15
2 ET

# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

## CRITERIOS SOBRE LA ELECCION DE NODOS, EN GRAFICAS ASOCIADAS A LAS MATRICES RALAS

**T E S I S**

QUE PARA OBTENER EL TITULO DE:

**A C T U A R I O**

P R E S E N T A :

CARLOS RAMON CARAVEO ACOSTA

DIRECTOR DE TESIS: M. EN C. VIRGINIA ABRIN BATULE



FACULTAD DE CIENCIAS
SECCION ESCOLAR

NOVIEMBRE DE 1995

VNIVERSIDAD NACIONAL
AVPNOMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

CRITERIOS SOBRE LA ELECCION DE NODOS, EN GRAFICAS ASOCIADAS A LAS MATRICES RALAS.

realizado por CARLOS RAMON CARAVEO ACOSTA

con número de cuenta 8730416-1 , pasante de la carrera de ACTUARIA

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

| | | |
|---|---|---|
| Director de Tesis<br>Propietario | M. EN C. VIRGINIA ABRIN BATULE | Virginia Abrín Batule |
| Propietario | DR. FERNANDO BRAMBILA PAZ | F. Brambila P. |
| Propietario | M. EN C. JOSE GUERRERO GRAJEDA | |
| Suplente | ACT. DAVID LOPEZ SERVIN | |
| Suplente | ACT. ARTURO LORENZO VALDES | |

Consejo Departamental de Matemáticas

M EN C. ALEJANDRO BRAVO MOJICA

# AGRADECIMIENTOS

**A mis Padres**

**Por su cariño, apoyo y sobre todo porque son lo más importante para mí.**

**Los quiero mucho.**

**A mis hermanos Luis y Enrique**

**Gracias por su ejemplo**

**A mi hermana Lorena**

**Espero que nos superes en todos los aspectos**

**Confío en tí**

A Leticia

Por tu ejemplo, amor y comprensión en todo momento

**Te amo**

**A Virginia**

**Por tu confianza, tiempo y estímulo**

**Mil Gracias**

# ÍNDICE

**Introducción**.....1

Capítulo I. **Conceptos Preliminares**

1.1 Sistemas Ralos.....3

1.2 Gráficas.....7

1.3 Gráficas y Matrices.....15

1.4 Almacenamiento.....19

1.5 Teoría de la Complejidad.....20

1.5.1 Algoritmos.....21

1.5.2 Complejidad.....23

Capítulo II. **Criterios para la elección de Nodos Iniciales**

2.1 Nodos Periféricos.....29

2.1.1 Algoritmo de nodos pseudoperiféricos.....29

2.1.2 Algoritmo de potencias de la matriz de adyacencias.....31

2.1.3.Algoritmo de trayectorias mínimas.....33

Algoritmo de Dijkstra.....35

Algoritmo de Floyd.....37

2.2 Nodos de Grado Mínimo.....37

Capítulo III. **Criterios para la Numeración de Nodos no Iniciales**


3.1 Algoritmo de Cuthill - Mckee.....40

3.2 Algoritmo RCM.....43

3.3 Algoritmo de Grado Mínimo.....45

3.4 Algoritmo de Disección Anidada.....46

**Caso Ejemplo**.....52

**Conclusiones**.....62

**Apéndice**.....63

**Bibliografía**.....85

# INTRODUCCIÓN

Los sistemas de ecuaciones lineales a gran escala, en muchas ocaciones tienen una característica en común, ésta es, que su matriz de coeficientes presenta una gran cantidad de elementos iguales a cero, es decir son sistemas dispersos o ralos.

Algunos de estos sistemas surgen al tratar de encontrar una solución óptima a problemas en áreas tan diversas como:

- Ingeniería, en el diseño y análisis de estructuras a diversas cargas.
- Desarrollo Social, distribución de servicios para un conjunto de comunidades.
- Planeación Urbana, simulación de luces de tránsito.
- Aviación, control de tráfico aéreo.
- Organizacionales, si en una empresa el empleado g interactúa con el empleado k , etc.

Sin embargo dada su magnitud, resultan difíciles de resolver de manera directa debido a los requerimientos computacionales necesarios, tanto para encontrar su matriz inversa o tan solo un determinante. Debido a esto, se han desarrollado métodos para explotar al máximo la raleza de éstos sistemas, almacenando en arreglos unidimensionales sólo a los elementos distintos de cero y a los subíndices necesarios para reconocer a la matriz original.

Ahora bien, el obtener un ordenamiento o permutación de las ecuaciones de un sistema con las características antes mencionadas, puede influir drásticamente, tanto en el tiempo de ejecución como en la memoria necesaria para solucionarlo. La razón principal para afirmar esto, es que durante el proceso de solución se produce generalmente un llenado, es decir, las entradas que eran igual a cero en la matriz de coeficientes, se transforman en elementos distintos de cero, lo cual afecta claramente a los recursos computacionales con los que se cuenten. El objetivo principal de realizar dicho ordenamiento, es precisamente reducir o eliminar el llenado.

Sin embargo, nos encontramos con otro problema, ya que el encontrar un ordenamiento que minimice el llenado para cualquier matriz, es un problema de combinatoria sumamente complejo clasificado como NP-Completo. Una opción para el estudio e implementación de estos ordenamientos, se tiene debido a la relación tan estrecha que existe entre matrices simétricas y gráficas, de esta manera el problema del ordenamiento de las entradas de una matriz, se transforma en el de encontrar una numeración adecuada para los nodos de su gráfica asociada.

Por lo anterior, el presente trabajo está organizado de la siguiente manera:

En el Capítulo 1, se da una introducción al manejo de sistemas ralos, una breve introducción a la Teoría de Gráficas, se plantea la relación entre gráficas y matrices, y por último se presentan los conceptos básicos de la Teoría de la Complejidad.

En el Capítulo 2, se exponen algunos algoritmos para la obtención de nodos iniciales.

El Capítulo 3, plantea diversos algoritmos para la obtención de reordenamientos para los nodos de una gráfica, que pretenden reducir el ancho de banda, el llenado y el perfil entre otros.

Un caso ejemplo, que mustra los distintos anchos de banda y perfiles, que se obtienen al tomar distintos nodos como iniciales, conclusiones y finalmente un apéndice, que muestra la implementación computacional de la mayoría de los algoritmos presentados.

# CAPÍTULO I

# CONCEPTOS PRELIMINARES

## 1.1 SISTEMAS RALOS

Una forma de resolver un sistema de ecuaciones lineales de la forma Ax=b, es utilizando el método eliminación Gaussiana. Este consiste en obtener una matriz U equivalente a A que sea triangular. Sin embargo, existen sistemas de ecuaciones con características especiales que permiten el desarrollo de diversos métodos para encontrar una solución, reduciendo considerablemente el número de operaciones elementales. En particular, sistemas en donde A sea simétrica y definida positiva, esto último significa que cualquier vector x de A distinto de cero, cumpla con la propiedad de que $x^tAx \geq 0$.

Una solución de este tipo de sistemas, utiliza una variante del método de eliminación Gaussiana conocida como el método de Cholesky, el cual consiste en realizar una factorización de A en $LL^t$ donde L es un factor triangular. La i-ésima descomposición de la matriz A en submatrices $H_0, H_1, \ldots, H_n$, es:

$$A_i = H_i = \begin{pmatrix} d_{i+1} & v_{i+1}^t \\ v_{i+1} & H_{i+1} \end{pmatrix}$$

En términos generales, está descomposición consta de un escalar positivo **d**, una submatriz H' y v que es un vector cuya dimensión al igual que la de H', varía de acuerdo al paso en la descomposición.

$$A = \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & H' \end{pmatrix} \begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I_{n-1} \end{pmatrix}$$

donde $H' = H - vv^t / d$. Ahora podemos factorizar $H'$ en $L_{H'} L_{H'}{}^t$

$$A = \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & L_{H'} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & L_{H'}{}^t \end{pmatrix} \begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I_{n-1} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & L_{H'} \end{pmatrix} \begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & L_{H'}{}^t \end{pmatrix} = LL^t$$

De esta manera obtenemos la factorización $A=LL^t$ y al sustituirla en nuestro sistema original, tenemos:

$$LL^tx=b$$

Ahora, si asignamos $y=L^tx$, el problema original se reduce a resolver los sistemas triangulares :

$$Ly=b \ , \ L^tx=y$$

Además esta factorización existe y es única, siempre y cuando la matriz A es simétrica y definida positiva, como habíamos señalado[1] .

Uno de los inconvenientes de este método, es que al encontrar el factor L, se produce generalmente un llenado, es decir, las entradas que eran cero en la matriz A se hacen distintas de cero en L.

Este llenado se obtiene como la diferencia entre los elementos distintos de cero de la matriz A, denotados por:

$$Nonz(A) = \{ (i,j) \mid aij \neq 0 \text{ con } i \neq j \}$$

y los elementos distintos de cero de la matriz de llenado F de'A:

$$Nonz(F) = \{ (i,j) \mid \lambda ij \neq 0 \text{ con } i \neq j \}$$

esta matriz de llenado, se define como la suma de los factores L y $L^T$. De esta manera podemos establecer que:

$$llenado(A)=Nonz(F) - Nonz(A)$$

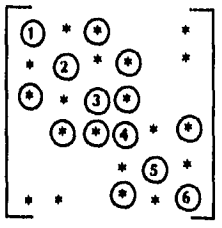
Dada esta relación y suponiendo que siempre se genera algún llenado, se tiene que:

$$Nonz(A) \subset Nonz(F)$$

[1] Para una demostración, ver George Alan [12 | pp. 15-16.

Ejemplo.

F(A) =



Los elementos distintos de cero *, en A son:

$$Nonz(A) = \{ (1,2),(1,6),(2,3),(2,6),(4,5),(5,6) \}$$

El llenado de A, denotado por ⊛, está conformado por los elementos:

$$llenado(A) = \{ (1,3),(2,4),(3,4),(4,6) \}$$

En ambos conjuntos sólo se tomó el triángulo superior de la matriz por ser simétrica.

Claramente, el llenado altera la posibilidad de resolver el sistema con un mínimo de operaciones elementales, así como la reducción en el almacenamiento si se utiliza algún procesador en su solución.

Por este motivo se han desarrollado métodos que evitan o reducen en la mayoría de los casos este llenado. Uno de estos consiste en dar un reordenamiento a las entradas de una matriz A, que equivale a realizar permutaciones de renglones y/o columnas, obteniendo una matriz $PAP^t$ equivalente a A. Este tipo de reordenamientos tienen sentido, si la matriz A posee una gran cantidad de elementos iguales a cero, es decir, si A es una matriz *rala* o dispersa.

Existen conceptos más precisos para definir la raleza de una sistema de ecuaciones, por ejemplo Brayton (1970), propone un número constante de elementos distintos de cero por renglón (2-10), según el tamaño de la matriz; Alvarado (1979), considera que una matriz de orden n es rala, si el número de elementos distintos de cero es $n^{1+\psi}$ con $\psi<1$. Así en una matriz de orden 500 y $\psi=0.9$, los elementos no nulos serían 134,290 aproximadamente.

Una permutación o reordenamiento óptimo de una matriz A, se obtiene cuando logramos colocar a sus elementos distintos de cero lo más cerca posible de la diagonal principal. Este conjunto de elementos constituyen la Banda de la matriz, y a la distancia del elemento más alejado a la diagonal, se le conoce como el ancho de Banda de A. Estos conceptos se precisan a continuación.

Si tenemos una matriz A de nxn simétrica, $f_i(A)$ indica el índice de la columna en donde se localiza el primer elemento distinto de cero para i-ésimo renglón de A, es decir:

$$f_i(A) = \min \{ j \mid a_{ij} \neq 0 \}$$

Por lo que podemos definir el i-ésimo ancho de banda de A como:

$$\beta_i(A) = i - f_i(A)$$

que representa la distancia del primer elemento distinto de cero del i-ésimo renglón a la diagonal, y como la matriz es simétrica podemos establecer que:

$$f_i(A) \leq i \qquad y \qquad \beta_i(A) \geq 0$$

Así en forma general para toda la matriz, se puede definir lo siguiente:

*Def.* El ancho de banda de una matriz A, simétrica y definida positiva es:

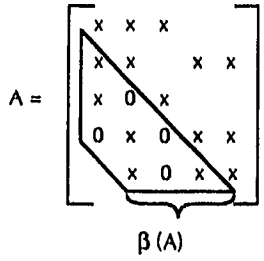
$$\beta(A) = \max \{ \beta_i(A) \mid 1 \leq i \leq N \}$$

$$= \max \{ |i - j| \mid a_{ij} \neq 0 \}$$

Por lo tanto, la región que contiene a todos los elementos desde $\beta(A)$ hasta la diagonal principal es:

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

Ejemplo.




| i | $f_i$ (A) | $\beta_i$ (A) |
|---|---|---|
| 1 | 1 | 0 |
| 2 | 1 | 1 |
| 3 | 1 | 2 |
| 4 | 2 | 2 |
| 5 | 2 | 3 |

En este caso, la región de la matriz encerrada en el trapecio, es la Banda de A, y como puede apreciarse su ancho es 3.

Este problema de reducir el ancho de Banda, se originó el los años 50's cuando Ingenieros al analizar diversas características de las estructuras de acero, como la elasticidad y resistencia a diversas cargas, las cuales generaban sistemas de ecuaciones muy grandes y ralos, notaron que el tiempo para obtener una matriz Inversa o determinantes se reducía considerablemente, si colocaban las entradas distintas de cero en una estrecha banda a lo largo de la diagonal principal[2].

Una alternativa para el estudio de sistemas de ecuaciones con las características antes mencionadas, es el uso de gráficas asociadas a las matrices de coeficientes de dichos sistemas, la importancia y utilidad de estas gráficas se discutirán más adelante. A continuación se dan algunos conceptos de la Teoría de Gráficas que serán de utilidad en el desarrollo del presente trabajo.

## 1.2 GRÁFICAS.

- Una gráfica G=(V, A) consiste de:

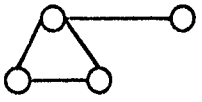
i) Un conjunto finito, no vacío, de elementos $V=(v_1, v_2, v_3, ..., v_n)$ llamados nodos, vértices o puntos.

[2] Referencia obtenida de Chinn P.Z., Chvátalová J. , Dewdney,A.K., Gibbs, N.E. [6 ]. pp. 224-225.

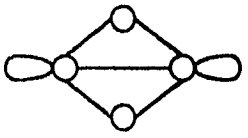
ii) Un subconjunto A, del producto cartesiano VxV, cuyos elementos se conocen como aristas o arcos si están dirigidos.

Las gráficas se pueden clasificar en:

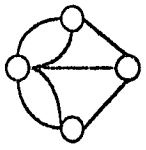
- Gráficas simples. Son aquellas estructuras en las cuales, cada par de vértices está unido sólo por una arista.



- Pseudográficas. Son gráficas que contienen lazos, es decir, aristas que unen a un vértice con él mismo.



- Multigráficas . A cada par de vértices los puede unir una o más aristas.



- Dos vértices son vecinos o adyacentes, si existe una arista que los une.

- La valencia o grado de un vértice, es el número de nodos adyacentes a él. Se denota como $val_G(v)$.

- Se dice que v, es un vértice aislado, si $val_G(v)=0$, y si la $val_G(v)=1$, se dice que es un vértice terminal.

- El grado mínimo de una gráfica.G, es la valencia mínima del conjunto de vértices que la conforman. Así:

$$\delta(G) = \min (val_G(u)) \; ; \; u \in V(G).$$

- Análogamente, el grado máximo de alguna gráfica G, es la valencia máxima que se obtiene de su conjunto de vértices V. Se denota como:

$$\Delta(G) = \max \{val_G(u)\} \; ; \; u \in V(G).$$

- Un camino ( $v_0$ , $a_0$ , $v_1$ , $a_1$ ,...,$a_{n-1}$ ,$v_n$ ), es una sucesión alternada de vértices y aristas de G, que inicia y termina en un vértice.

- Una gráfica conexa, es aquella en la cual para cualquier par de vértices (u,v), podemos encontrar un camino que los una.

Ejemplo.

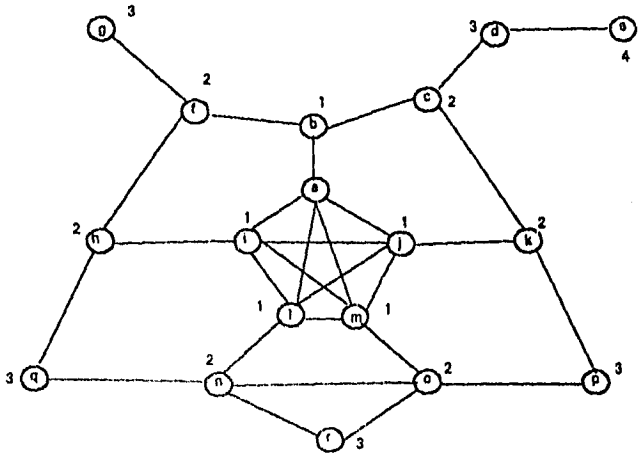



*Fig. 1*

- Un paseo en una gráfica G, es un camino en el que no se repiten aristas.
  En la figura un paseo es (a,i,m,a,b,c,d,e)

- Una trayectoria en una gráfica G, es un paseo en el que no se repiten vértices.
  Por ejemplo (e,d,c,k,j,l,n,q)

- Un camino cerrado, es un camino cuyo vértice inicial coincide con el vértice final.

- Un ciclo, es un camino cerrado $C=(v_0, v_1, v_2, \ldots, v_n, v_0)$, donde $v_i \neq v_j \ \forall\ i \neq j$. Por ejemplo (f,b,a,i,h,f)

- La distancia d(u,v), entre dos vértices de G, es la longitud de una trayectoria mínima que los une. De esta manera podemos decir que:

  1) $d(u,v) \geq 0$

  2) $d(u,v)=0, \Leftrightarrow u=v$

  3) $d(u,v)=1, \Leftrightarrow u(ady_G)v$

  4) $d(u,v)=d(v,u)$

  5) $d(u,v)+d(v,w) \geq d(u,w)$

  6) $d(u,v)=\infty$, si u ó v son puntos aislados

  7) $d(u,v) \in \mathbf{N}$.

En la figura 1, se indican las distancias del vértice **a**, a cada uno de los vértices restantes de G.

- La excentricidad del vértice u, es la distancia máxima que existe entre él, y cualquier otro vértice de G. Se denota como:

$$e=\max\{d(u,v)\}.$$

En la figura 1, tenemos las distancias del vértice **a**, a todos los vértices restantes, y la más grande es al vértice e, por lo que e(a)=4.

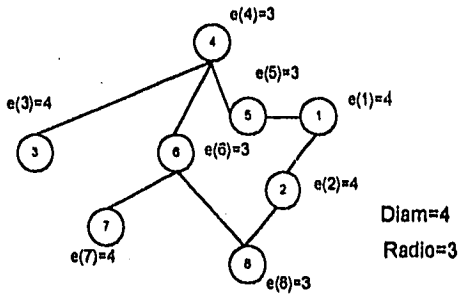
- El diámetro de una gráfica G, es la excentricidad máxima del conjunto de vértices de G. Se denota como:

$$Diam(G)=\max\{e(u)\}.$$

- El radio de una gráfica G, es la excentricidad mínima del conjunto de vértices de G. Se denota como:

$$Radio(G)=\min\{e(u)\}.$$

Ejemplo.




- Una gráfica completa, es aquella en donde todo par de vértices es adyacente.

- Una subgráfica de G, es una gráfica H donde $V(H) \subset V(G)$ y $A(H) \subset A(G)$.

