑ **MMDNUM.** Esta subroutina ejecuta el paso final que se produce en el vector de permutación y de permutación inversa, en la versión de eliminación múltipe del algoritmo de ordenamiento del grado mínimo.

❑ **MMDUPD.** Esta subroutina actualiza los grados de los nodos después de un paso de eliminación múltiple.

❑ **ORDRB5.** Esta es la subroutina maestra para encontrar el ordenamiento de grado mínimo ,el cual se designó para reducir el relleno de una matriz simétrica. El algoritmo utiliza la idea de la eliminación múltiple y emplea el grado mínimo externo. El algoritmo se implementa usando el modelo del grafo cociente.

❑ **EMSGA.** Esta subroutina se usa para manipular los errores en el sistema.

❑ **PRTIVL.** Esta subroutina se usa para imprimir un arreglo de enteros en formato longint.

❑ **PRTPIC.** La subrutina PRTPIC llama a la subrutina PICTUR para imprimir la matriz de adyacencia de un grafo reordenado dado.

❑ **ANORM5.** Esta subrutina calcula la norma uno de una matriz almacenada usando la estrucutra de datos comprimidos.

❑ **COPYLI.** Esta subroutina copia los n elementos enteros del vector A al B. (ambos A y B son arreglos de enteros en formato longint)

❑ **EMSGA1.** Esta subroutina está diseñada para imprimir un mensaje de error para los módulos de entrada de la matriz estructural.

❑ **EREST5.** Esta es la sobroutina que dirige la estimación del error relativo en la solución calculada x para el sistema Ax=b ,donde A es una matriz simétrica que se almacena usando la estructura de datos comprimidos.

❑ **GSFCT.** Esta subroutina ejecuta la factorización simétrica para un sistema general sparse almacenado en formato de datos comprimidos.

❑ **GSSLV.** Esta subroutina ejecuta la solución de un sistema factorizado,donde la matriz se almacena en el formato sparse comprimido.

❑ **IJBEGN.** Esta sobrutina prefija los valores defectuosos de algunos de los parámetros del usuario e inicializa los bloques comunes en el paquete. Debe llamarse antes de cualquier subroutina de adyacencia de entrada.

❑ **INAIJ5.** Esta subroutina da entrada a un número dentro de la (I , J) ésima posición de la matriz almacenada usando la estructura de datos comprimidos. Asume que I es mayor o igual a J.

❑ **INCLQ.** Esta subroutina se usa para entrar una estructura de submatriz llena en el paquete. La menor estructura de adyacencia se fija en la forma de listas enlazadas, de forma tal que pueda construirse eventualmente una estructura de adyacencia.

❑ **INIJ.** Esta subroutina se usa para entrar un par de adyacencia en el paquete. Llamando repetidamente a INIJ,la menor estructura de adyacencia puede fijarse en la forma de listas enlazadas de forma tal que puede construirse eventualmente una estructura de adyacencia. Los pares pueden aparecer en cualquier orden e INIJ ignora los pares duplicados.

❑ **INIJIJ.** Esta subroutina se usa para entrar un par de una lista de adyacencia en el paquete. La menor estructura de adyacencia se fija en la forma de listas enlazadas de forma tal que pueda construirse eventualmente una estructura de adyacencia.

❑ **INMAT5.** Esta subroutina da entrada a un conjunto de números en la matriz almacenada usando la estructura de datos comprimidos.

❑ **INROW.** Esta subroutina se usa para entrar en la lista de adyacencia del paquete. Por llamadas repetidas de INROW la estructura de menor adyacencia puede fijarse en forma de listas enlazadas de forma tal que la estructura de adyacencia puede construirse eventualmente.

❑ **PICTUR.** La subroutina PICTUR imprime la matriz de adyacencia de un grafo reordenado dado. Los valores nulos se representan por asterisco *.

❑ **PRNTRV.** Esta subroutina se usa para imprimir un vector en punto flotante.

❑ **RCOPYL.** La subroutina RCOPYL copia los n elementos del vector A al vector B desde n hasta 1. Esto algunas veces es útil cuando A y B son el mismo vector. (A y B son arreglos de enteros longint)

❑ **REORGA.** Después de obtener un buen ordenamiento y la localización de la estructura de los datos, no es necesario mantener la estructura de adyacencia. REORGA se llama para reorganizar el vector de almacenamiento y ajustar los punteros en el mapa del bloque común de almacenaje.