- H es subgráfica inducida de G, si :

  1) H es subgráfica de G.
  2) $\forall\, u,v \in V(H) \Rightarrow a \in A(H)$, si $a=(u,v) \in A(G)$.

- El centro de G, es la subgráfica inducida por el conjunto de vértices con excentricidad mínima.

$$Centro(G) = < \{v \in V(G) \mid e(v) = Radio(G)\} >$$

- La periferia de G, es el conjunto de vértices con excentricidad máxima.

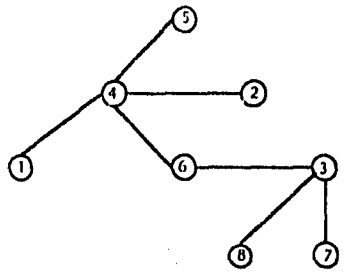
$$Periferia(G) = \{v \in V(G) \mid e(v) = Diam(G)\}$$

- Un *clan*, es una subgráfica completa, que es maximal con respecto a la contención, en la figura 1, los nodos (a,i,l,m,j), y las aristas que los unen, forman la subgráfica completa $k_5$.

- Un árbol, es una gráfica $T = [V', A']$ conexa y acíclica.

Ejemplo.

T=




Sea G=[V,A] una gráfica con n vértices. Para n≥2, las siguientes definiciones son equivalentes y caracterizan a un árbol.

a) G es conexa y acíclica.

b) G es acíclica y tiene n-1 aristas.

c) G es acíclica y si se agrega una arista se forma exactamente un ciclo.

d) G es conexa y tiene n-1 aristas.

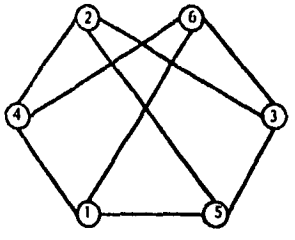
e) G es conexa pero deja de serlo si se elimina una arista.

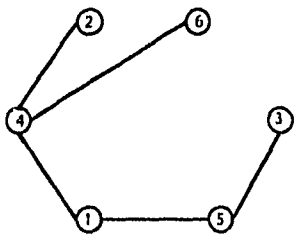
f) Existe en G, una única trayectoria entre todo par de vértices.

- Sea G=[ V,A] una gráfica. Un árbol generador de G, es una subgráfica T=[V,A'] que es un árbol.

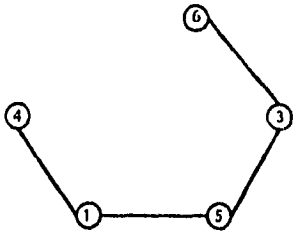
Ejemplo.

Sea G=






Árbol generador de G




Árbol no generador de G

- Sea G=[V,A] una gráfica, y sea s un elemento de V; entonces s se llama *raíz* de G, si existe un camino de s a x para todo $x \in V$.

- Una arborescencia, es un árbol que tiene una raíz.

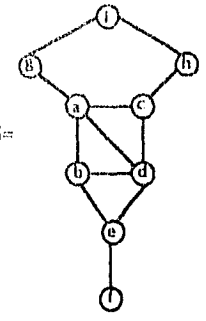
- La estructura de nivel N(v), de una gráfica G=(V,A) es una partición de sus vértices en subconjuntos, así:

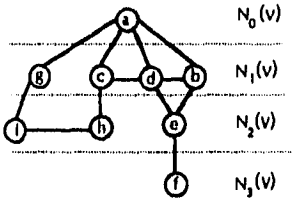
$$N(v)=\{N_0(v), N_1(v),\ldots,N_{e(v)}(v)\}$$

Donde $N_i(v)$, con i=1,...,e(v), representa el conjunto de vértices adyacentes al nivel i-1.

Ejemplo.

G=




Estructura de nivel cimentada en a

- La *profundidad* de una estructura de nivel, es el número de niveles que contenga.

- Al número de elementos del nivel $N_i(v)$, se le conoce como la *anchura* de ese nivel.

- Sea G=(V,A) una gráfica. Decimos que f: V → **N**, es una *numeración* de V si para cada v ∈ V , f (v)=n , donde n es un numero natural, y además f es inyectiva.

En las gráficas también nos encontramos con el problema de reducir el ancho de banda, este surgió en el laboratorio de propulsión de Pasadena California en 1962, en donde cierta clase de errores en los cálculos, se representaban como diferencias de aristas en un hipercubo cuyos vértices eran claves de un código que al aproximarse minimizaban el error absoluto en los cálculos[3] .

- El ancho de banda relativo de una gráfica, es la diferencia máxima de todo par de vértices adyacentes en una numeración f, de los nodos de G.

$$\beta_f(G)=\max\{|f(v_i)-f(v_j)|\}\text{ , donde } v_i \text{ ady } v_j.$$

- El ancho de banda de una gráfica G, es el mínimo ancho de banda relativo de todas las numeraciones f. A este conjunto de numeraciones se le denomina $N_G$. Por lo que:

$$\beta(G)=\min\{\beta_f(G)\}\ f \in N_G.$$

A continuación, se presentan algunos teoremas, que permiten establecer límites superiores e inferiores para el ancho de banda de cierto tipo de gráficas[4] .

Teorema 1.2.1 Si H es una subgráfica de G, entonces $\beta(H) \leq \beta(G)$

Teorema 1.2.2 Si G tiene componentes $G_1, G_2, \ldots, G_m$ , entonces

$$\beta(G) = \max\{\beta(G_1), \beta(G_2), \ldots, \beta(G_m)\}$$

Teorema 1.2.3 Sea G una gráfica cualquiera, si tomamos a p como el número de vértices y q el número de aristas, el ancho de banda es:

[3] Ibid., pág. 224
[4] Para una consulta más amplia y algunas demostraciones consultar Chvátal V.[7 ], Chvátalová J. [8 ] y Dewdney A. K.[10 ]

$$\beta(G) \geq p - \frac{2 - [(2p-1)^2 - 8q]^{1/2}}{2}$$

Teorema 1.2.4 Para cualquier gráfica G, se cumple que:

$$\beta(G) \geq [\Delta(G)/2]$$

Teorema 1.2.5 Para cualquier gráfica G sin triángulos:

$$\beta(G) \geq |(3\delta(G) - 1)/2|$$

Teorema 1.2.6 Si G es una gráfica conexa, entonces:

$$[(p-1)/\mathrm{Diam}(G)] \leq \beta(G) \leq p - \mathrm{Diam}(G)$$

Teorema 1.2.7 Para cualquier gráfica conexa G :

$$\beta(G) \leq \Delta(G) \cdot (\Delta(G) - 1)^{e(G)-1}$$

## 1.3 GRÁFICAS Y MATRICES.

Como se mencionó anteriormente, dada una matriz A simétrica de orden n, se le puede asociar una gráfica G=(V,A), con n nodos.

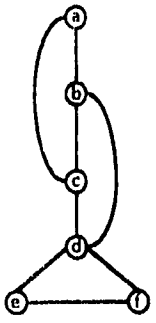
Esta gráfica nos ayuda visualizar las relaciones que podemos encontrar entre las entradas de dicha matriz. Una de estas relaciones es la de adyacencia que se describe a continuación.

•Dada una gráfica G= [V,A] con n nodos, podemos asociarle una matriz de adyacencia B, cuyas entradas son de la forma:

$$b_{ij} = \begin{cases} 1, & \text{si el nodo } i \text{ es adyacente al } j \\ 0, & \text{en otro caso} \end{cases}$$

Ejemplo.

G=




B=

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| a | 0 | 1 | 1 | 0 | 0 | 0 |
| b | 1 | 0 | 1 | 1 | 0 | 0 |
| c | 1 | 1 | 0 | 1 | 0 | 0 |
| d | 0 | 1 | 1 | 0 | 1 | 1 |
| e | 0 | 0 | 0 | 1 | 0 | 1 |
| f | 0 | 0 | 0 | 1 | 1 | 0 |

Al obtener la factorización de Cholesky, como se estableció en la sección 1.1, se crean submatrices $H_0$ , $H_1$ ,..., $H_n$ . El efecto de obtener estas matrices en la gráfica asociada se puede representar por medio de las Gráficas de Eliminación. Estas se obtienen eliminando nodos y agregando aristas entre los vecinos del nodo eliminado, que no eran adyacentes entre sí. Estas aristas representan el llenado de la matriz, descrito también en la primera sección de este capítulo.

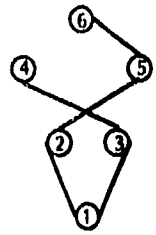
Sea Go, una gráfica y Ho su matriz asociada. Las gráficas de eliminación se obtienen de la siguiente manera:

Para i=1,...,n

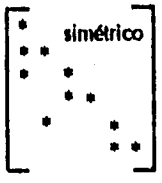
1) Eliminar el nodo $x_i$ y sus aristas.

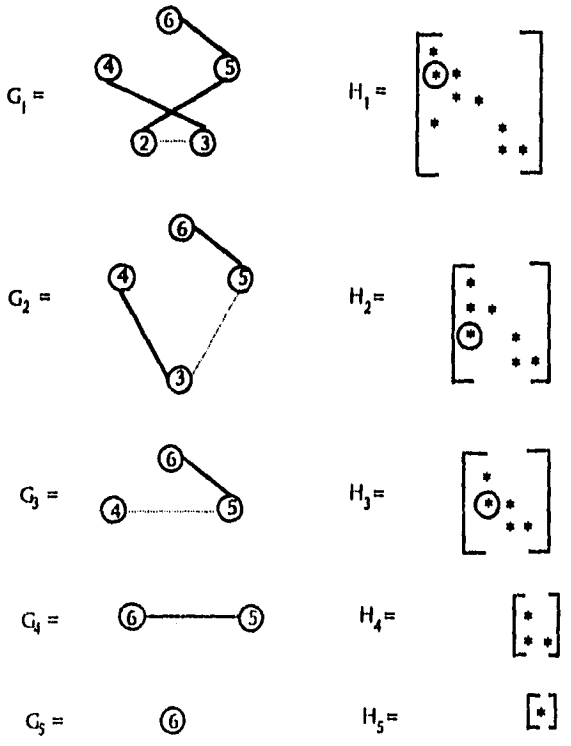
2) Agregar las aristas necesarias para que los nodos que eran adyacentes al $x_i$ , sean adyacentes entre sí.

$G_0$=

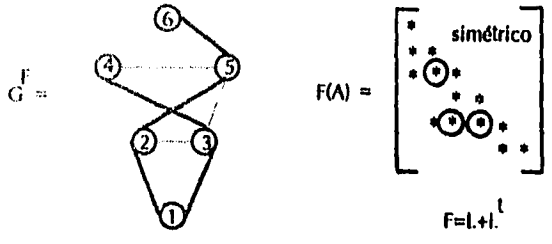



$H_0$=






La siguiente gráfica, muestra las aristas que se agregaron como resultado del proceso de la eliminación con líneas punteadas y la matriz F(A), es su matriz de llenado.



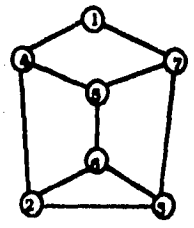

Este concepto de Gráficas de Eliminación , es utilizado para dar un ordenamiento a los nodos de una gráfica con el Algoritmo de Grado Mínimo que se discutirá en el Capítulo III.

Para comprender mejor el efecto que provoca el dar una nueva numeración a los nodos de una gráfica sobre las entradas de su matriz asociada, tenemos la siguiente secuencia que muestra los distintos anchos de banda que se pueden obtener para cada numeración.

Ejemplo.

$$A_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} x & 0 & 0 & x & 0 & 0 & x \\ 0 & x & x & x & 0 & x & 0 \\ 0 & x & x & 0 & 0 & x & x \\ x & x & 0 & x & x & 0 & 0 \\ 0 & 0 & 0 & x & x & x & x \\ 0 & x & x & 0 & x & x & 0 \\ x & 0 & x & 0 & x & 0 & x \end{bmatrix}$$

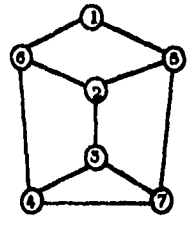
$G_0^A$ -



$\beta_0 = 6$

$$A_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} x & 0 & 0 & 0 & x & x & 0 \\ 0 & x & x & 0 & x & x & 0 \\ 0 & x & x & x & 0 & 0 & x \\ 0 & 0 & x & x & 0 & x & x \\ x & x & 0 & 0 & x & 0 & x \\ x & x & 0 & x & 0 & x & 0 \\ 0 & 0 & x & x & x & 0 & x \end{bmatrix}$$

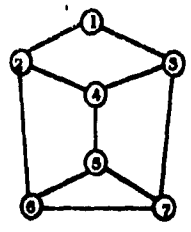
$G_1^A$ -



$\beta_1 = 5$

$$A_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} x & x & x & 0 & 0 & 0 & 0 \\ x & x & 0 & x & 0 & x & 0 \\ x & 0 & x & x & 0 & 0 & x \\ 0 & x & x & x & x & 0 & 0 \\ 0 & 0 & 0 & x & x & x & x \\ 0 & x & 0 & 0 & x & x & x \\ 0 & 0 & x & 0 & x & x & x \end{bmatrix}$$

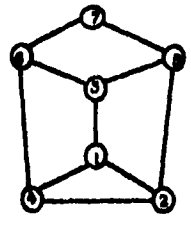
$G_2^A$ -



$\beta_2 = 4$

$$A_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} x & x & x & x & 0 & 0 & 0 \\ x & x & 0 & x & x & 0 & 0 \\ x & 0 & x & 0 & x & x & 0 \\ x & x & 0 & x & 0 & x & 0 \\ 0 & x & x & 0 & x & 0 & x \\ 0 & 0 & x & x & 0 & x & x \\ 0 & 0 & 0 & 0 & x & x & x \end{bmatrix}$$

$G_3^A$ -



$\beta_3 = 3$

Nótese que para una gráfica con n nodos, el número de total de permutaciones distintas es n!, de esta manera cuando n es grande, resulta costoso el cálculo del ancho de banda relativo, así como el total de permutaciones.

Esto se puede evitar teniendo criterios para numerar a la gráfica de tal forma que con la menor cantidad de permutaciones posibles se obtenga un ancho de banda óptimo. Estos criterios se discuten en los siguientes capítulos.

## 1.4 ALMACENAMIENTO

En general, la mayoría de los sistemas ralos que podemos encontrar en numerosos campos de la ciencia y la ingeniería, involucran un gran número de variables, por lo que es necesario el uso de una computadora para encontrar alguna solución. De esta manera nos interesa optimizar tanto el tiempo de ejecución como la memoria necesaria para almacenar los datos del problema.

Por lo que se han creado diversos métodos para almacenar, sólo a los elementos distintos de cero en arreglos unidimensionales, teniendo esto como consecuencia en muchas ocasiones, la reducción del tiempo de ejecución, ya que se reduce el número de operaciones elementales al resolverlos.

Este tipo de almacenamiento, generalmente se divide en dos partes, el *almacenamiento primario*, que contiene a los elementos distintos de cero, y el *almacenamiento secundario*, que contiene punteros, subíndices o cualquier otra información necesaria para reconocer la estructura de la matriz original.

Un ejemplo de esto es la siguiente matriz, la cual se almacenó por renglones, colocando en un arreglo primario a los elementos distintos de cero de cada renglón en orden de aparición, en el arreglo secundario se indica el número de la columna al que pertenece cada elemento y por último el arreglo de índices de tamaño n, sirve para distinguir el comienzo de cada renglón, esto se logra numerando a los elementos del arreglo primario desde 1...m, donde m representa el total de elementos de este arreglo.

Debido a que la matriz es simétrica, podemos almacenar solamente a los elementos que se encuentran debajo de la diagonal principal.

$$A=\begin{bmatrix} 2&0&0&8&0&2\\ 0&5&3&1&0&0\\ 0&3&6&0&2&4\\ 8&1&0&7&3&0\\ 0&0&2&3&9&0\\ 2&0&4&0&0&1 \end{bmatrix}$$

Arreglo primario: { 2,5,3,6,8,1,7,2,3,9,2,4,1 }
Arreglo secundario: { 1,2,2,3,1,2,4,3,4,5,1,3,6 }
Arreglo de índices: { 1,2,3,5,8,11 }

Otro tipo de almacenamiento se logra colocando en un arreglo unidimensional a los elementos distintos de cero, separados por identificadores de renglón o columna (-l) dependiendo del recorrido. Por ejemplo si almacenamos la matriz anterior por columnas, en la columna (-1) y renglón 1 está el elemento 2, en el renglón 4 está el 8, y el renglón 6 está el 2. Así para la columna 1 tenemos (-1 1 2 4 8 6 2). La matriz A sería:

A={ -1 1 2 4 8 6 2, -2 2 5 3 3 4 1, -3 3 6 5 2 6 4, -4 4 7 5 3, -5 5 9, -6 6 1}

El esquema de almacenamiento que utilizaremos en este trabajo, consiste en primer lugar, de un arreglo ADJNCY, que almacena a los elementos distintos de cero renglón por renglón, y un segundo arreglo XADJ, que apunta al principio de cada vecindad del arreglo anterior. Si almacenamos a la matriz anterior con este esquema resultan los arreglos[5]:

ADJNCY= ( 2,8,2 ; 5,3,1; 3,6,2,4; 8,1,7,3; 2,3,9; 2,4,1 )

XADJ = ( 1 4 7 11 15 18 22)

## 1.5 TEORÍA DE LA COMPLEJIDAD

En esta sección se presentan de manera breve, algunos aspectos de la teoría de la complejidad, ya que a lo largo de este trabajo se estudiarán diversas maneras de encontrar nodos periféricos, de grado mínimo, numeraciones para una gráfica y otras; cada una de las cuales involucra un determinado número de procesos u operaciones, que nos indican que

[5] Se han desarrollado muchos otros esquemas de almacenamiento, algunos de los más relevantes se pueden encontrar en Abrín V. [ 1 ], pp. 3-20., y Tewarson R. [ 22 ] pp. 2-10.

tan difíciles son de resolver. Por ejemplo, el encontrar para cada gráfica G y un entero positivo k, si una numeración de los nodos de G, genera un ancho de banda que no exceda a k. Sin embargo, el encontrar una numeración con estas características es un problema de combinatoria sumamente complejo, el cual se clasifica como NP-Completo (este concepto se define y discute más adelante).

Generalmente cada uno de estos problemas se plantea como un *algoritmo* , que podemos analizar y clasificar en relación al tiempo necesario para encontrar una solución. A continuación se presentan las características principales de un algoritmo y la manera de analizarlo.

### 1.5.1 Algoritmos

Existen varias definiciones de algoritmo, como "Un algoritmo, es un método especial para resolver cierta clase de problemas", o "Un algoritmo es una secuencia de pasos lógicos, concretos y finitos, que conducen a encontrar la solución a un problema específico".[6]

Sin embargo todo algoritmo debe cumplir con cinco características básicas:

1) Estar bien *definido*, esto es, cada paso debe definirse de modo preciso y claro.

2) Las operaciones que realice deben ser *efectivas*, por ejemplo sumar dos enteros, pero sumar dos número reales no, debido a que algunos valores pueden ser expresados con una cadena infinita de decimales.

3) Puede tener cero o más *entradas* o datos iniciales.

4) Puede tener una o más *salidas* o resultados.

5) Termina después de un número *finito* de operaciones.

Todas las características anteriores excepto la terminación, describen también a un *procedimiento*.

[6] Horowitz E. [ 17 ] pág. 1

Un ejemplo claro de un procedimiento, es el sistema operativo de una computadora, el cual controla todas las operaciones que se realizan, y cuando no hay tareas por hacer, no termina, si no que permanece en estado de espera hasta la siguiente instrucción.

Un *programa*, "es la expresión de un algoritmo en un lenguaje de programación"[7]. Algunas veces las palabras procedimiento o subrutina son usadas como sinónimos de programa.

El acto de crear un algoritmo, nunca será totalmente automatizado, sin embargo, se pueden diseñar expresiones de algoritmos claras y concisas mediante la programación estructurada.

Una vez que un algoritmo es desarrollado, será necesario corroborar que realiza correctamente las operaciones para todas las entradas posibles. Este proceso se conoce como validación, su objetivo es asegurar que el algoritmo funciona correctamente independientemente del lenguaje de programación en el que se escriba.

El análisis de algoritmos se refiere al proceso para determinar el tiempo de ejecución y la capacidad de almacenamiento que un algoritmo necesita.

El tiempo necesario para solucionar un problema, se mide de acuerdo al número de operaciones elementales (sumas, multiplicaciones, comparaciones, etc.) que realiza, así logramos que el tiempo de solución sea independiente de las características propias del equipo de cómputo con el que se cuente si consideramos que cada operación elemental se realiza en una unidad de tiempo.

Existen dos fases para analizar el tiempo de ejecución de un algoritmo. El análisis a priori y a posteriori. En el análisis a priori se obtiene una función, la cual delimita el tiempo de ejecución de un programa. Por otra parte en el análisis a posteriori, se obtienen las estadísticas del consumo de memoria y tiempo en su ejecución.

---

[7] Ibid., pág. 2

Por ejemplo, si tenemos la instrucción x← x+y, en algún lugar del programa y queremos determinar su tiempo total de ejecución, necesitamos conocer el número de veces que se va a ejecutar esta instrucción y el tiempo que se tarda en realizarla una vez.

De esta manera si tomamos tres segmentos distintos del programa para la misma instrucción:

```
x = x + y

Para i = 1 hasta n
  x(i) = x + y
siguiente i

Para i = 1 hasta n
  Para j = 1 hasta n
    x(i) = x(j) + y
  siguiente j
siguiente i
```

a) b) c)

En el caso a) se realiza 1 operación, en b) n y en c) $n^2$. Estos resultados se conocen como la magnitud o frecuencia del problema. Pero si tenemos tres algoritmos para resolver el mismo problema y sus magnitudes son n, $n^2$ y $n^3$, naturalmente elegiríamos la primera debido a que la segunda y tercera son progresivamente más lentas, así para n=10 cada uno de estos algoritmos necesitarían 10, 100, 1000 unidades de tiempo para resolverse respectivamente.

### 1.5.2 Complejidad.

La teoría de complejidad estudia la dificultad de resolver un problema en base al número de operaciones que realiza. Si logramos representar éste número mediante la función f(n), podemos definir el orden de la magnitud de ejecución de dicho problema como:

- $f(n) = O(g(n))$, si existen dos constantes positivas c y $n_0$, de tal forma que:

$$|f(n)| \leq c|g(n)| \quad \forall n \geq n_0$$

Por lo que decimos que un algoritmo se resuelve en tiempo O(g(n)), si el tiempo resultante es menor o igual que $|g(n)|$, para cualquier valor de n.

Por ejemplo, el algoritmo que calcula un árbol de peso mínimo, requiere un tiempo de ejecución para una gráfica con n arcos de O(nlogn). Para ejemplificar esto, tomamos dos

algoritmos para resolver un mismo problema con magnitudes $n^2$ y nlogn, para n=1024 requieren 1,048,576 contra 10,240 operaciones.

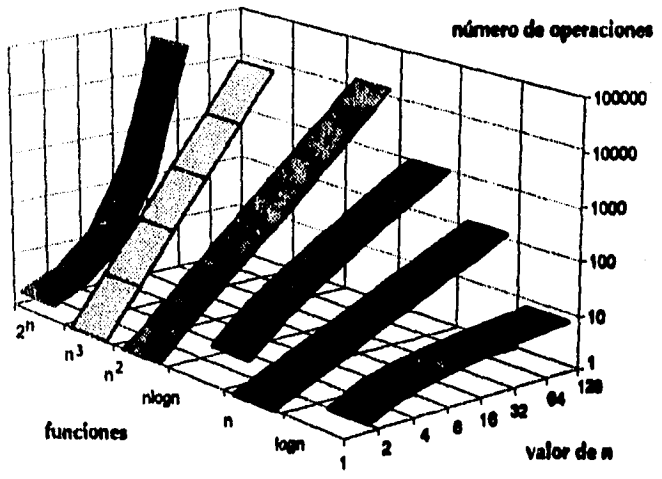
Si cada operación toma un microsegundo, el tiempo de ejecución sería de 1.05 y 0.01 segundos respectivamente. Ahora para n=2048 el primer algoritmo tomaría 4,194,304 contra 22,528 operaciones, es decir 4.2 y 0.02 segundos respectivamente. Por lo que si n se duplica, en $O(n^2)$ el algoritmo toma cuatro veces más tiempo en completarse mientras el segundo apenas se duplica.

Los tiempos de ejecución más comunes son:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

O(1), significa que el número de ejecuciones de operaciones elementales es fija, y el tiempo total está limitado por una constante. Las primeras seis magnitudes tienen una característica en común, están acotadas por un polinomio. La última se dice que requiere tiempo exponencial para su solución.

En la siguiente gráfica se observan los tiempos de ejecución para varios valores de n.



Así, podemos clasificar el tiempo de solución de los algoritmos en dos grupos, los que se solucionan en tiempo polinomial y los de tiempo no polinomial.

Un ejemplo de un problema que se soluciona en tiempo polinomial P, es el de encontrar una trayectoria de peso mínimo en una gráfica con m nodos y datos no negativos; el algoritmo de Dijkstra necesita $O(m^2)$ operaciones elementales, este número es independiente de los valores numéricos que representan el peso de los arcos.

Los problemas que se resuelven en tiempo no polinomial, generalmente son de tiempo exponencial, por ejemplo cuando se calcula la numeración de $2^k$, por medio de vectores k-dimensionales de 0-1. Esta función f(k), no esta acotada por ninguna función polinomial en k, y no requiere valores muy grandes de k, para exceder a las funciones polinomiales. El tiempo exponencial, puede ocurrir cuando el número de operaciones está en función al tamaño de los valores de entrada.

Tradicionalmente, el tamaño de un problema de optimización, se definía como el número de variables y el número de constantes que tenía. Sin embargo, existen algoritmos en donde el número de pasos para su solución, depende explícitamente de la magnitud de los datos. Así, el tamaño de un problema depende de la cantidad de información requerida para representarlo.

Algoritmos no determinísticos de tiempo polinomial y problemas NP.

Se conocen diversos algoritmos de programación lineal que se resuelven en tiempo polinomial, pero existen muchos casos especiales en problemas de optimización que necesitan estudiarse más a fondo para saber su tiempo de solución. A estos problemas se les denomina 'problemas de factibilidad '. Estos problemas denotados por la pareja X=(D,F) donde $F \subseteq D$, son conjuntos de cadenas binarias finitas. D se denomina el campo de alternativas de X, y F es el campo de alternativas factibles. Dada una alternativa $d \in D$, nos interesa determinar si $d \in F$.

En un algoritmo determinístico, el resultado de cada operación es único. Esta restricción se puede eliminar, y para alguna operación de un algoritmo podemos tener un conjunto limitado de soluciones, y se puede elegir cualquiera de estas salidas sujeto a la

condición para terminar el proceso. A este tipo de problemas se les conoce como no determinísticos.

Un algoritmo no determinístico consiste de dos fases. La primera fase es la de elección, aquí escogemos una cadena binaria $d \in D$, la cual pasa a la fase dos. Esta fase, llamada fase de verificación, es un algoritmo que trabaja con la cadena d, y si funciona correctamente genera un resultado y establece que $d \in F$.

Decimos que un algoritmo no determinístico es polinomial, si el tiempo necesario para la obtención de cada $d \in F$, es polinomial y depende únicamente de la longitud de la cadena d.

A esta clase de problemas de factibilidad, en donde cada resultado $d \in F$, se obtiene en tiempo polinomial, por medio de un algún algoritmo no determinístico, se les conoce como problemas NP.

Un ejemplo de lo anterior, es el problema de encontrar un ciclo Hamiltoniano en una gráfica G. Un ciclo Hamiltoniano es aquel que pasa por todos los vértices de la gráfica.

*Proposición 1* . El problema de encontrar un ciclo Hamiltoniano es NP.

*Dem*. Se da un algoritmo no determinístico de tiempo polinomial para el ciclo Hamiltoniano.

Entrada. Una gráfica $G=(V,A)$.

fase 1. Elegimos un subconjunto $A' \subseteq A$

fase 2. paso1. Si el grado de cada nodo de $G'=(V,A')$ es dos, ir al paso 2, si no salir.

paso2. Si $G'=(V,A')$ es conexa, entonces $G \in F$, si no regresar.

De esta manera, cada una de las fases se resuelve en tiempo polinomial.

La clase $C_0NP$.

Cada problema de factibilidad X=(D,F), tiene relacionado un problema $X^C=(D,F^C)$, conocido como el complemento de X. El problema es que no se sabe si los complementos de cualquier problema de factibilidad está en NP; sin embargo, estableceremos la terminología relacionada con los complementos de los problemas NP como $C_0NP = \{ X : X$ es un problema de factibilidad y $X^C \in NP \}$

*Proposición 2*. Si X es un problema de factibilidad y $X \in P$, entonces $X \in NP \cap C_0NP$.

*Dem.* Todo algoritmo de tiempo polinomial es también un algoritmo no determinístico de tiempo polinomial. Simplemente se ignora la fase de elección (fase 1), y se aplica el algoritmo de tiempo polinomial en la fase 2. Ya que $X \in P \Rightarrow X \in NP$. Pero si $X \in P$, también $X^C \in P$, y por lo anterior, si $X^C \in P$ implica que $X^C \in NP$. Si esto se cumple tenemos que $X \in C_0NP$.

Los problemas NP más difíciles.

Los problemas en NPC, conocidos como NP-completos, son un subconjunto de la clase NP, tal que si existe alguna $X \in NPC \cap P$, entonces cualquier problema NP está en P, esto es, NP = P.

Para encontrar este tipo de problemas, se recurre a una técnica que consiste en transformar polinomialmente un problema en otro, esta idea surge porque cuando nos encontramos un problema nuevo, tratamos de moldearlo como algún problema que sabemos resolver. Lo único que hay que añadir, es que la transformación se haga en tiempo polinomial.

Para ejemplificar esto, tomemos a $X_i = ( D_i , F_i )$, i=1,2, como dos problemas de factibilidad y sea la función g: $D_1 \Rightarrow D_2$, tal que para todo $d \in D_1$, $g(d) \in F_2$, si y solo si $d \in F_1$. Si esta función se calcula en tiempo polinomial, entonces decimos que $X_1$ es transformable polinomialmente en $X_2$.

De lo anterior podemos establecer que:

*Proposición 3*. Si $X_1$ es transformable polinomialmente en $X_2$, y $X_2 \in P$, entonces $X_1 \in P$.

Existe una técnica, llamada reducción polinomial , que es una forma más general que la transformación polinomial, para establecer cuando un problema puede ser resuelto en tiempo polinomial, si otro problema análogo si se puede. Así, tenemos que si $X_1$ es polinomialmente reducible a $X_2$ , si existe un algoritmo para $X_1$ , el cual utilice un algoritmo para $X_2$ como subrutina y se ejecute en tiempo polinomial, suponiendo que cada llamada a la subrutina toma una unidad de tiempo.

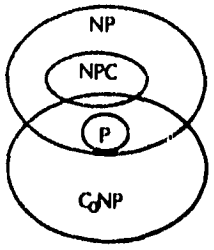
De esta manera, podemos establecer que un problema $X \in NP$, es NP-Completo, si todos los problemas en NP se pueden reducir polinomialmente a X. La existencia de estos problemas esta dada por la siguiente proposición.

*Proposición 4.* Si X es NP-completo, entonces P=NP si y solo si $X \in P$.

*Dem.* $X \in NP$ y P=NP, implica que $X \in P$, por otra parte si X es NP-completo y también es P, existe un algoritmo polinomial para cualquier problema en NP.

Una vez que tenemos un problema NP-Completo, podemos encontrar otros utilizando la reducción polinomial, esto es:

*Proposición 5.* Si $X_1$ es NP-Completo y $X_1$ es polinomialmente reducible en $X_2$ , y $X_2$ es NP, entonces $X_2$ es NP-Completo.



# CAPÍTULO II

# NODOS INICIALES

## 2.1 Nodos Periféricos.

*Def.* Un *nodo periférico*, es aquel cuya excentricidad es igual al diámetro de la gráfica, es decir, pertenece a la periferia.

Se han observado diversos ejemplos en los que al cimentar una estructura de nivel en un nodo de excentricidad máxima o periférico, se produce una profundidad en esta mayor. Como consecuencia de ello, se ha podido constatar la reducción del ancho de banda relativo a cada nivel. Este parámetro que claramente incide en la reducción del ancho de banda de la gráfica, es factor fundamental en los algoritmos que trabajan sobre la gráfica asociada a un sistema de ecuaciones.

Una de las desventajas de estos nodos, es que su localización resulta ser, en la mayoría de los casos, muy costosa debido a la cantidad de operaciones necesarias para encontrar todas las trayectorias distintas que unen a cada par de vértices en una gráfica.

Sin embargo existen algoritmos como el que se muestra a continuación[8], que encuentra nodos pseudoperiféricos, es decir, nodos cuya excentricidad se aproxima al diámetro de la gráfica. Estos nodos que claramente son más fáciles de encontrar, no resultan ser en la mayoría de los casos idóneos como iniciales.

**Algoritmo 2.1.1**

1. Elegir un nodo inicial $v \in V(G)$.
2. Generar la estructura de nivel cimentada en v.
3. Elegir un nodo v' , de grado mínimo del último nivel de la estructura.
4. Generar la estructura de nivel con v' como raíz.
   4.1 Si la e(v')>e(v), asignar v=v' y regresar a 3.
5. El nodo v' es pseudoperiférico.

Una manera de localizar nodos periféricos, es mediante el uso de las potencias de la matriz de adyacencia de una gráfica. Para esto, demostraremos las siguientes proposiciones:

[8] debido a Gibbs, Poole y Stockemeyer [14]

*Proposición 1.* Sea A la matriz de adyacencia de una gráfica G, donde $V(G)= \{v_1, v_2, ..., v_p\}$. Entonces la entrada aij de $A^{(n)}$, $n \geq 1$, indica el número de caminos distintos de longitud n, de $v_i$ a $v_j$.

*Dem.* Por inducción:

Para n=1. La matriz que se obtiene es la de adyacencia de G, es decir aij=1, implica que $v_i$ y $v_j$, están unidos por un camino de longitud 1. De otra forma, si aij=0 no existe un camino de $v_i$ a $v_j$ en G de longitud 1.

H.I.

Sea $A^{(n-1)}=\{aij^{(n-1)}\}$, la potencia n-1 de A y $n \geq 2$. Suponemos que la entrada aij de $A^{n-1}$ representa el número de caminos distintos de $v_i$ a $v_j$ de longitud n-1 en G.

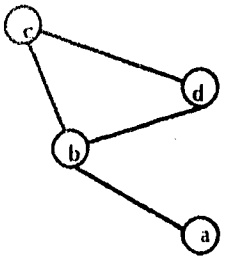
Consideremos ahora $A^{(n)}= \{aij^{(n)}\}$. Como $A^{(n)}= A^{(n-1)} \cdot A$, por la definición de producto de matrices:

$$a_{ij}^{(n)} = \sum_{k=1}^{n} a_{ik}^{(n-1)} a_{kj}$$

Por lo que cualquier camino de $v_i$ a $v_j$ de longitud n en G, consta de un camino de longitud n-1 y un camino de longitud 1. El primero de $v_i$ a $v_k$, y el segundo de $v_k$ a $v_j$. Así se concluye que $aij^{(n)}$, representa los caminos de longitud n de $v_i$ a $v_j$ en G.

Ejemplo.

G=




$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 3 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 0 & 3 & 1 & 1 \\ 3 & 2 & 4 & 4 \\ 1 & 4 & 2 & 3 \\ 1 & 4 & 3 & 2 \end{bmatrix}$$

Los caminos de longitud 3 de la entrada (1,2) en $A^3$ son:

Cam1= (a,b,c,b)

Cam2= (a,b,d,b)

Cam3=(a,b,a,b)

*Proposición 2.* Sea A la matriz de adyacencia de una gráfica con p vértices, la distancia del vértice i al j , es el mínimo entero n que satisface $aij^{(n)} > 0$ .

*Dem.* Por inducción.

Para n=1, la entrada aij=1 indica que hay un camino de longitud 1 entre el vértice i y el j (proposición 1). Como este camino es el mínimo o el primero que los une, podemos afirmar que la distancia del vértice i al j es uno para todo i,j=1,...,p.

H.I.

Sea n=k-1, suponemos que $aij^{k-1} = 0$ , y todas sus entradas correspondientes en las anteriores iteraciones son cero.

Sea n=k. Si la entrada $aij^k = s$, s>0 implica que existen s caminos de longitud k del vértice i al j, y por la hipótesis anterior, deducimos que estos son caminos mínimos entre el vértice i y el j. Así, podemos concluir que el mínimo entero n que satisface $aij^{(n)} > 0$, representa la distancia del vértice i al j.

A partir de la proposiciones anteriores se genera el siguiente algoritmo para encontrar nodos periféricos en base a las potencias de la matriz de adyacencias de una gráfica.

**Algoritmo 2.1.2**

1. Generar la matriz de adyacencias A, para una gráfica con p nodos.
2. En una matriz alterna D (matriz de distancias) de pxp, en cada entrada colocar:

$$d_{ij} = \begin{cases} 1, \text{ si } a_{ij}>0 \\ 0, \text{ en otro caso} \end{cases}$$

2.1 k=1

3. Sea k=k+1 , calcular $A^k$ , la matriz de potencias de A

4. Calcular para todo elemento de la matriz :

   4.1 Si $a_{ij}^k>0$ y $d_{ij} = 0$ y $i \neq j$
   hacer $d_{ij}=k$

   4.2 Actualizar la matriz D

5 . i) Si la matriz D, no se modificó o k=n-1 ir a 6
   ii) si no ir a 3.

6. (Fin) De la matriz D , obtener el máximo dij, los nodos i,j son periféricos y su excentricidad es k.

El número total de operaciones elementales necesarias para obtener la matriz $A^n$ de nxn, se puede calcular de la siguiente manera:

1. Para obtener el elemento $a_{ij}^2$ , se necesitan n productos y n-1 sumas:

$$a_{ij}^2 = \sum_{k=1}^{n} a_{ik}a_{kj} ,$$

y debido a que la matriz $A^2$, tiene $n^2$ elementos, el número de operaciones para obtenerla es $n^2( 2n-1)$. De aquí que para el cálculo de $A^n$ se realizan n-1 iteraciones. Por lo que el total de operaciones elementales está dado por:

$$n-1( n^2(2n-1))=2n^4-3n^3-n^2$$

por lo que este algoritmo se realiza en el peor de los casos en $O(n^4)$ . Sin embargo, en la mayoría de las ocasiones se obtiene la matriz de distancias con un número relativamente pequeño de iteraciones.

Otro método para localizar nodos periféricos, se describe a continuación.