❑ **SMBFCT.** Esta subroutina ejecuta la factorización simbólica de un sistema lineal permutado y también fija la estrucutra de datos comprimidos para el sistema.

❑ **SOLVES.** Esta es la subroutina que dirige la ejecución de la factorización y la solución por remonte o por descenso de un sistema almacenado ,usando la estructura de datos comprimidos. La solución puede encontrarse en la primera localización NEQNS del vector de almacenamiento.

BIBLIOGRAFIA

[1] F.L.ALVARADO, Manipulation and Visualization of Sparse Matrices, ORSA J. Computing 2, pp. 186-207.

[2] P.R. ALMEIDA BENITEZ, Resolución Directa de Sistemas Sparse por Grafos, Tesis Doctoral, Dep. de Mat. Aplicada e Informática y Sistemas, Univ. de Las Palmas De Gran Canaria, 1989.

[3] W.ALLEN SMITH, Análisis Numérico, Prentice Hall, México, 1986.

[4] J. ANDERSEN ET AL, The Scheduling o£ Sparse Natrix-Vector Multiplication on a Massively Parallel DAP Computer, Parallel Computing, North-Holland, 1992.

[5] L.V.ATKINSON ET AL., Numerical Methods with FORTRAN 77. A Practical Introduction, Addison-Wesley, 1989.

[6] R.BELLMAN, Introduction to Matrix Analisis, McGraw-Hill, México, 1960.

[7] H.E.BROWN, Solution of Large Networks by Matrix Methods, John Wiley and Sons, Chichester (U.K.), 1975.

[8] J. R. BUNCH, Anal i si s of the Diagonal Pivoting Method, SIAM J.Numerical Anal., pp.656-680, Philadelphia, 1971.

[9] J.R.BUNCH, of Sparse Elimination, *of* Sequential and Parallel Numerical Algorithms 8,197-220, Academic Press, New York, 1973.

[10] J.R.BUNCH ET AL., Sparse Matrix Computation, Academic Press, New York, 1976.

[11] R.L. BURDEN ET J.D. FAIRES, Análisis Numérico, Grupo Editorial Iberoamérica, México, 1985.

[12] E. CHU, Ai GEORGE, J. LIU AND E. NG, Waterloo Sparse Matrix Package. User's Guide for SPARSPAK-A.Research Report, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[13] C.CONDE LAZARO AND G.WINTER ALTHAUS, Métodos y Algoritmos Básicos del Algebra Numérica, Ed. Reverté, Barcelona, 1990.

[14] E.CUTHILL, Several Strategies for Reducing the Bandwith of Matrices, Papers *of* the Symposium on Sparse Matrices and Their Applications, IBM Thomas J.Watson Research Center, New York, 1971.

[15] J.J.DONGARRA ET AL., LINPACK Users 'Guide,SIAM, Delphia,1979.

[16] J.J.DONGARRA ET AL., Solving Linear Systems on Vector and Shared Memory Computers, SIAM, Philadelphia, 1991.

[17] I. S. DUFF ET AL. Sparse Matrix Proceedings, SIAM, Philadelphia, 1978.

[18] I.S.DUFF, A.M. ERISMAN AND J.K.REID, Direct Methods for Sparse Matrices, Monographs on Numerical Analysis, Oxford Press, Oxford, 1989.

[19] G.E. FORSYTHE ET AL., Computer Methods for Mathematical Computations, Prentice-Hall, London, 1977.

[20] J.L.DE LA FUENTE O'CONNOR, Tecnologías Computacionales para Sistemas de Ecuaciones, Optimización Lineal y Entera, Ed. Reverté, Barcelona, 1993.

[21] K.A. GALLIVAN ET AL., Parallel Algorithms for Matrix Computations, SIAM, Philadelphia, 1991.

[22] J. GASTINEL , NOEL, Analyse numérique lineaire , Hernan, Paris ,1975.

[23] A.GEORGE, AND LIU,The evolutión of the minimum degree ordering algorithm,Siam , Vol 31, pag 1 - 17.