**Algoritmo 2.1.3**

1. Generar la matriz de adyacencias de una gráfica G.

2. En una matriz de distancias D, asociar a cada entrada :

$$d_{ij} = \begin{cases} 1, \text{ si el nodo i es adyacente al j} \\ 0, \text{ en otro caso} \end{cases}$$

2.1 Hacer k=0

3. k=k+1

3.1 Recorrer por renglones la matriz de la sig. manera:

```
i=1,n
    j=1,n
      Si dij=1 recorrer la columna j :
        l=1,n
       Si dij=k y dil=0 e i≠j
           hacer dil=k+1
             l=l+1
       fin si
     j=j+1
      fin si
 i=i+1
```

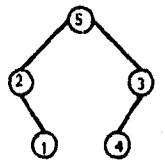
4. Si todo elemento dij (i≠j) es mayor a cero ir a 5, en otro caso ir a 3.

5. (FIN) Cada entrada de la matriz D, indica la distancia entre ese par de nodos

5.1 La entrada dij mayor, es la excentricidad máxima de la gráfica asociada, por lo que los nodos periféricos son esos i,j.

Ejemplo.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad G^{\wedge} =$$




En la matriz $D_1$ , se recorre el primer renglón y como el elemento d(1,2)=1, se recorre la columna 2 para encontrar los nodos adyacentes al nodo 2. En este caso d(2,5)=1, es decir, podemos ir del nodo 1 al 2 en un paso y del nodo 2 al 5 en otro paso, esto es, podemos ir del nodo 1 al 5 en dos pasos, de esta manera etiquetamos la entrada d(1,5)=2 .

Solamente podemos actualizaremos a las entradas no etiquetadas y que estén fuera de la diagonal. Se recorre todo el renglón 1 y como no hay más nodos adyacentes, cambiamos al renglón 2 y repetimos el proceso. Esto se hace para todos los renglones de la matriz.

$D_1 =$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 | 0 | 0 |

En la matriz $D_2$ ya actualizada, repetimos el recorrido tomando a los nodos cuya distancia es 1, y al recorrer la columna tomamos a las entradas etiquetadas con el 2. En este caso vamos del nodo 1 al 2 en un paso, y del nodo 2 al 3 en dos pasos, por lo que la distancia d(1,3)=3. Al cambiar de renglón, vamos del nodo 2, al 5 en un paso y del nodo 5 al 4 en dos pasos, así d(2,4)=3.

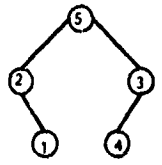
$D_2 =$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 2 |
| 2 | 1 | 0 | 2 | 0 | 1 |
| 3 | 0 | 2 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 2 |
| 5 | 2 | 1 | 1 | 2 | 0 |

Ejemplo.

$$A=\begin{bmatrix}0&1&0&0&0\\1&0&0&0&1\\0&0&0&1&1\\0&0&1&0&0\\0&1&1&0&0\end{bmatrix}$$

$G^{\wedge}=$




En la matriz $D_1$ , se recorre el primer renglón y como el elemento d(1,2)=1, se recorre la columna 2 para encontrar los nodos adyacentes al nodo 2. En este caso d(2,5)=1, es decir, podemos ir del nodo 1 al 2 en un paso y del nodo 2 al 5 en otro paso, esto es, podemos ir del nodo 1 al 5 en dos pasos, de esta manera etiquetamos la entrada d(1,5)=2 .

Solamente podemos actualizaremos a las entradas no etiquetadas y que estén fuera de la diagonal. Se recorre todo el renglón 1 y como no hay más nodos adyacentes, cambiamos al renglón 2 y repetimos el proceso. Esto se hace para todos los renglones de la matriz.

$D_1 =$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 | 0 | 0 |

En la matriz $D_2$ ya actualizada, repetimos el recorrido tomando a los nodos cuya distancia es 1, y al recorrer la columna tomamos a las entradas etiquetadas con el 2. En este caso vamos del nodo 1 al 2 en un paso, y del nodo 2 al 3 en dos pasos, por lo que la distancia d(1,3)=3. Al cambiar de renglón, vamos del nodo 2, al 5 en un paso y del nodo 5 al 4 en dos pasos, así d(2,4)=3.

$D_2=$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 2 |
| 2 | 1 | 0 | 2 | 0 | 1 |
| 3 | 0 | 2 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 2 |
| 5 | 2 | 1 | 1 | 2 | 0 |

Por último, la matriz $D_3$, se recorre en busca de los nodos a distancia 3, y se actualizan las entradas etiquetadas con 0 y fuera de la diagonal.

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 3 & 0 & 2 \\ 1 & 0 & 2 & 3 & 1 \\ 3 & 2 & 0 & 1 & 1 \\ 0 & 3 & 1 & 0 & 2 \\ 2 & 1 & 1 & 2 & 0 \end{bmatrix} \end{array}$$

Por lo tanto se obtiene la siguiente matriz, que indica las distancias entre todo par de vértices, de estas tomamos la mayor d(4,1)=4, por lo que los nodos periféricos son el 1 y el 4.

$$D = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ 3 & 2 & 0 & & \\ 4 & 3 & 1 & 0 & \\ 2 & 1 & 1 & 2 & 0 \end{bmatrix}$$

Otro algoritmo que puede ser utilizado para encontrar nodos pseudoperiféricos, es el algoritmo de Dijkstra. Generalmente, se utiliza para encontrar la arborescencia de rutas más cortas en una gráfica dirigida y con peso en los arcos.

Sin embargo se pueden modificar estas condiciones y aplicarlo en una gráfica no dirigida y suponer que el peso de cada arista es uno.

**Algoritmo de Dijkstra**

1. Tomar un nodo inicial s y etiquetarlo permanentemente con d(s)=0. Para todo nodo $x \neq s$, etiquetarlos temporalmente con $d(x)=\infty$. Sean a(x)=x, las etiquetas que indicarán el predecesor de x en la arborescencia. Sea p=s.

2. Para todo x, que sea adyacente a p y tenga etiqueta temporal, actualizarlo con:

$$d(x) = \min \{ d(x) , d(p) + d(p,x) \}$$

Si d(x) se modificó, hacer a(x)=p. Sea x* tal que d(x*) = mín { d(x) | d(x) es temporal }. Si d(x*)=∞, terminar ya que no existe arborescencia alguna con nodo inicial s. En otro caso tomar a d(x*) como permanente. Sea p=x*.

3. i) Si sólo se desea la ruta más corta de s a t, si p=t terminar, d(p) es la longitud del camino más corto. Si p≠ t ir a (2).

ii) Si se desea la excentricidad de nodo s, cuando todos los vértices tengan etiquetas permanentes, tomar de estas la mayor ya que es la excentricidad de vértice inicial. En otro caso regresar a (2).

4. Regresar a (1) y elegir como inicial a otro nodo, repetir este proceso hasta que se calculen todas las excentricidades del conjunto de vértices. De estas tomar las mayores, estos nodos son periféricos.

Una manera de resolver el problema de encontrar la distancia entre todo par de vértices de una gráfica G=[V,A], es encontrar la arborescencia de trayectorias mínimas de raíz x para todo x ∈ V.

Sin embargo existen procedimientos más eficientes para calcular la distancia entra todo par de vértices, como el algoritmo desarrollado por R.W. Floyd en 1962, que se aplica en gráficas dirigidas con cualquier valor en sus arcos. Sin embargo podemos modificar este criterio y aplicarlo a gráficas no dirigidas, suponiendo que el peso de cada arista es 1.

En este algoritmo se da una numeración a los vértices de una gráfica G con n nodos, y se utiliza una matriz C de orden n, para calcular las trayectorias mínimas para cada par de vértices; al terminar de aplicar el algoritmo, la entrada cij de C es igual a la distancia del vértice i al j. Para nuestros propósitos, elegiríamos la distancia mayor y a los vértices que la generan como periféricos.

**Algoritmo de Floyd.**

1. Se etiqueta la matriz C de orden n, de la siguiente manera:

$$c_{ij} = \begin{cases} 0, \text{ si } i=j \\ 1, \text{ si el nodo } i \text{ es adyacente al } j \\ \omega, \text{ en otro caso} \end{cases}$$

1.1 Hacer k=0.

2. Sea k=k+1. Para todo $i \neq k$ tal que $c_{ik} \neq \infty$ y para todo $j \neq k$ tal que $c_{kj} \neq \infty$ hacer:

$$c_{ij} = \min \{ c_{ij}, c_{ik} + c_{kj} \}$$

3. i) Si $c_{ij} \geq 0$, para toda i,j=1,..,n, y k=n ir a 4.

ii) Si $c_{ij} \geq 0$, para toda i,j=1,..,n, y k<n ir a 2.

4. Fin. La entrada $c_{ij}$ mayor, representa la distancia máxima entre los vértices i,j por lo tanto estos son periféricos.

**2.2 Nodos de grado mínimo.**

Estos nodos se caracterizan por tener una cantidad menor de vecinos que el resto de los vértices, por lo que si los elegimos como iniciales, garantizamos que la cardinalidad del siguiente nivel va a ser mínima. Este criterio nos llevaría a pensar en tomar a un nodo de grado mínimo como inicial, sin embargo se ha visto en la práctica que no siempre se obtiene, utilizando sólo este criterio, un ancho de banda óptimo.

Por lo que es conveniente el conjuntar ambos criterios (grado mínimo y excentricidad máxima), para obtener mejores resultados al generar la estructura de nivel.

Para encontrar a los nodos de grado mínimo, se puede utilizar la segunda potencia de la matriz de adyacencia de una gráfica, ya que los elementos de su diagonal , nos

indican el total de vecinos de cada nodo y de éstos tomaríamos los menores. Esta afirmación se basa en lo siguiente.

Debido a que en la matriz de adyacencia de una gráfica, los elementos distintos de cero del renglón i, son el total de vecinos del nodo i, podemos establecer lo siguiente:

$$Val_G(v_i) = \sum_{j=1}^{n} a_{ij};\ i=1,..,n. \qquad ..... (1)$$

Y de esto se desprende la siguiente proposición:

*Proposición 3.* El elemento $a_{ii}^{(2)}$ i=1,...,n , de una matriz de adyacencia de una gráfica G de orden n, indica el número de vecinos del vértice i.

*Dem.* Sabemos que el número de vecinos del vértice i, está dado por (1), y lo que obtenemos en la entrada $a_{ii}^{(2)}$ es:

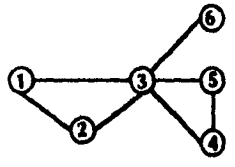
$$a_{ii}^{(2)} = \sum_{k=1}^{n} a_{ik}\, a_{ki}$$

Pero como la matriz es simétrica, se tiene que $a_{ik} = a_{ki}$, de donde:

$$\sum_{j=1}^{n} a_{ij} = \sum_{k=1}^{n} (a_{ik})^2$$

Ejemplo.

G =



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad \overset{2}{A} = \begin{bmatrix} 2 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 5 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Como se puede notar los elementos de la diagonal en $A^2$, indica el total de vecinos para cada nodo. Otra forma más fácil de calcular los grados de los nodos , es el formar listas de adyacencias para cada nodo, es decir, establecer en arreglos a los vecinos de cada vértice; o bien, simplemente sumar los renglones de la matriz de adyacencia .

# CAPÍTULO III

# NODOS NO INICIALES

En esta sección, se establecerán diversos algoritmos que permiten numerar a una gráfica con el principal objetivo de reducir el llenado al obtener el factor triangular L.

### 3.1 Algoritmo de Cuthill- Mckee ( CM )

El ordenamiento de Cuthill-Mckee (1969), fue diseñado para reducir el ancho de banda de una matriz rala y simétrica, utilizando el siguiente criterio. Sea a un nodo numerado, y b un vecino de a no numerado; para minimizar el ancho de banda del renglón asociado a b, éste nodo debe ser numerado lo más pronto posible después de a.

Es un algoritmo bastante rápido aunque el tiempo requerido para su ejecución es aproximadamente proporcional a n veces el grado promedio de los vértices de la gráfica, donde n es el total de vértices.

Descripción del Algoritmo.

1. Generar la estructura de nivel en cada vértice v, cuyo grado sea menor o igual que:

$$\max(\min\{(gr_{max} + gr_{min})/2 , gr_{promedio}-1\}, gr_{min})$$

2. A cada estructura de nivel obtenida en (1), numerarla nivel por nivel, con enteros consecutivos de manera que:

   2.1 Al vértice raíz se le asigna el 1

   2.2 Para los niveles sucesivos, numerar a los vértices no numerados en orden creciente a los grados, rompiendo los empates arbitrariamente, así hasta numerar todo el nivel actual. Se repite este proceso hasta numerar toda la gráfica.

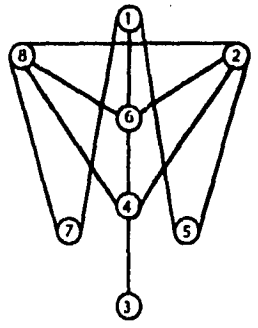
3. Para cada numeración producida en 2.2, se calcula su ancho de banda correspondiente.

4. Fin, se selecciona la numeración con ancho de banda menor.

Ejemplo.

De la siguiente gráfica, calculamos el grado máximo, mínimo y promedio, para encontrar los nodos iniciales de cada numeración.

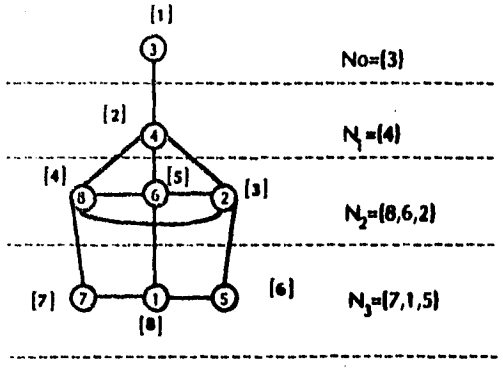
G=




$$A^G = \begin{bmatrix} x & 0 & 0 & 0 & x & 0 & x & 0 \\ 0 & x & 0 & x & x & x & 0 & x \\ 0 & 0 & x & x & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & x & 0 & x \\ x & x & 0 & 0 & x & 0 & 0 & 0 \\ x & x & 0 & x & 0 & x & 0 & x \\ x & 0 & 0 & 0 & 0 & 0 & x & x \\ 0 & x & 0 & x & 0 & x & x & x \end{bmatrix}$$

$gr_{max} = 4$ ; $gr_{min} = 1$ ; $gr_{promedio} = 3$

Por lo que generamos las estructuras de nivel en los nodos que tengan grado menor o igual a 2.




$\beta r_1 = 3$

fig. 3.1

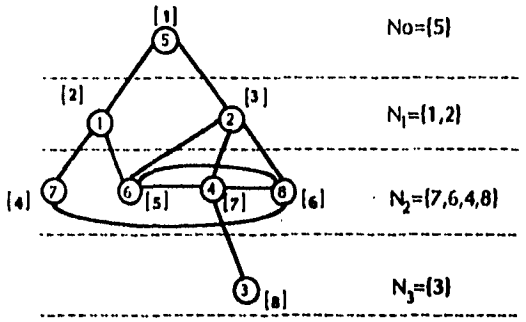



fig. 3.2

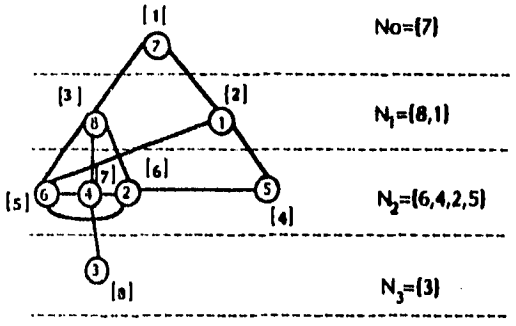



fig. 3.3

En este caso se toma como numeración ideal, la de la fig. 1 ya que su ancho de banda fue el menor. Comparando las entradas de la matriz original con la asociada a ésta numeración , se comprueba la reducción del ancho de banda en la matriz de 6 a 3.

$$A = \begin{bmatrix} x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 \\ 0 & x & x & x & x & x & 0 & 0 \\ 0 & x & x & x & x & 0 & x & 0 \\ 0 & x & x & x & x & 0 & 0 & x \\ 0 & 0 & x & 0 & 0 & x & 0 & x \\ 0 & 0 & 0 & x & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \end{bmatrix}$$

Matriz obtenida de la numeración de la figura 3.1

### 3.2 Algoritmo de Cuthill-Mckee inverso ( RCM )

Se ha comprobado que este algoritmo, no mejora el ancho de banda que se obtiene a partir del algoritmo anterior, pero se puede lograr una considerable reducción en el *perfil* de la matriz, éste último surge de la sigulente definición.

*Def.* La *envolvente* de una matriz A es el conjunto de elementos de cada renglón, que se toman desde el primer elemento distinto de cero, hasta uno antes de la diagonal principal, esto es:

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

Ejemplo.

A=

| | | | | | | |
|---|---|---|---|---|---|---|
| x | x | 0 | 0 | x | 0 | 0 |
| x | x | 0 | x | x | 0 | x |
| 0 | 0 | x | 0 | 0 | 0 | x |
| 0 | x | 0 | x | 0 | 0 | 0 |
| x | x | 0 | 0 | x | x | 0 |
| 0 | 0 | 0 | 0 | x | x | 0 |
| 0 | x | x | 0 | x | 0 | x |

Env(A)

| i | $\beta_i$ |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 3 | 0 |
| 4 | 2 |
| 5 | 4 |
| 6 | 1 |
| 7 | 5 |

*Def.* El *Perfil* o tamaño de la envolvente es la suma de los anchos de banda relativos $\beta_i$ de cada renglón, es decir :

$$| Env(A) | = \sum_{i=1}^{n} \beta_i (A)$$

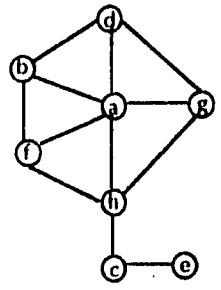
Así, el perfil para el ejemplo anterior es 13.

Descripción de Algoritmo.

1. Aplicar el algoritmo de Cuthill-Mckee a una gráfica G.
2. A la numeración obtenida en (1), 1,2,3,...,N, asignarle la secuencia N,N-1,...,1.

Ejemplo.

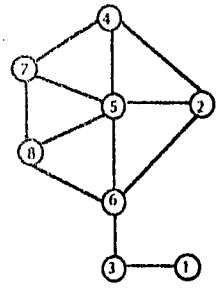
Sea G=

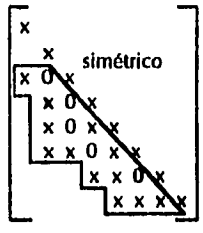



Aplicando el RCM, con nodo inicial f, tenemos:

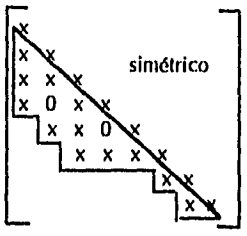
| Nodo | Numeración | Invertida |
|---|---|---|
| f | 1 | e |
| b | 2 | g |
| h | 3 | c |
| a | 4 | d |
| d | 5 | a |
| c | 6 | h |
| g | 7 | b |
| e | 8 | f |

Gráficamente, el ordenamiento RCM es:




Este ordenamiento genera un perfil de 17. Ahora si tomamos como inicial al nodo **e**, nos genera la siguiente matriz con un perfil de 14.




### 3.3 Algoritmo de Grado Mínimo

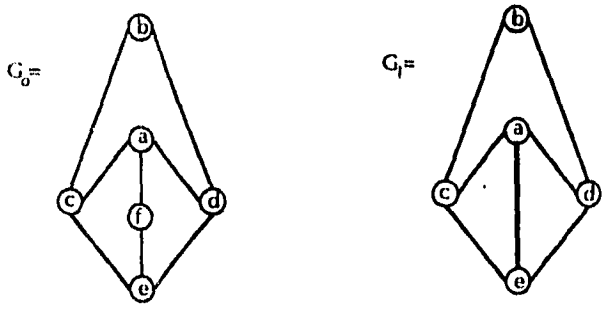
Este Algoritmo, diseñado para reducir el llenado al encontrar el factor L, provoca una numeración de los nodos de una gráfica G=(V,A) en base a sus gráficas de eliminación, cada una de las cuales se genera al tomar, como nodo a eliminar, a uno de grado mínimo.

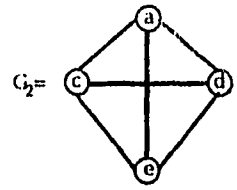
Descripción del Algoritmo.

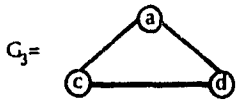
1. i=1
2. De la Gráfica de Eliminación $G_{i-1} = ( V_{i-1} , E_{i-1} )$, elegir un nodo de grado mínimo en $G_{i-1}$.
3. Obtener la nueva Gráfica $G_i=( V_i ,A_i )$, eliminando el nodo $x_i$ de la gráfica $G_{i-1}$ .
4. i=i+1. Si $i > |V|$ salir, si no ir a 2.

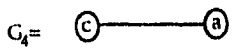
Pueden haber varios nodos de grado mínimo en alguna gráfica $G_i$, estos empates se rompen arbitrariamente, aunque se pueden establecer criterios como los mostrados en el capítulo anterior, que provocan diferentes versiones para este algoritmo.

Ejemplo.







| Secuencia de la Eliminación | | |
|---|---|---|
| $G_i$ | Nodo Eliminado | Grado |
| 0 | f | 2 |
| 1 | b | 2 |
| 2 | e | 3 |
| 3 | d | 2 |
| 4 | a | 1 |
| 5 | c | 0 |

### 3.4 Algoritmo de Disección Anidada

Al igual que el algoritmo de Grado mínimo, este algoritmo intenta minimizar el llenado al encontrar el factor triangular L. Este algoritmo necesita menos almacenamiento para su ejecución y resulta más rápido. En esta sección introducimos el concepto de Separadores de una gráfica.

*Def.* Sea A una matriz simétrica y G^ su gráfica asociada. Un conjunto de vértices de G^ es un separador S, si divide a la gráfica en dos subconjuntos de vértices $C_1$ y $C_2$ , más o menos iguales. Fig. 3.4

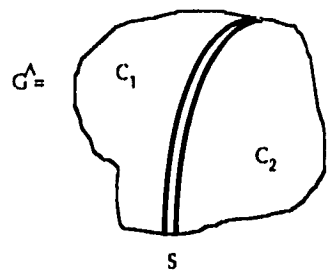



fig. 3.4

Si los nodos de S son numerados después que los vértices de $C_1$ y $C_2$, se induce una partición en el orden de la matriz asociada como se muestra a continuación.

$$A = \begin{bmatrix} A_1 & 0 & V_1 \\ 0 & A_2 & V_2 \\ V_1^T & V_2^T & A_S \end{bmatrix}$$

Esta partición tiene dos características básicas. La primera, es que los bloques de elementos nulos no se alteran durante el proceso de factorización, y la segunda es que se pueden factorizar las matrices $A_1$ y $A_2$ independientemente, esto nos lleva a pensar en una factorización en paralelo de ambas matrices en distintos procesadores, lo cual se traduce en una forma más rápida de encontrar dichos factores.

Este procedimiento es aplicable tanto a mallas regulares o matrices, como a sus gráficas asociadas. Analizaremos primero la disección en matrices, así como la manera de numerarlas. Para ejemplificar lo anterior, tenemos el siguiente algoritmo:

**Algoritmo 3.4.1**

1. i=0. Sea A, una matriz de orden n.

2. Se elige un conjunto de elementos de A (una columna o renglón), de manera que ésta se divida en dos submatrices $SM^i$ y $SM^{i+1}$ , más o menos iguales, a este conjunto llamarlo el separador $S^i$ .

2.1 Si cada submatriz $SM^i$ obtenida en 2 se puede dividir hacer i=i+1, e ir a 2.
2.2 Si no ir a 3.

3. A cada pareja de submatrices $SM^i$, $SM^{i+1}$ , numerarlas secuencialmente dejando al final al conjunto $S^i$ , que las separa.

4. Si quedan submatrices o separadores sin numerar:

4.1 Hacer i=i-1 e ir a 3.

5. Salir

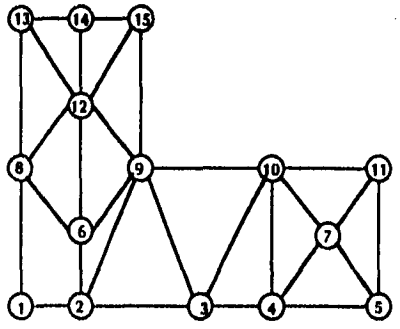
Ejemplo. Sea A una matriz de orden 7, aplicando el algoritmo anterior se obtiene la siguiente disección anidada.

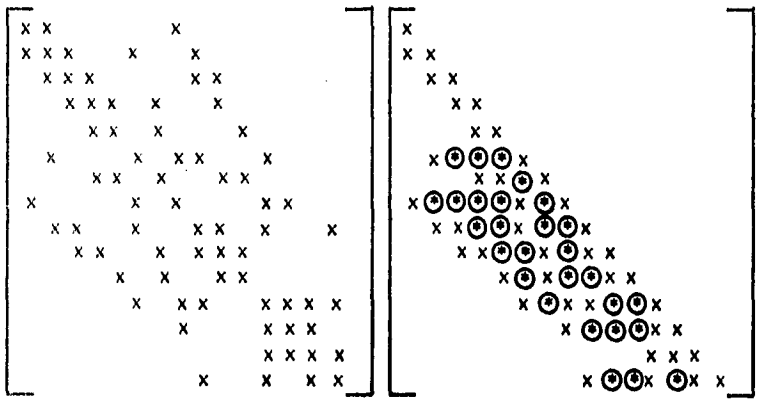
$$A = \begin{bmatrix} 35 & 39 & 32 & 49 & 14 & 18 & 11 \\ 36 & 38 & 33 & 48 & 15 & 17 & 12 \\ 34 & 37 & 31 & 47 & 13 & 16 & 10 \\ 42 & 41 & 40 & 46 & 21 & 20 & 19 \\ 26 & 30 & 23 & 45 & 5 & 9 & 2 \\ 27 & 29 & 24 & 44 & 6 & 8 & 3 \\ 25 & 28 & 22 & 43 & 4 & 7 & 1 \end{bmatrix}$$

El procedimiento de disección para una gráfica, es análogo al anterior, sólo que los separadores son subconjuntos de vértices, que al eliminarlos, divide a la gráfica original en dos componentes más o menos iguales.

Este procedimiento se sigue hasta que ya no podamos dividir a ninguna de las subgráficas, en este momento se procede a dar una numeración secuencial a cada subconjunto de vértices y al final a los separadores.

Ejemplo. En la siguiente figura se muestra una gráfica, y sus correspondientes matrices A y el factor de Cholesky L, el llenado se denota por ⊙.



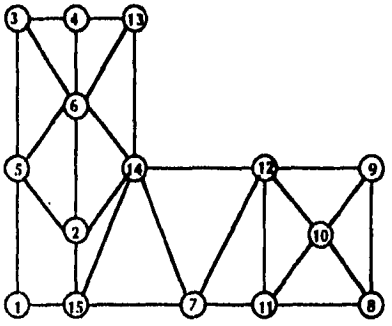



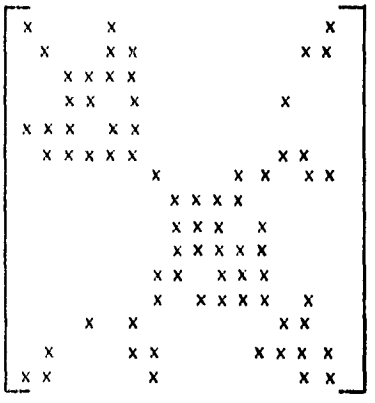
A L

Ahora, si elegimos como separadores a los conjuntos de nodos:

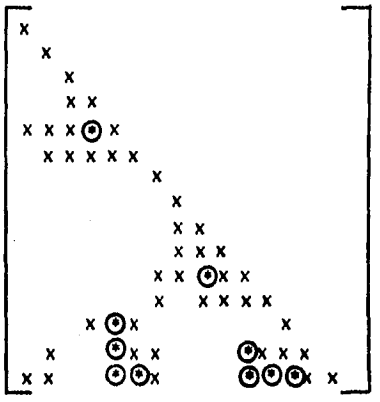
$S^0 = \{ 2,9,15 \}$
$S^1 = \{ 4,10 \}$
$S^2 = \{ 8,12 \}$

y numeramos de acuerdo al algoritmo, se generan las siguientes matrices A y L, en donde esta última tiene un llenado mucho menor.



A



L

La efectividad de éste algoritmo, depende en gran medida tanto de la cardinalidad de los conjuntos separadores, así como del tamaño relativo de cada pareja de subgráficas que se obtienen al efectuar una separación.

De esta manera, se obtienen mejores resultados si elegimos conjuntos separadores mínimos, que permitan obtener subgráficas del mismo tamaño.

Una manera de obtener conjuntos separadores, es el de generar una estructura de nivel cimentada en un nodo pseudoperiférico, como se describió en el Capítulo II, y elegir como separador a algún subconjunto de vértices de un nivel intermedio. Este procedimiento se describe con más detalle en el siguiente algoritmo.

**Algoritmo 3.4.2**

Sea G= (V,A) una gráfica.

1. Hacer R= V y M = |V|

2. Encontrar una componente conexa G(C) en G(R) y cimentar la estructura de nivel en un nodo pseudoperiférico v:

$$N(v)= \{ N_0, N_1, \ldots, N_r \}$$

3. Si r ≤2.

   3.1 Hacer S = C e ir a 4.
   3.2 Si no, sea j = [ ( r+1) / 2 ], y determinar el conjunto $S \subset N_j$ donde:

$$S = \{ y \in N_j \mid Ady(y) \cap N_{j+1} \neq \emptyset \}$$

4. Numerar los nodos en el separador S desde ( M - |S| + 1 ) hasta M .

5. Sean ahora R = R - S y M = M - |S|

   5.1 Si R ≠ ∅, ir a 2
   5.2. En otro caso salir.

En el paso 3, el separador S se puede obtener descartando los nodos en $N_j$ los cuales no son adyacentes a ningún nodo de $N_{j+1}$. Esto en muchos casos, reduce el tamaño de los separadores.

# CASO EJEMPLO

En esta sección, se presentan algunos ejemplos que muestran las diferencias obtenidas en el ancho de banda al aplicar los distintos métodos para la obtención de nodos iniciales, así como los distintos perfiles que se obtienen con los métodos de reordenamiento.

Se generaron varias corridas del programa modificando en cada una de ellas la dimensión de la matriz generada, así como también la raleza del sistema tomando como parámetros, matrices en donde el total de elementos distintos de cero se encuentran entre el 5 y el 50%.

Se presentan también tres gráficas:

La primera compara el ancho de banda que se obtiene con los distintos métodos para la obtención de nodos iniciales ( Original, Matpot, Trayec, Pseudo y Gmin ), variando la cantidad de elementos distintos de cero en forma descendente de izquierda a derecha, para cada dimensión de la matriz.

La segunda compara los perfiles que se obtienen al aplicar los métodos de reordenamiento CM y RCM, tomando como en la primera gráfica distintos nodos iniciales.

Por último, se presenta una matriz de dimensión 75, en la cual se modificó el número de elementos distintos de cero y se comparan los anchos de banda para cada nodo inicial.

| DIM=25 | | | |
|---|---|---|---|
| TOTAL 625 | | CM | RCM |
| no ceros | 306 | β perfil | perfil |
| ORIGINAL | | 24 291 | 291 |
| MATPOT | | 20 278 | 269 |
| TRAYEC | | 21 291 | 273 |
| PSEUDO | | 19 260 | 244 |
| GMIN | | 18 257 | 242 |
| no ceros | 228 | | |
| ORIGINAL | | 22 256 | 256 |
| MATPOT | | 17 241 | 222 |
| TRAYEC | | 17 237 | 219 |
| PSEUDO | | 17 237 | 219 |
| GMIN | | 17 241 | 222 |
| no ceros | 182 | | |
| ORIGINAL | | 22 262 | 262 |
| MATPOT | | 14 205 | 181 |
| TRAYEC | | 13 195 | 176 |
| PSEUDO | | 13 195 | 176 |
| GMIN | | 14 205 | 181 |
| no ceros | 136 | | |
| ORIGINAL | | 23 237 | 237 |
| MATPOT | | 13 201 | 182 |
| TRAYEC | | 11 184 | 168 |
| PSEUDO | | 14 206 | 179 |
| GMIN | | 14 206 | 179 |
| no ceros | 74 | | |
| ORIGINAL | | 19 163 | 163 |
| MATPOT | | 10 125 | 78 |
| TRAYEC | | 9 115 | 82 |
| PSEUDO | | 9 115 | 82 |
| GMIN | | 10 125 | 78 |

| no.ceros | DISECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 49 | 24 | 276 | 24 | 342 |
| 36 | 24 | 261 | 23 | 308 |
| 29 | 21 | 253 | 24 | 314 |
| 22 | 24 | 240 | 23 | 264 |
| 12 | 22 | 186 | 22 | 192 |

| DIM=50 | | | | |
|---|---|---|---|---|
| TOTAL 2500 | | CM | | RCM |
| no ceros | 1230 | β | perfil | perfil |
| ORIGINAL | | 46 | 1160 | 1160 |
| MATPOT | | 44 | 1167 | 1110 |
| TRAYEC | | 43 | 1162 | 1137 |
| PSEUDO | | 42 | 1152 | 1120 |
| GMIN | | 43 | 1152 | 1125 |
| no ceros | 966 | | | |
| ORIGINAL | | 49 | 1162 | 1162 |
| MATPOT | | 40 | 1128 | 1064 |
| TRAYEC | | 39 | 1124 | 1063 |
| PSEUDO | | 41 | 1122 | 1059 |
| GMIN | | 41 | 1122 | 1059 |
| no ceros | 754 | | | |
| ORIGINAL | | 46 | 1151 | 1131 |
| MATPOT | | 38 | 1068 | 1014 |
| TRAYEC | | 39 | 1082 | 1012 |
| PSEUDO | | 37 | 1061 | 1003 |
| GMIN | | 40 | 1065 | 991 |
| no ceros | 494 | | | |
| ORIGINAL | | 49 | 1060 | 1060 |
| MATPOT | | 36 | 1021 | 897 |
| TRAYEC | | 34 | 994 | 866 |
| PSEUDO | | 33 | 1005 | 877 |
| GMIN | | 34 | 969 | 876 |
| no ceros | 242 | | | |
| ORIGINAL | | 46 | 693 | 693 |
| MATPOT | | 22 | 700 | 572 |
| TRAYEC | | 22 | 700 | 572 |
| PSEUDO | | 22 | 700 | 572 |
| GMIN | | 26 | 783 | 616 |

| no ceros | DISECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 49 | 49 | 1179 | 49 | 1238 |
| 39 | 48 | 1154 | 49 | 1412 |
| 30 | 45 | 1089 | 49 | 1262 |
| 20 | 46 | 1044 | 47 | 1142 |
| 10 | 46 | 877 | 46 | 918 |

| DIM=75 | | | |
|---|---|---|---|
| TOTAL 5625 | CM | | RCM |
| no ceros 2748 | β | perfil | perfil |
| ORIGINAL | 72 | 2700 | 2700 |
| MATPOT | 67 | 2683 | 2600 |
| TRAYEC | 66 | 2670 | 2618 |
| PSEUDO | 68 | 2666 | 2613 |
| GMIN | 67 | 2672 | 2617 |
| no ceros 2216 | | | |
| ORIGINAL | 73 | 2679 | 2679 |
| MATPOT | 65 | 2593 | 2519 |
| TRAYEC | 66 | 2645 | 2555 |
| PSEUDO | 64 | 2611 | 2507 |
| GMIN | 66 | 2623 | 2537 |
| no ceros 1690 | | | |
| ORIGINAL | 74 | 2618 | 2618 |
| MATPOT | 60 | 2520 | 2424 |
| TRAYEC | 63 | 2548 | 2405 |
| PSEUDO | 63 | 2566 | 2462 |
| GMIN | 62 | 2545 | 2447 |
| no ceros 1098 | | | |
| ORIGINAL | 72 | 2481 | 2481 |
| MATPOT | 53 | 2336 | 2153 |
| TRAYEC | 57 | 2360 | 2162 |
| PSEUDO | 56 | 2402 | 2189 |
| GMIN | 56 | 2406 | 2210 |
| no ceros 490 | | | |
| ORIGINAL | 69 | 1938 | 1938 |
| MATPOT | 46 | 1946 | 1596 |
| TRAYEC | 43 | 1910 | 1516 |
| PSEUDO | 47 | 2044 | 1563 |
| GMIN | 47 | 2044 | 1563 |

| no cems | DISECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 49 | 74 | 2717 | 74 | 3216 |
| 39 | 73 | 2669 | 73 | 3198 |
| 30 | 74 | 2620 | 73 | 3043 |
| 20 | 72 | 2470 | 73 | 2764 |
| 9 | 73 | 2158 | 73 | 2537 |

| DIM=100 | | | | |
|---|---|---|---|---|
| TOTAL 10000 | | CM | | RCM |
| no ceros | 4926 | β | perfil | perfil |
| ORIGINAL | | 99 | 4846 | 4846 |
| MATPOT | | 92 | 4842 | 4743 |
| TRAYEC | | 92 | 4821 | 4730 |
| PSEUDO | | 91 | 4784 | 4689 |
| GMIN | | 91 | 4784 | 4689 |
| no ceros | 4062 | | | |
| ORIGINAL | | 98 | 4813 | 4813 |
| MATPOT | | 90 | 4779 | 4638 |
| TRAYEC | | 90 | 4782 | 4679 |
| PSEUDO | | 90 | 4762 | 4625 |
| GMIN | | 90 | 4762 | 4625 |
| no ceros | 2960 | | | |
| ORIGINAL | | 99 | 4756 | 4756 |
| MATPOT | | 86 | 4610 | 4456 |
| TRAYEC | | 87 | 4655 | 4473 |
| PSEUDO | | 87 | 4659 | 4441 |
| GMIN | | 86 | 4610 | 4456 |
| no ceros | 1926 | | | |
| ORIGINAL | | 97 | 4540 | 4540 |
| MATPOT | | 82 | 4482 | 4214 |
| TRAYEC | | 82 | 4479 | 4188 |
| PSEUDO | | 81 | 7759 | 4160 |
| GMIN | | 81 | 4482 | 4213 |
| no ceros | 1008 | | | |
| ORIGINAL | | 94 | 4295 | 4295 |
| MATPOT | | 68 | 3907 | 3428 |
| TRAYEC | | 67 | 3964 | 3482 |
| PSEUDO | | 70 | 3960 | 3374 |
| GMIN | | 68 | 3907 | 3428 |

| no ceros | DISECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 49 | 99 | 4856 | 99 | 5597 |
| 41 | 99 | 4814 | 99 | 5267 |
| 30 | 98 | 4729 | 98 | 5461 |
| 19 | 98 | 4497 | 98 | 5279 |
| 10 | 93 | 4202 | 99 | 4811 |

| DIM=150 | | | | |
|---|---|---|---|---|
| TOTAL 22500 | | CM | | RCM |
| no ceros | 11046 | β | perfil | perfil |
| ORIGINAL | | 148 | 11020 | 11020 |
| TRAYEC | | 141 | 10981 | 10870 |
| PSEUDO | | 140 | 10934 | 10825 |
| GMIN | | 140 | 10934 | 10825 |
| no ceros | 8856 | | | |
| ORIGINAL | | 139 | 10967 | 10957 |
| TRAYEC | | 139 | 10906 | 10734 |
| PSEUDO | | 141 | 10897 | 10706 |
| GMIN | | 141 | 10897 | 10706 |
| no ceros | 6822 | | | |
| ORIGINAL | | 148 | 10940 | 10940 |
| TRAYEC | | 135 | 10734 | 10423 |
| PSEUDO | | 133 | 10699 | 10372 |
| GMIN | | 135 | 10737 | 10429 |
| no ceros | 4398 | | | |
| ORIGINAL | | 147 | 10965 | 10965 |
| TRAYEC | | 130 | 10430 | 9818 |
| PSEUDO | | 129 | 10367 | 9885 |
| GMIN | | 128 | 10341 | 9783 |
| no ceros | 2286 | | | |
| ORIGINAL | | 147 | 9967 | 9967 |
| TRAYEC | | 117 | 9786 | 8780 |
| PSEUDO | | 113 | 9670 | 8803 |
| GMIN | | 113 | 9670 | 8803 |