[24] A.GEORGE ET AL., Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, London, 1981.

[25] A.GEORGE AND E.NG, Waterloo Sparse Matrix Package. User Guide for SPARSPAK-B. Research Report CS-84-37, Departarnent *of* Computer Science, University of Waterloo, Waterloo, Ontario, Canadá, 1984.

[26] A.GEORGE AND JOHN R.GILBERT, Graph Theory and sparse matrix computation, Springer-Verlag, 1993.

[27] F.G.GUSTAVSON, Some Basic Techniques for Solving Sparse Systems *of* Linear Equations, The IBM Rescarch Symposia Series, pp-41-52, Plenum Press, New York, 1972.

[28] E.HELLERMAN AND D.RARICK, The Partitioned Preassigned 4 Pivot Procedure Sparse Matrices and Their Applications, Plenum Press, New York, 1972.

[29] A. JENNINGS ET J.J. MCKEOWN, Matrix Computation, John Wiley and Sons, Chichester (U.K.), 1992.

[30] J.M.ORTEGA AND W.C.RHEINBOLDT, Introduction to Parallel and Vector Solution of Linear Systems, Plenum Press,NewYork, 1988.

[31] G.H.PAULINO ET AL., A new algoritm for finding a pseudoperipheral vertex or the endpoints of a pseudodiameter in a graph, John Wily and Sons, Chichester (U.K.), 1994.

[32] S.PISSANETZKY, Sparse Matrix Technology, Academic Press, New York, 1984.

[33] J.K.REID, Large Sparse Sets of Linear Equations, Academic Press, New York,1971.

[34] J.R.RICE, Matrix Computations and Mathematical Software, McGraw-Hill, New York, 1983.

[35] D.J.ROSE AND R.A.WILLOUGHBY, Sparse Matrices and Their Applications, Plenum Press, New York, 1972.

[36] T. SCHREIBER ET AL., A New Efficient Parallelization Strategy for the QR Algoritm, Parallel Computing 20, 63-75, Nort-Holland, 1994.

[37] Z. SHANG ET H.HONG-CI, Domain Descomposition by a Over-lapping and Prconditioned Conjugate Gradient Method, Journal of Computational Mathematics, Vol. 11, No. 1, pp. 63-72, 1993.

[38] G.W.STEWART, Introduction to Matrix Computation, Academic Press, New York, 1973.

[39] G.W.STEWART, On The Perturbation *of* LU, Cholesky And QR Factorizations, SIAM J. Matrix Anal. Appl-, Vol. 14, No. 4,1141-1145, 1993.

[40] G.STRANG, Linear Algebra and Applications, Academic Press, New York, 1976.

[41] SUAREZ SARMIENTO A. Tesis Doctoral " Contribución a algoritmos de biortogonalización para la resolución de sistemas de ecuaciones lineales.Universidad de Las Palmas de G.C .1995.

[42] R. P. TEWARSON, Sparse Matrices, Academic Press, New York, 1973.

[43] M. THUNÉ, The Partitioning Problem for a Class of Data Parallel Algorithms, Parallel Computing, North-Holland, 1992.

[44] J.W. VAN DER WOUDE, Graph Theoretic Methods for the Computation of Disturbance Decoupling Feedback Matrices for Structured Systems,Elsevier Science, New York, 1994.

[45] D.S.WATKINS, Fundamentals of Natrix Computations, John Wiley and Sons, New York, 1991.

[46] R.J. WILSON, Introducción a la teoría de grafos, Alianza Editorial, Madrid, 1983.

[47] R.J. WILSON ET J.J. WATKINS, Graphs An Introductory Approach, John Wiley and Sons, New York, 1990.

[48] H.A.G.WIJSHOFF,Direct Methods for Solving Linear Systems, Papers *of* the Large-Scale Scientific Parallel Computing Course, Abingdon (UK), 1992.

[49] D.C. WOOD, A technique for colouring a graph applicable to large scale timetabling problems. The Computer, 1969.

[50] Z.ZLATEV, Computational Methods for General Sparse Matrices, Kluwer Academic Publishers, London, 1991.

[51] * YOSEF SAAD, Iterative methods for sparse linear systems .Pws Publishing company. University of Minesota .1995.