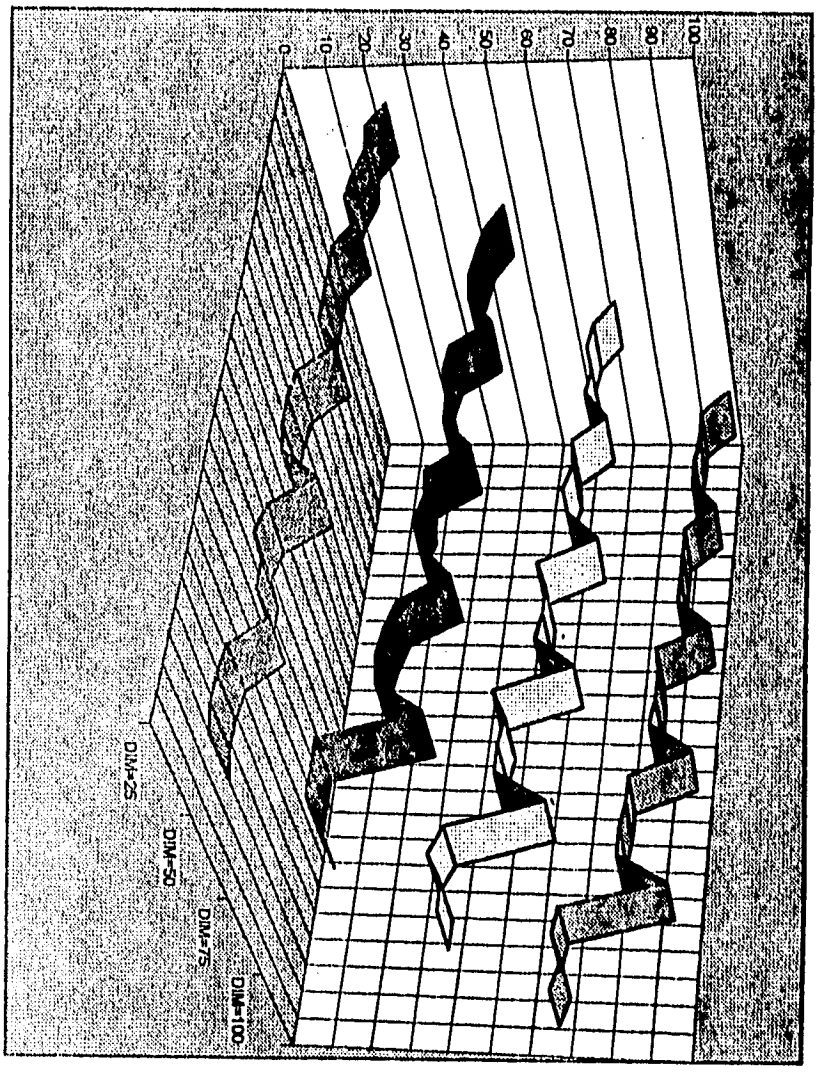
| no ceros | DISECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 49 | 149 | 11024 | 148 | 12548 |
| 39 | 149 | 10965 | 148 | 12537 |
| 30 | 148 | 10814 | 147 | 11667 |
| 20 | 149 | 10535 | 149 | 12248 |
| 10 | 147 | 9801 | 144 | 10460 |

| DIM=200 | | | | |
|---|---|---|---|---|
| TOTAL 40000 | | CM | | RCM |
| no ceros | 7902 | β | perfil | perfil |
| ORIGINAL | | 198 | 19124 | 19124 |
| TRAYEC | | 178 | 18768 | 18103 |
| PSEUDO | | 173 | 18791 | 17993 |
| GMIN | | 174 | 18824 | 18200 |
| no ceros | 4098 | | | |
| ORIGINAL | | 198 | 18066 | 18066 |
| TRAYEC | | 165 | 17817 | 16583 |
| PSEUDO | | 164 | 17954 | 16498 |
| GMIN | | 160 | 17795 | 16516 |

| no ceros | DIRECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 20 | 198 | 19061 | 199 | 21209 |
| 10 | 199 | 18041 | 197 | 20131 |

| DIM=250 | | | | |
|---|---|---|---|---|
| TOTAL 62500 | | CM | | RCM |
| no ceros | 6454 | β | perfil | perfil |
| ORIGINAL | | 249 | 26001 | 26001 |
| TRAYEC | | 216 | 26918 | 27353 |
| PSEUDO | | 206 | 26465 | 26573 |
| GMIN | | 214 | 26733 | 26906 |
| no ceros | 3076 | | | |
| ORIGINAL | | 244 | 26458 | 26458 |
| TRAYEC | | 183 | 26009 | 22895 |
| PSEUDO | | 184 | 26316 | 23048 |
| GMIN | | 181 | 25982 | 22792 |

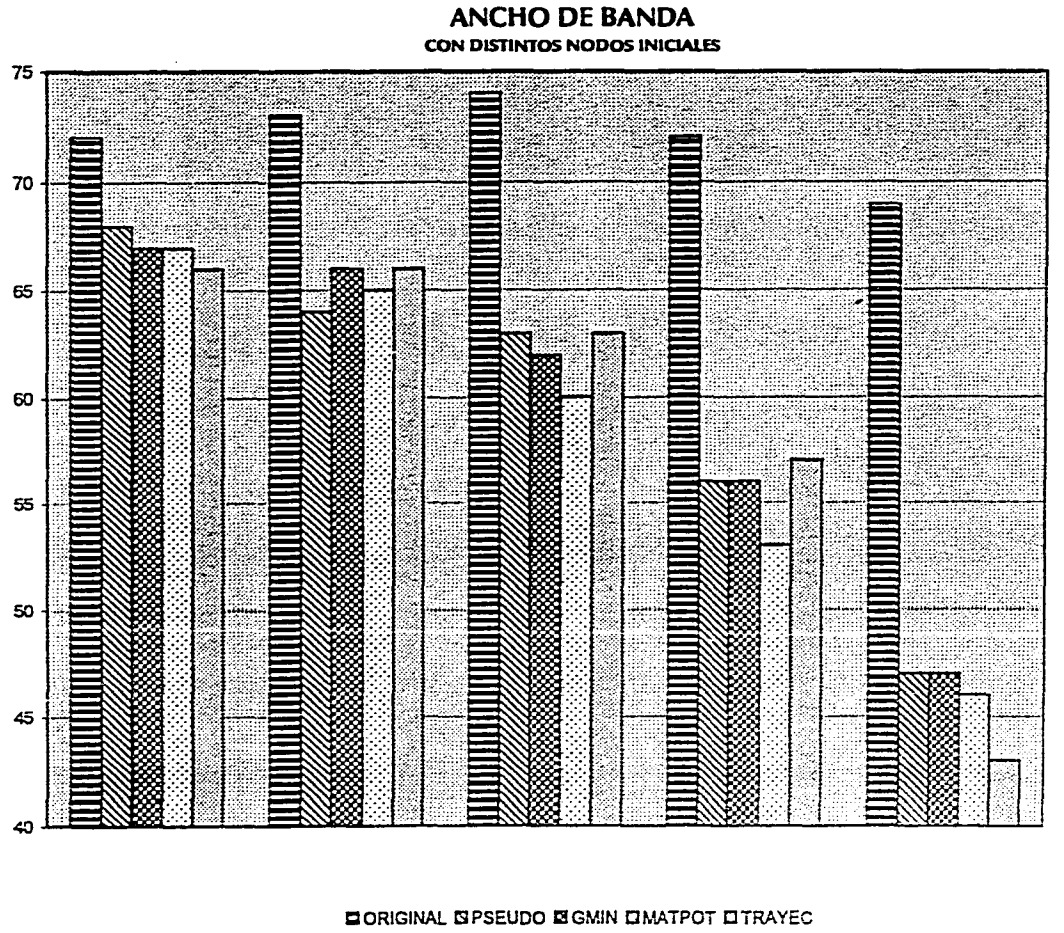
| no ceros | DIRECCION | | GRADO MINIMO | |
|---|---|---|---|---|
| % | β | perfil | β | perfil |
| 10 | 246 | 26936 | 247 | 32864 |
| 5 | 244 | 26886 | 244 | 30001 |




ANCHO DE BANDA

PERFIL
DIM = 150

CRITERIOS PARA LA SELECCION DE NODOS


ANCHO DE BANDA
CON DISTINTOS NODOS INICIALES
75
70
65
60
55
50
45
40
ORIGINAL PSEUDO GMIN MATPOT TRAYEC

# CONCLUSIONES

La reducción en el ancho de banda que se obtiene al elegir como nodo inicial a un nodo periférico, resulta en la mayoría de los casos la óptima. Sin embargo hay gráficas para las cuales no solo se necesita que el nodo inicial sea periférico, sino que también sea de grado mínimo, esto se puede apreciar, ya que en algunos ejemplos, resultó mejor la reducción en el ancho de banda obtenido con un nodo de grado mínimo o uno pseudoperiférico. Esto se debe a que el algoritmo de nodos pseudoperiféricos conjunta ambos criterios, nodos de grado mínimo y excentricidad máxima, para la elección del nodo inicial, sin embargo se puede mejorar si se toma como inicial un nodo periférico y de grado mínimo.

El problema ahora por resolver, es el de establecer criterios para romper los empates, ya que normalmente se presentan varios nodos con estas características. Creo que sería conveniente el utilizar análisis estadístico para alguna muestra de matrices generadas aleatoriamente, que contengan una gran cantidad de estos nodos.

En cuanto a los algoritmos de reordenamiento enfocados a reducir el llenado, podemos notar que no mejoran en ningún caso el perfil o el ancho de banda, en cambio el llenado que se obtiene por el Algoritmo de Disección Anidada es menor que el obtenido por el Algoritmo de Grado Mínimo.

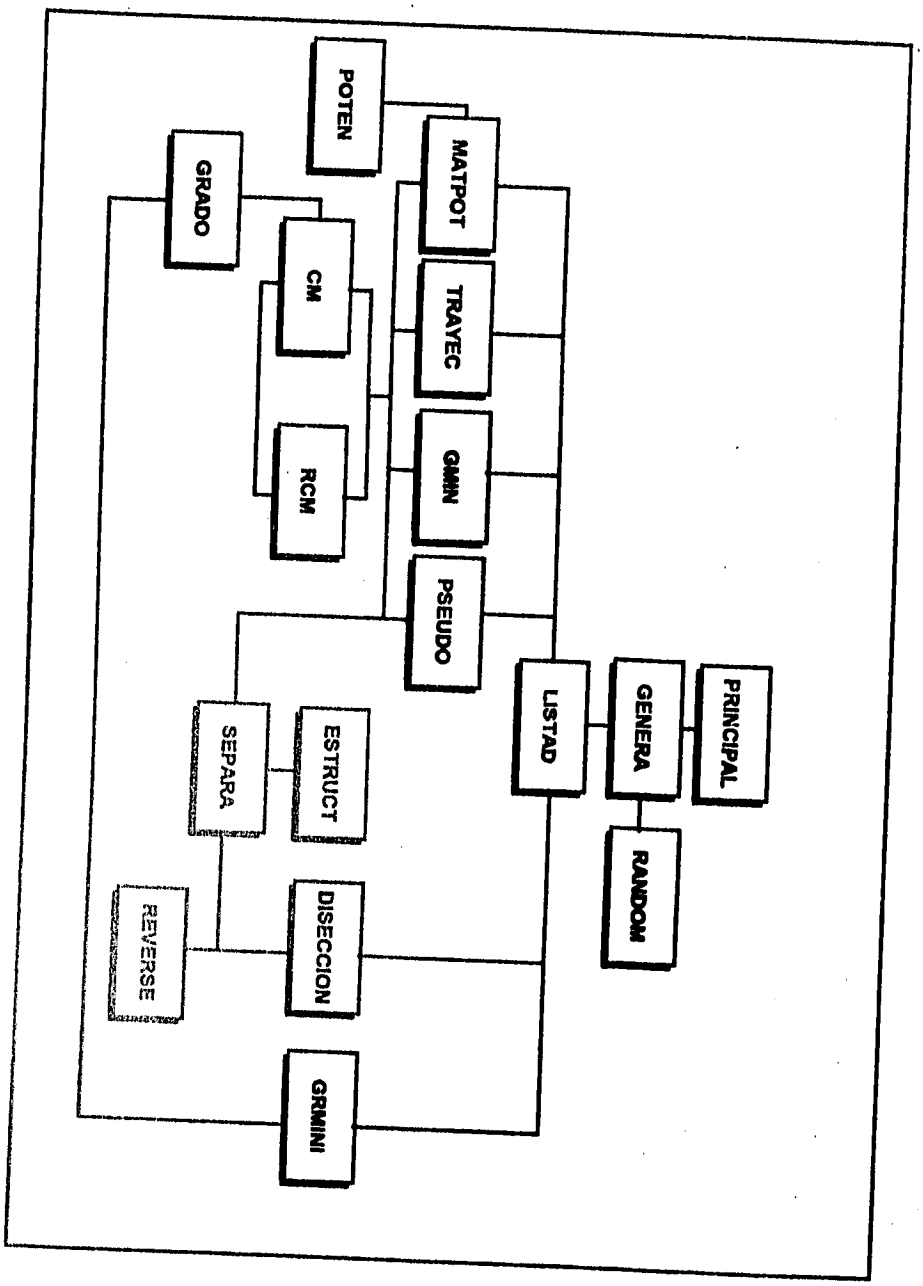
# APÉNDICE

# IMPLEMENTACIÓN COMPUTACIONAL

En esta sección, se presenta la implementación computacional de la mayoría de los algoritmos que se mencionaron en los capítulos anteriores, con la finalidad de que puedan ser utilizados en estudios posteriores relacionados con éste tema.

Se muestra además un diagrama, en donde se aprecia la relación que existe entre cada una de las subrutinas para la elección de nodos iniciales MATPOT, TRAYEC, GMIN, PSEUDO, con las que generan los distintos reordenamientos CM, RCM, DISECCION Y GRMINI.

Para la implementación, se utilizó el lenguaje de programación FORTRAN-77, y cada subrutina tiene una breve introducción donde se detalla su función en el programa.


POTEN
MATPOT
GRADO
CM
TRAYEC
RCM
GMIN
PSEUDO
LISTAD
GENERA
PRINCIPAL
SEPARA
ESTRUCT
RANDOM
REVERSE
DISECCION
GRMINI

## GENERA

Esta subrutina genera una matriz de manera aleatoria, rala y simétrica cuyas entradas son ceros o unos. La raleza o cantidad de ceros que contiene, se controla al tomar solo a los elementos que se obtienen de RANDOM que son mayores a un cierto número, calcula también el ancho de banda y el perfil.

```
C*************************************************************************************
C
C                    ********** GENERA **********
C
C*************************************************************************************

      SUBROUTINE GENERA(A,N)
C*
      INTEGER N,T,I,J,K,A(N,N),BANDAT,BANDA,CONT,GMIN,PERFIL,PERF
      REAL H

       BANDA=0
       PERFIL=0
       PERF=0
       BANDAT=0
       CONT=0
       WRITE(*,*)'* ESCRIBE UN ENTERO POSITIVO (GENERADOR ALEATORIO) * '
       READ 300,T
  300  FORMAT(I10)
       DO 600 I=1,N
        DO 500 J=1,I

C*    LLAMA A LA SUBRUTINA RANDOM Y DEVUELVE UN NÚMERO ENTRE O Y 1

        CALL RANDOM(T,H)
         IF(H.GT.0.95)THEN                  ' Controla el número de elementos distintos
           K=1                              ' de cero en la matriz.
         ELSE
           K=0
         ENDIF
         IF(K.EQ.1.AND.I.NE.J)THEN
           BANDAT=I-J
           IF(BANDAT.GE.BANDA) BANDA=BANDAT
           IF(BANDAT.GE.PERF)  PERF=BANDAT
         ENDIF
         A(I,J)=K
         A(J,I)=K
           IF(I.EQ.J)THEN
             A(I,J)=0
           ENDIF
        IF((A(I,J).NE.0)) CONT=CONT+2
```

```
 500    CONTINUE
        PERFIL=PERFIL+PERF
        PERF=0
 600    CONTINUE
        WRITE(*,*)
        WRITE(*,*)'   ** LA MATRIZ DE COEFICIENTES DISPERSA ES:  **'
        WRITE(*,700)
 700    FORMAT(/,' A=  ')
        WRITE(*,*)
        DO 100 I=1,N
          WRITE(*,110) (A(I,J),J=1,I)
 100    CONTINUE
 110    FORMAT(2513)
        WRITE(*,*)
        WRITE(*,120)CONT,(N*N)
 120    FORMAT(/,5X,'ELEMENTOS NO NULOS *',I5, ' *  DE ',I7)
        WRITE(*,130)BANDA
 130    FORMAT(/,5X,'SU ANCHO DE BANDA ES *',I3, ' * ')
        WRITE(*,140)PERFIL
 140    FORMAT(/,5X,'SU PERFIL ES *',I5, ' * ')
        WRITE(*,*)
        WRITE(*,150)
 150    FORMAT(40X,'* PRESIONA ENTER PARA CONTINUAR * ')
        PAUSE
      RETURN
      END

C*************************************************************************
C*              ********************  RANDOM  ********************
C*
C*            GENERA NÚMEROS ALEATORIOS ENTRE O Y 1, TOMANDO
C*              COMO PARÁMETRO DE ENTRADA EL GENERADOR T
C*
C*************************************************************************
C*
      FUNCTION RANDOM(T,RANX)
C*
      INTEGER T
      REAL RANX
C*
        T=2045*T+1
        T=T-(T/1048576)*1048576          ' Aprovecha el error que se genera al
        RANX=REAL(T+1)/1048577.0         ' obtener el cociente y el producto para
                                         ' un mismo número.
C*
      RETURN
      END
C*
C*************************************************************************
```

## LISTAD

La matriz generada, se alamacena en un par de arreglos unidimensionales ADJNCY y XADJ. El primero contiene a la mitad de los elementos distintos de cero de la matriz y el segundo apunta al inicio de la vecindad asociada a cada nodo. (Sección 1.4). El arreglo SECC, indica subgráficas generadas al almacenar sólo a los elementos no nulos.

```
C*************************************************************************
C*
C*                    ****************** LISTAD ******************
C*
C* OBTIENE LA LISTA DE ADYACENCIAS DE TODOS LOS NODOS DE UNA GRÁFICA
C*
C*************************************************************************
C*
      SUBROUTINE LISTAD(A,XADJ,ADJNCY,SECC,N,GMIN,RAIZ)
C*
      INTEGER N,A(N,N),ADJNCY(1),XADJ(1),SECC(1),K,MIN,J,L,I,
     1  NODO,GMIN,GINI,RAIZ

      K=1
      DO 600 I=1,N
         MIN=0
         DO 500 J=1,N
          IF(A(I,J).GT.0)THEN
             ADJNCY(K)=J                    ' Almacena los vecinos de cada nodo
            IF(MIN.EQ.0)THEN
               XADJ(I)=K                    ' Apunta al inicio de cada vecindad
            ENDIF
            MIN=MIN+1
            K=K+1
          ENDIF
500     CONTINUE
        IF(MIN.EQ.0)THEN
          SECC(I)=0
        ELSE
          SECC(I)=1
        ENDIF
600   CONTINUE
      XADJ(N+1)=K
      ADJNCY(K)=0
      GMIN=N

      DO 700 NODO=1,N                       ' Toma como inicial a un nodo
        GINI=XADJ(NODO+1)-XADJ(NODO)        ' de grado mínimo (GMIN)
        IF(GINI.LE.GMIN)THEN
          GMIN=GINI
```

```
          RAIZ=NODO
        ENDIF
 700  CONTINUE

C*
C*   IMPRIME LA LISTA DE ADYACENCIAS
C*
      WRITE(*,800)
 800  FORMAT(//,'** LISTA DE ADYACENCIAS **')
      WRITE(*,*)
      DO 950 I=1,N
        IF(ADJNCY(I).GT.0)THEN
          WRITE(*,900)I,(ADJNCY(L),L=XADJ(I),XADJ(I+1)-1)
 900      FORMAT('ADJ(',I3,')=',17I4)
        ENDIF
 950  CONTINUE
      PAUSE
      RETURN
      END
```

C**********

**MATPOT**

Subrutina que encuentra nodos periféricos utilizando el método que describe el algoritmo 2.1.2 del Capítulo II. Utiliza la subrutina POTEN, que eleva a una cierta potencia a la matriz de adyacencias de una gráfica dada. Además, verifica si la gráfica es conexa y calcula el grado de cada nodo.

```
C**********************************************************************************
C*
C*                    ********** MATPOT **********
C*
C*        OBTIENE LA MATRIZ DE DISTANCIAS DE UNA GRAFICA CUALQUIERA,
C*        EN BASE A LA MATRIZ POTENCIA DE SU MATRIZ DE ADYACENCIAS.
C*
C**********************************************************************************

    INTEGER N,MAX,H,Q,I,J,A(N,N),D(N,N),P(N,N),C(N,N)
   1          NUMZ,GMIN,Y,K,CONEX,RAIZ
C*
     MAX=0
     DO 500 I=1,N
        DO 300 J=1,N
          P(I,J)=A(I,J)
          IF((P(I,J)).GT.0.OR.I.EQ.J) THEN
              D(I,J)=1
          ELSE
```

```
              D(I,J)=-1
            ENDIF
300       CONTINUE
500  CONTINUE
     CONEX=-1
     GMIN=N
C*
     DO 200 K=2,N-1
       NUMZ=0
      CALL POTEN(A,P,C,N)
        DO 100 I=1,N
          IF(K.EQ.2.AND.(P(I,I)).LE.GMIN) THEN
          Y=I
          GMIN=P(I,I)
          ENDIF
            DO 90 J=1,I
                IF((P(I,J)).GT.0.AND.(D(I,J)).EQ.-1) THEN
                    D(I,J)=K
                    MAX=K
                    H=I
                    Q=J
                 ENDIF
                IF((D(I,J)).LT.0) THEN
                     NUMZ=NUMZ+1
               ENDIF
 90          CONTINUE
100       CONTINUE
         IF(NUMZ.EQ.0)GO TO 250
         IF(CONEX.EQ.NUMZ)GO TO 240
         CONEX=NUMZ
200  CONTINUE
     WRITE(*,*)'MATRIZ DE DISTANCIAS'
     DO 150 I=1,N
       WRITE(*,160)(D(I,J),J=1,I)
150  CONTINUE
160  FORMAT(50I1)
240  WRITE(*,*) '** LA GRAFICA ASOCIADA A ESTA MATRIZ NO ES CONEXA **'
250  CONTINUE
     WRITE(*,270)H,Q,MAX
270  FORMAT(/,'LA DISTANCIA MAXIMA ES d(',I3,',',I3,')= ',I3)
     RAIZ = H
     WRITE(*,280)Y,GMIN
280  FORMAT(/,'EL NODO DE GRADO MINIMO ES ',I3,' CON',I3,' VECINOS.')
     RETURN
     END

C**********************************************************************
```

```
C*************************************************************************************
C*                  ******************  POTEN  ******************
C*
C*     ELEVA LA MATRIZ DE ADYACENCIAS A LA POTENCIA NECESARIA PARA
C*        OBTENER TODAS LAS DISTANCIAS PARA CADA PAR DE NODOS
C*
C*************************************************************************************

      SUBROUTINE POTEN(A,P,C,N)

      INTEGER N,A(N,N),P(N,N),C(N,N),I,J,K
C
      DO 100 I=1,N
       DO 100 J=1,N
        C(I,J)=0
         DO 100 K=1,N
  100     C(I,J)=C(I,J)+A(I,K)*P(K,J)
          DO 500 I=1,N
          DO 500 J=1,N
  500       P(I,J)=C(I,J)
C*
      RETURN
      END
C*************************************************************************************
```

**TRAYEC**

Esta subrutina, encuentra las distancias entre todo par de vértices, utilizando el método que describe el algoritmo 2.1.3 del Capítulo II. Además, verifica la conexidad de la gráfica resultante con la variable CONEX, almacena la distancia máxima en la variable MAX y obtiene a los nodos periféricos denotados por Q Y R.

```
C*************************************************************************************
C*
C*            ************************  TRAYEC  ************************
C*
C*  OBTIENE TRAYECTORIAS DE LONGITUD MÍNIMA PARA TODO PAR DE VÉRTICES
C*
C*************************************************************************************
C*
      SUBROUTINE TRAYEC(A,Q,N)
C*
      INTEGER N,A(N,N),CONEX,MAX,Q,R,NUMZ,I,J,K,L
C*
      NUMZ=0                          ' Almacena el total de elementos nulos
      DO 100 I=1,N                    ' en la variable NUMZ
        DO 100 J=1,N
```

```
      IF((A(I,J)).EQ.0.AND.I.NE.J)NUMZ=NUMZ+1
 100 CONTINUE
C*
      CONEX=-1
      MAX=1
      DO 700 K=1,N
        DO 500 I=1,N
          DO 300 J=1,N
            IF((A(I,J)).GE.MAX)THEN
               MAX=A(I,J)                    ' Calcula los nodos periféricos
               Q=I
               R=J
            ENDIF
            IF((A(I,J)).EQ.1)THEN            ' Recorre la matriz por renglón y columna en
               DO 200 L=1,N                  'busca de elementos nulos, los cuales actualiza
                 IF((A(L,J)).EQ.K.AND.(A(I,L)).EQ.0.AND.I.NE.L)THEN
                    A(I,L)=K+1
                    A(L,I)=K+1
                    NUMZ=NUMZ-2
                 ENDIF
 200           CONTINUE
            ENDIF
 300      CONTINUE
C*
 500  CONTINUE
        IF(NUMZ.EQ.CONEX)GO TO 800           ' Indica si la matriz es conexa o no.
        WRITE(*,*)'MATRIZ DE DISTANCIAS'
        WRITE(*,600)
 600   FORMAT(/,' D = ')
        DO 650 I=1,N
 650     WRITE(*,680)(A(I,J),J=1,I)          ' Imprime la matriz resultante
 680    FORMAT(25I3)
        IF(NUMZ.EQ.0)GO TO 900
        CONEX=NUMZ                           ' Si aún hay elementos nulos
 700  CONTINUE                               ' regresa
 800  WRITE(*,*)
      WRITE(*,*) '* LA GRAFICA ASCOCIADA A ESTA MATRIZ NO ES CONEXA *'
 900  WRITE(*,950)Q,R,MAX
 950   FORMAT(/,'LA DISTANCIA MAXIMA ES D(',I3,',',I3,')=',I3)
      RETURN
      END

C*************************************************************************
```

## PSEUDO

Encuentra nodos pseudoperiféricos al generar las estructuras de nivel y tomando un nodo de grado mínimo del último nivel, si la excentricidad de éste es mayor que la del nodo anterior lo toma como inicial y se ejecuta nuevamente, sino el nodo anterior se elige como inicial (Algoritmo 2.1.1 Capítulo II).

```
C*************************************************************************
C*                    ********** PSEUDO **********
C*
C*     ENCUENTRA UN NODO INICIAL PARA CADA ESTRUCTURA DE NIVEL
C*
C*************************************************************************

      SUBROUTINE PSEUDO(RAIZ, XADJ, ADJNCY, SEC, NLVL, XLS, LS)
C*
      INTEGER RAIZ,ADJNCY(1),LS(1),SECC(1),XLS(1),XADJ(1),
     1 NLVL,CCSIZE,JSTRT,MINDEG,J,NODO,NDEG,KSTRT,KSTOP,NABOR,
     1 NLVLN,K
C*
C*    DETERMINA LA ESTRUCTURA DE NIVEL CIMENTADA EN RAIZ
C*
C     RAIZ=1
      CALL ESTRUC(RAIZ, XADJ, ADJNCY, SECC, NLVL, XLS, LS)
      CCSIZE = XLS(NLVL+1)-1
      IF (NLVL.EQ.1.OR.NLVL.EQ.CCSIZE)RETURN
C*
C* TOMA UN NODO DE GRADO MINIMO DEL ULTIMO NIVEL
C*
 100  JSTRT = XLS(NLVL)
      MINDEG = CCSIZE
      RAIZ = LS(JSTRT)
      IF(CCSIZE.EQ.JSTRT)GO TO 400
        DO 300 J = JSTRT,CCSIZE
          NODO = LS(J)
          NDEG = 0
          KSTRT = XADJ(NODO)
          KSTOP = XADJ(NODO+1)-1
            DO 200 K = KSTRT,KSTOP
              NABOR = ADJNCY(K)
              IF(SECC(NABOR).GT.0) NDEG = NDEG+1
 200         CONTINUE
            IF(NDEG.GE.MINDEG)GO TO 300
              RAIZ = NODO
              MINDEG = NDEG
 300    CONTINUE
C*
C*   SE GENERA SU ESTRUCTURA DE NIVEL
```

```
C*
 400   CALL ESTRUC(RAIZ,XADJ,ADJNCY,SECC,NLVLN,XLS,LS)
         IF(NLVLN.LE.NLVL)RETURN
         NLVL = NLVLN
         IF(NLVL.LT.CCSIZE) GO TO 100
         RETURN
       END
```

C**********

## CMKEE

Obtiene el ordenamiento de Cuthill-Mckee descrito en el Capítulo III, para cada componente conexa especificada por SECC y RAIZ, el arreglo PERM contiene el ordenamiento CM, utiliza a GRADO para establecer el grado de cada nodo en el presente nivel.

```
C**********
C*
C*                    ****************** CMKEE ******************
C*
C*          DA UNA NUMERACIÓN A LOS NODOS DE UNA GRÁFICA EN ORDEN
C*                          CRECIENTE DE LOS GRADOS.
C*
C**********

      SUBROUTINE CMKEE(RAIZ,XADJ,ADJNCY,SECC,PERM,CCSIZE,
     1  DEG,MARCA)
C*
      INTEGER ADJNCY(1),DEG(1),SECC(1),PERM(1),XADJ(1),
     1        CCSIZE,FNBR,I,J,JSTOP,JSTRT,K,L,LBEGIN,LNBR,
     1        LPERM,LVLEND,NBR,NODO,RAIZ
C*
C*    ** OBTIENE LOS GRADOS DE TODOS LOS NODOS EN LA GRAFICA **
C*
      CALL GRADO(RAIZ,XADJ,ADJNCY,SECC,DEG,CCSIZE,PERM)
      SECC(RAIZ)=0
      IF(CCSIZE.LE.1)RETURN
      LVLEND=0
      LNBR=1
C*
C*   ** LBEGIN,LVLEND APUNTAN AL PRINCIPIO Y FINAL DEL NIVEL ACTUAL **
C*
      M=1
 100  LBEGIN=LVLEND+1
      LVLEND=LNBR
      DO 600 I=LBEGIN,LVLEND
C*
```

```
C * PARA CADA NODO DEL PRESENTE NIVEL *
C*
      NODO=PERM(I)
      JSTRT=XADJ(NODO)
      JSTOP=XADJ(NODO+1)-1
C*
C     ** ENCUENTRA LOS VECINOS NO NUMERADOS DE NODO. FNBR Y LNBR
C     APUNTAN RESPECTIVAMENTE AL PRINCIPIO Y AL FINAL DEL CONJUNTO
C     DE VECINOS NO NUMERADOS DEL NODO ACTUAL ESPECIFICADO EN PERM **
C*
      FNBR=LNBR+1
      DO 200 J=JSTRT,JSTOP
         NBR=ADJNCY(J)
           IF(SECC(NBR).EQ.0)GO TO 200
             LNBR=LNBR+1
             SECC(NBR)=0
             PERM(LNBR)=NBR
200     CONTINUE
      IF(FNBR.GE.LNBR)GO TO 600
C*
C   ** NUMERA A LOS VECINOS DE NODO EN ORDEN CRECIENTE A SUS GRADOS **
        K=FNBR
300     L=K
        K=K+1
        NBR=PERM(K)
400       IF(L.LT.FNBR)GO TO 500
          LPERM=PERM(L)
          IF(DEG(LPERM).LE.DEG(NBR))GO TO 500
            PERM(L+1)=LPERM
            L=L-1
            GO TO 400
500           PERM(L+1)=NBR
          IF(K.LT.LNBR)GO TO 300
600     CONTINUE
      IF(LNBR.GT.LVLEND)GO TO 100
      IF(MARCA.EQ.1)RETURN

C     ** IMPRIME EL ORDENAMIENTO CM **
      WRITE(*,*)
      WRITE(*,*)'** EL ORDENAMIENTO (CM) ES: **'
      WRITE(*,*)
      PAUSE
      DO 750 I=1,CCSIZE
       WRITE(*,650)PERM(I),I
650     FORMAT('NODO(',I3,')=',I3)
750     CONTINUE
      RETURN
      END
C*******************************************************************************
```

## GRADO

Registra los grados de los nodos en el arreglo DEG, de cada componente especificada con SECC y RAIZ, el arreglo XADJ, es utilizado temporalmente para indicar cuales nodos han sido ya considerados. El arreglo LS, se utiliza para almacenar los nodos de la componente actual nivel por nivel.

```
C*************************************************************************
C*                    ***************** GRADO *****************
C*
C*                       OBTIENE EL GRADO DE CADA NODO
C*************************************************************************
      SUBROUTINE GRADO(RAIZ,XADJ,ADJNCY,SECC,DEG,CCSIZE,LS)
C*
      INTEGER ADJNCY(1),DEG(1),LS(1),SECC(1),XADJ(1),CCSIZE,
     1        I,IDEG,J,JSTOP,JSTRT,LBEGIN,LVLEND,LVSIZE,NBR,
     1        NODO,RAIZ
C*
      LS(1)=RAIZ
      XADJ(RAIZ)=-XADJ(RAIZ)
      LVLEND=0
      CCSIZE=1
  100 LBEGIN=LVLEND+1
      LVLEND=CCSIZE

C * ENCUENTRA LOS GRADOS DE LOS NODOS POR SECCIONES *
C*
      DO 400 I=LBEGIN,LVLEND
       NODO=LS(I)
       JSTRT=-XADJ(NODO)
       JSTOP=IABS(XADJ(NODO+1))-1
       IDEG=0
       IF(JSTOP.LT.JSTRT)GO TO 300
        DO 200 J=JSTRT,JSTOP
          NBR=ADJNCY(J)
          IF(SECC(NBR).EQ.0)GO TO 200
            IDEG=IDEG+1
            IF(XADJ(NBR).LT.0)GO TO 200
              XADJ(NBR)=-XADJ(NBR)
              CCSIZE=CCSIZE+1
              LS(CCSIZE)=NBR
  200   CONTINUE
  300   DEG(NODO)=IDEG
  400 CONTINUE
C*
C  * CALCULA EL ANCHO DE ESTE NIVEL *
C*
      LVSIZE=CCSIZE-LVLEND
```

```
      IF(LVSIZE.GT.0)GO TO 10
      DO 500 I=1,CCSIZE
         NODO=LS(I)
         XADJ(NODO)=-XADJ(NODO)
500   CONTINUE
      RETURN
      END
C*******************************************************************************
```

**RCM**

Obtiene el ordenamiento de Cuthill-Mckee con inversión, el cual aunque no mejora el ancho de banda obtenido por el algoritmo CM, si mejora en gran medida el perfil.

```
C*******************************************************************************
C*
C*                        ***************** RCM *****************
C*
C*          DA UNA NUMERACIÓN INVERSA A LA OBTENIDA EN EL CM.
C*
C*******************************************************************************
C*
      SUBROUTINE RCM(RAIZ,XADJ,ADJNCY,SECC,PERM,CCSIZE,DEG)
C*
      INTEGER PERM(1),LPERM,L,K,CCSIZE,RAIZ,XADJ(1),ADJNCY(1),
     1        SECC(1),DEG(1),I
C*
      MARCA=1
      CALL CMKEE(RAIZ,XADJ,ADJNCY,SECC,PERM,CCSIZE,DEG,MARCA)
C*
      K=CCSIZE/2
      L=CCSIZE
      DO 100 I=1,K
           LPERM=PERM(L)
           PERM(L)=PERM(I)
           PERM(I)=LPERM
           L=L-1
100   CONTINUE
C*
      WRITE(*,*)
      WRITE(*,*)'** EL ORDENAMIENTO (RCM) ES: ** '
      WRITE(*,*)
        DO 200 I=1,CCSIZE
          WRITE(*,300)PERM(I),I
300       FORMAT('NODO(',I3,')=',I3)
200   CONTINUE
      RETURN
      END
C*******************************************************************************
```

## DIAN

Utilizando la subrutina SEPARA se obtienen separadores que dividen a una gráfica en partes más o menos iguales, así se logra dar una numeración que minimiza el llenado, aún más que el obtenido por el algoritmo de grado mínimo. Utiliza también la subrutina REVERSE para cambiar la numeración asignada a los separadores ya que estos deben numerarse hasta el final para obtener la permutación deseada. La subrutina ESTRUCT genera las estructura de nivel que le permite a SEPARA dividir a la subgráfica en dos partes iguales.

```
C*************************************************************************
C*
C*                        ********** DIAN **********
C*
C*      ESTA SUBRUTINA GENERA UN ORDENAMIENTO DE LOS NODOS DE
C*       UNA GRAFICA, UTILIZANDO EL METODO DE DISECCION ANIDADA
C*                          ( NESTED DISSECTION )
C*
C*************************************************************************

      SUBROUTINE DIAN (N,XADJ,ADJNCY,SECC,PERM,XLS,LS )

C     ************************  ************************

      INTEGER ADJNCY(1), SECC(1), LS(1), PERM(1), XLS(1),
     1        XADJ(1), I, N, NSEP, NUM, RAIZ

C     ************************  ************************

      DO 100 I=1,N
       SECC(I)=1
  100  CONTINUE
      NUM=0
      DO 300 I=1,N
C*
C* *** PARA CADA COMPONENTE.. ***
C*
  200  IF (SECC(I).EQ.0) GO TO 300
      RAIZ = I
C*
C*** ENCUENTRA UN CONJUNTO SEPARADOR Y LO NUMERA ***
C*
      CALL SEPARA (RAIZ, XADJ, ADJNCY, SECC, NSEP,
     1            PERM(NUM+1), XLS, LS)
C*
        NUM=NUM+NSEP
        IF(NUM.GE.N) GO TO 400
        GO TO 200
```

```
 300  CONTINUE
C*
C     *** DEBIDO A QUE LOS SEPARADORES SE DEBEN NUMERAR AL
C*        FINAL, LA SUBRUTINA REVRSE DA UN ORDEN INVERSO
C*              A LOS NODOS DEL SEPARADOR          ***
C*
 400  CALL REVRSE(N,PERM)
       WRITE(*,*)
       WRITE(*,*)'LA PERMUTACION ES: '
       WRITE(*,*)
        DO 550 I=1,N
         WRITE(*,500)I,PERM(I)
 500     FORMAT('NODO(',I3,')=',I3)
 550  CONTINUE
C*
     RETURN
     END

C*******************************************************************************
C*
C*                          ******** REVRSE ********
C*
C*                    CAMBIA EL ORDEN DE LA PERMUTACION
C*
C*******************************************************************************

     SUBROUTINE REVRSE(N,PERM)
C*
     INTEGER N,PERM(1)
C*
     DO 100 I=1,N
      DO 100 J=N,1
       PERM(J)=PERM(I)
 100  CONTINUE
C*
       RETURN
     END

C*******************************************************************************
C*
C*                        ********** SEPARA **********
C*
C*             ENCUENTRA UN SEPARADOR MINIMO PARA CADA COMPONENTE
C*               CONEXA ESPECIFICADA POR SECC EN LA GRAFICA DADA
C*
C*******************************************************************************

     SUBROUTINE SEPARA (RAIZ, XADJ, ADJNCY, SECC, NSEP, SEP,
    1               XLS, LS)
```

```
C*
      INTEGER ADJNCY(1), LS(1), SECC(1), SEP(1), XLS(1), XADJ(1),
     1        I,J,JSTOP,JSTRT, MIDBEG, MIDENED, MIDLVL, MP1BEG,
     1        MP1END, NBR, NLVL, NODE, NSEP, RAIZ

C*
      CALL PSEUDO(RAIZ, XADJ, ADJNCY, SECC, NLVL, XLS, LS)

C*      *** SI EL NIMERO DE NIVELES ES MENOR A 3, TODA LA COMPONENTE
C*                SE TOMARIA COMO EL SEPARADOR          ***

      IF ( NLVL.GE.3 ) GO TO 200
        NSEP = XLS(NLVL+1)-1
        DO 100 I=1,NSEP
          NODO=LS(I)
          SEP(I)=NODO
          SECC(NODO)=0
  100     CONTINUE
          RETURN
C*
C     *** ENCUENTRA EL NIVEL MEDIO DE LA ESTRUCTURA ***
C*
  200   MIDLVL = (NLVL+2)/2
        MIDBEG = XLS(MIDLVL)
        MP1BEG = XLS(MIDLVL +1)
        MIDEND = MP1BEG -1
        MP1END = XLS(MIDLVL+2)-1
C*
      DO 300 I = MP1BEG,MP1END
        NODO = LS(I)
        XADJ(NODO) = - XADJ(NODO)
  300   CONTINUE
C*
          NSEP = 0
          DO 500 I=MIDBEG,MIDEND
            NODO = LS(I)
            JSTRT = XADJ(NODO)
            JSTOP = IABS(XADJ(NODO+1))-1
              DO 400 J=JSTRT,JSTOP
                NBR = ADJNCY(J)
                IF(XADJ(NBR).GT.0)GO TO 400
                  NSEP = NSEP + 1
                  SEP(NSEP) = NODO
                 SECC(NODO) = 0
                 GO TO 500
  400         CONTINUE
  500     CONTINUE
C*
```

```
C*
      INTEGER ADJNCY(1), LS(1), SECC(1), SEP(1), XLS(1), XADJ(1),
     1        I,J,JSTOP,JSTRT, MIDBEG, MIDENED, MIDLVL, MP1BEG,
     1        MP1END, NBR, NLVL, NODE, NSEP, RAIZ

C*
      CALL PSEUDO(RAIZ, XADJ, ADJNCY, SECC, NLVL, XLS, LS)

C*     *** SI EL NIMERO DE NIVELES ES MENOR A 3, TODA LA COMPONENTE
C*                SE TOMARIA COMO EL SEPARADOR          ***

      IF ( NLVL.GE.3 ) GO TO 200
        NSEP = XLS(NLVL+1)-1
        DO 100 I=1,NSEP
          NODO=LS(I)
          SEP(I)=NODO
          SECC(NODO)=0
  100     CONTINUE
          RETURN
C*
C     *** ENCUENTRA EL NIVEL MEDIO DE LA ESTRUCTURA ***
C*
  200   MIDLVL = (NLVL+2)/2
        MIDBEG = XLS(MIDLVL)
        MP1BEG = XLS(MIDLVL +1)
        MIDEND = MP1BEG -1
        MP1END = XLS(MIDLVL+2)-1
C*
      DO 300 I = MP1BEG,MP1END
        NODO = LS(I)
        XADJ(NODO) = - XADJ(NODO)
  300   CONTINUE
C*
          NSEP = 0
          DO 500 I=MIDBEG,MIDEND
           NODO = LS(I)
           JSTRT = XADJ(NODO)
           JSTOP = IABS(XADJ(NODO+1))-1
             DO 400 J=JSTRT,JSTOP
               NBR = ADJNCY(J)
               IF(XADJ(NBR).GT.0)GO TO 400
                 NSEP = NSEP + 1
                 SEP(NSEP) = NODO
                SECC(NODO) = 0
                GO TO 500
  400         CONTINUE
  500     CONTINUE
C*
```

```
C  *** ASIGNA A XADJ SU SIGNO CORRECTO ***
C*
      DO 600 I = MP1BEG,MP1END
        NODO = LS(I)
        XADJ(NODO) = - XADJ(NODO)
 600    CONTINUE
C*
       RETURN
       END

C*******************************************************************************
```

**ESTRUC**

Genera una estructura de nivel cimentada en el nodo RAIZ, la cual se almacena en los arreglos (XLS,LS), NLVL indica el número de niveles obtenidos para cada estructura y aquellos nodos marcados con SECC = 0 se ignoran.

```
C*******************************************************************************
C*
C*                    ****************** ESTRUC ******************
C*
C*        GENERA LA ESTRUCTURA DE NIVEL CIMENTADA EN EL NODO RAIZ
C*
C*******************************************************************************

      SUBROUTINE ESTRUC(RAIZ,XADJ,ADJNCY,SECC,NLVL,XLS,LS)
C*
      INTEGER ADJNCY(1),LS(1),XLS(1),SECC(1),LVSIZE,
     1        XADJ(1),I,J,JSTOP,JSTRT,LBEGIN,CCSIZE,LVLEND,
     1        NODO,RAIZ,NLVL,NBR
C*
C*
      SECC(RAIZ)=0
      LS(1)=RAIZ
      NLVL=0
      LVLEND=0
      CCSIZE=1
C*
C  * LBEGIN APUNTA AL PRINCIPIO Y LVLEND AL FINAL DEL NIVEL ACTUAL *
C*
 200  LBEGIN=LVLEND+1
      LVLEND=CCSIZE
      NLVL=NLVL+1
      XLS(NLVL)=LBEGIN
C*
```

```
C  * SE GENERA EL SIGUIENTE NIVEL *
C*
      DO 400 I=LBEGIN,LVLEND
        NODO=LS(I)
        JSTRT=XADJ(NODO)
        JSTOP=XADJ(NODO+1)-1
       IF(JSTOP.LT.JSTRT)GO TO 400
         DO 300 J=JSTRT,JSTOP
            NBR=ADJNCY(J)
            IF(SECC(NBR).EQ.0)GO TO 300
            CCSIZE=CCSIZE+1
            LS(CCSIZE)=NBR
            SECC(NBR)=0
 300   CONTINUE
 400 CONTINUE
C*
C * CALCULA EL ANCHO DEL PRESENTE NIVEL *
C*
      LVSIZE=CCSIZE-LVLEND
      IF(LVSIZE.GT.0)GO TO 200
C*
      XLS(NLVL+1)=LVLEND+1
      DO 500 I=1,CCSIZE
        NODO=LS(I)
        SECC(NODO)=1
 500  CONTINUE
C*
      WRITE(*,510)RAIZ,NLVL
 510   FORMAT(/,5X,'* LA ESTRUCTURA DE NIVEL CIMENTADA EN EL NODO '
     1 ,I3,' TIENE',I3,' NIVELES * ')
      RETURN
      END

C*************************************************************************************
```

**GRMINI**

Produce un ordenamiento que intenta minimizar el llenado, elimina a cada nodo de grado mínimo y si los vecinos a éste no son adyacentes, se unen con una nueva arista. El nodo eliminado se numera y se calcula nuevamente el grado de cada nodo. Este proceso continua hasta que todos los nodos son eliminados, el llenado es el total de aristas agregadas.

```
C*************************************************************************
C*
C*                  ****************** GRMINI ******************
C*
C*    ENCUENTRA UNA NUMERACION SEGUN EL ALGORITMO DE GRADO MINIMO
C*
C*
C*************************************************************************
C*
      SUBROUTINE GRMINI(RAIZ,XADJ,ADJNCY,SECC,PERM,DEG,N,LS,DEGR,ADJ)
C*
      INTEGER N,XADJ(1),ADJNCY(1),I,J,SECC(1),L,W,NABOR,DEG(1),
     1 VECINO,NODO,GMIN,INICIA,NGRADO,IINI,IFIN,PERM(1),CC,RAIZ,
     1 CCSIZE,LS(1),NNODO,NRAIZ,VECINA,T,DELET,CAMB,VAL,DEGR(1),
     1 CAMV,S,ADJ(1)
C*
C*    CALCULA EL GRADO DE CADA NODO
C*
      DO 225 NODO = 1,N
       IINI=XADJ(NODO)
       IFIN=XADJ(NODO+1)-1
          DO 223 I=IINI,IFIN
           ADJ(I)=ADJNCY(I)                    'Almacena el arreglo ADJNCY, en
 223         CONTINUE                          ' un arreglo auxiliar ADJ que será
       PERM(NODO)=NODO                         ' destruido.
       DEGR(NODO)=0
 225    CONTINUE
      LL=0
C*
      DO 600 CC=1,N-1
      GMIN = N                                 'Calcula el grado en cada iteración
      CALL GRADO(RAIZ,XADJ,ADJNCY,SECC,DEG,CCSIZE,LS)
      DO 330 NODO = 1,N
          DEG(NODO)=DEG(NODO)-DEGR(NODO)
          IF(SECC(NODO).EQ.0.OR.DEG(NODO).EQ.0)GO TO 30
          NGRADO=DEG(NODO)
              IF(NGRADO.LT.GMIN)THEN
                  GMIN=NGRADO
                  NNODO=NODO                   ' De cada iteración toma el nodo
              ENDIF                                    'de grado mínimo.
 330   CONTINUE
       INICIA=NNODO
       IINI=XADJ(INICIA)
       IFIN=XADJ(INICIA+1)-1
      IF(IFIN.LT.IINI)RETURN
      VAL=IFIN-IINI
C*
      DO 400 I=IINI,IFIN
       VECINO = ADJ(I)                         'Verifica si los nodos vecinos al
```

```
      IF(VECINO.EQ.INICIA)GO TO 400        'Inicial son adyacentes, sino los une
      IF(VAL.EQ.0)THEN                     'con una nueva arista.
      DO 295 L=XADJ(VECINO),XADJ(VECINO+1)-1
          IF(ADJ(L).EQ.INICIA)THEN
                ADJ(L)=VECINO
                ADJ(I)=INICIA
                DEGR(INICIA)=1
                DEGR(VECINO)=DEGR(VECINO)+1
                SECC(INICIA)=0
                GO TO 400
           ENDIF
295   CONTINUE
      ENDIF
C*
      MARCA=0
      MARC=0
      DO 300 K = I+1,IFIN
         VECINA = ADJ(K)
         IF(VECINA.EQ.INICIA)GO TO 300

         DO 296 L=XADJ(VECINO),XADJ(VECINO+1)-1
           IF(ADJ(L).EQ.INICIA) THEN
              CAMB=L
              IF(MARCA.EQ.1)GO TO 440
           ENDIF
           IF(ADJ(L).EQ.VECINA) MARCA=1
296       CONTINUE
C*
          DO 297 M=XADJ(VECINA),XADJ(VECINA+1)-1
            IF(ADJ(M).EQ.INICIA) THEN
                 CAMV=M
                 IF(MARC.EQ.1)GO TO 450
            ENDIF
             IF(ADJ(M).EQ.VECINO) MARC=1
297         CONTINUE
C*
300       CONTINUE
400   CONTINUE
C*
440    IF(MARCA.EQ.0)THEN
          ADJ(CAMB)=VECINA
          LL=LL+1
       ELSE
          DEGR(VECINO)=DEGR(VECINO)+1
          ADJ(CAMB)=VECINO
       ENDIF
       SECC(INICIA)=0
450    IF(MARC.EQ.0)THEN
         ADJ(CAMV)=VECINO
```

```
            ELSE
              DEGR(VECINA)=DEGR(VECINA)+1
              ADJ(CAMV)=VECINA
            ENDIF
            DO 298 T=IINI,IFIN
298           ADJ(T)=INICIA
            PERM(INICIA)=CC
            IF(CC.EQ.N)GO TO 500
            RAIZ=INICIA
600   CONTINUE
C*
500   WRITE(*,*)
      WRITE(*,*)'** EL ORDENAMIENTO DE GRADO MINIMO ES: ** '
      WRITE(*,*)
      DO 401 L=1,N
        WRITE(*,355)L,PERM(L)
355       FORMAT('NODO(',I3,')=',I3)
401     CONTINUE
      WRITE(*,*)
      WRITE(*,800)LL
800   FORMAT(10X,' ** SU LLENADO ES: ',I6,' ** ')
      PAUSE
     RETURN
    END

C*************************************************************************************
```

# BIBLIOGRAFÍA

[1] ABRIN, V. Algoritmos para Reordenar Matrices Ralas. Tesis de Maestría, Facultad de Ciencias UNAM, 1985.

[2] BUCKLEY F., HARARY F. Distance in Graphs. Addison-Wesley. 1990.

[3] CALDERÓN A. Yalepack, Subrutinas para Resolver Sistemas Lineales Grandes y poco Densos. IIMAS, UNAM. Manual 1 1991.

[4] CUTHILL E. Several Strategies for Reducing the Bandwidth of Matrices. De Rose y Willoughby. pp. 157-166.

[5] CHENG K. Minimizing the Bandwidth of Sparse Symmetric Matrices. Departament of Applied Mathematics and Statistics. State University of New York. 1972.

[6] CHINN, P. Z. , CHVÁTALOVÁ J. , DEWDNEY A. K. , GIBBS N.E. The Bandwidth Problem for Graphs and Matrices A Survey. Journal of Graph Theory Vol. 6 No. 3 1982.

[7] CHVÁTAL, V. A Remark on a Problem of Harary. Czechoslovak Math. Journal No.20 pp. 249-250. 1970.

[8] CHVÁTALOVÁ J. On the Bandwidth Problem for Graphs. Departament of Combinatorics and Optimization, University of Waterloo 1980.

[9] CHVÁTALOVÁ J. , DEWDNEY A. K. , GIBBS N.E. , KORFHAGE R. R. The Bandwidth Problem for Graphs: a Collection of recent results. Research Report 24, Departament of Computer Sciencie, UWO, London, Ontario (1975).

[10] DEWDNEY A.K. The Bandwidth Problem for Graphs. In 7th S.E. Conference on Combinatorics, Graph Theory and Computing. Utilitas Mathematica, Winnipeg 1976.

[11] DIMITRY D., MOTT T. Introduction to Fortran IV Programming. Rinehart and Winston.1966.

[12] GEORGE A. , LIU J. W. Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Englewood Cliffs, New Jersey. 1981.

[13] GIBBONS A, RYTTER W. Efficient Parallel Algorithms. Cambridge University Press 1988.

[14] GIBBS N.E., POOLE N.G., STOCKEMEYER P.K. An Algorithm for Reducing the Bandwidth and the Profile of a Sparse Matrix. SIAM Journal Number Anual 13. pp. 236-250. 1976.

[15] HARARY, F. Graph Theory. Addison Wesley, Reading M.A. 1971

[16] HEATH M. T. , RAGHAVAN P. A Cartesian Parallel Nested Dissection Algorithm. SIAM Journal on Matrix Analysis and Applications. Vol 16 No 1. Enero 1995. pp. 235-253.

[17] HOROWITZ E. Fundamentals of Computer Algorithms. Computer Science Press. 1984.

[18] LORENZO, A. Análisis de los Métodos Frontal y Multifrontal. Tesis de Licenciatura, Facultad de Ciencias UNAM, 1994.

[19] RAMESH H., KAPOOR S. Algorithms for Enumerating all Spanning Trees of undirected and weighted Graphs. SIAM Journal on Computing Vol 24 No. 2 . Abril 1995 . pp. 247-265.

[20] SEDGEWICK R., Algorithms. Brown University Addison-Wesley 1983.

[21] TANENBAW A. Computer Networks. Pretince-Hall, Englewood Cliffs New Jersey 1989.

[22] TEWARSON R., Sparse Matrices. Vol. 99 in Mathematics in Science and Engineering. Departament of Applied Mathematics and Statistics. University of New York 1973.

[23] ZWASS V. Programming in Fortran. Harper and Row, Publishers. 1981

[14] GIBBS N.E., POOLE N.G., STOCKEMEYER P.K. An Algorithm for Reducing the Bandwidth and the Profile of a Sparse Matrix. SIAM Journal Number Anual 13. pp. 236-250. 1976.

[15] HARARY, F. Graph Theory. Addison Wesley, Reading M.A. 1971

[16] HEATH M. T. , RAGHAVAN P. A Cartesian Parallel Nested Dissection Algorithm. SIAM Journal on Matrix Analysis and Applications. Vol 16 No 1. Enero 1995. pp. 235-253.

[17] HOROWITZ E. Fundamentals of Computer Algorithms. Computer Science Press. 1984.

[18] LORENZO, A. Análisis de los Métodos Frontal y Multifrontal. Tesis de Licenciatura, Facultad de Ciencias UNAM, 1994.

[19] RAMESH H., KAPOOR S. Algorithms for Enumerating all Spanning Trees of undirected and weighted Graphs. SIAM Journal on Computing Vol 24 No. 2 . Abril 1995 . pp. 247-265.

[20] SEDGEWICK R., Algorithms. Brown University Addison-Wesley 1983.

[21] TANENBAW A. Computer Networks. Pretince-Hall, Englewood Cliffs New Jersey 1989.

[22] TEWARSON R., Sparse Matrices. Vol. 99 in Mathematics in Science and Engineering. Departament of Applied Mathematics and Statistics. University of New York 1973.

[23] ZWASS V. Programming in Fortran. Harper and Row, Publishers. 1981