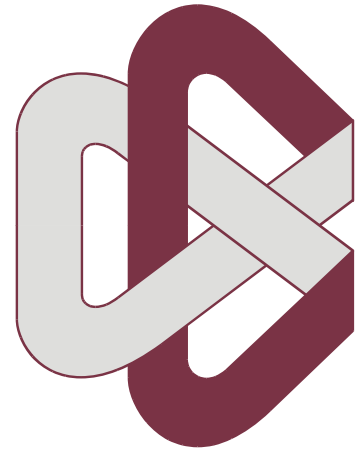


CIMAT

Reordenamiento para la factorización Cholesky

Miguel Vargas-Félix

miguelvargas@cimat.mx
http://www.cimat.mx/~miguelvargas

Factorización Cholesky para matrices dispersas

Describiremos tres estrategias para disminuir el tiempo y uso de memoria en la factorizaciones Cholesky:

- Reordenar los renglones y columnas de la matriz del sistema de ecuaciones para reducir el tamaño de las matrices resultantes de la factorización.

- Utilizar la factorización Cholesky simbólica para obtener la factorización exacta y formar con ésta matrices ralas sin elementos cero.

- Seleccionando la secuencia de llenado de la factorización es posible paralelizar [Heat91].

Para matrices ralas, resultantes de problemas de elemento finito, se puede reducir la complejidad del algoritmo a casi $O\left(\eta\left(\mathbf{A}\right)\right)$, tanto en tiempo de ejecución como en memoria utilizada.

Las estrategias anteriores son aplicables también a la factorización LU aplicada a matrices no simétricas en cuanto a sus valores, pero simétricas con respecto a su estructura.

Reordenamiento

Sea $\mathbf{A}\,\mathbf{x}=\mathbf{b}$ un sistema lineal de ecuaciones,

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix},$$

podemos cambiar el órden de las incógnitas en $\mathbf{x}$, por ejemplo

$$\begin{pmatrix} x_3 \\ x_1 \\ x_4 \\ x_2 \end{pmatrix}.$$

Lo que hace necesario cambiar el orden del las entradas en $\mathbf{A}$ y $\mathbf{b}$. Sea $\mathbf{A}'\,\mathbf{x}'=\mathbf{b}'$,

$$\begin{pmatrix} A_{33} & A_{31} & A_{34} & A_{32} \\ A_{13} & A_{11} & A_{14} & A_{12} \\ A_{43} & A_{41} & A_{44} & A_{42} \\ A_{23} & A_{21} & A_{24} & A_{22} \end{pmatrix} \begin{pmatrix} x_3 \\ x_1 \\ x_4 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_3 \\ b_1 \\ b_4 \\ b_2 \end{pmatrix}.$$

Al resolver este nuevo sistema de ecuaciones tendremos el mismo resultado que pero otro orden.

Matrices de permutación

Una matriz de permutación es una matriz cuadrada con sólo un “1” por renglón y por columna, el resto de las entradas es 0. Por ejemplo, sea **P**,

$$\mathbf{P}=\begin{pmatrix}0&1&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{pmatrix}.$$

Sea **A** la matriz

$$\mathbf{A}=\begin{pmatrix}a&b&c&d\\e&f&g&h\\i&j&k&l\\m&n&o&p\end{pmatrix}.$$

Multiplicando **P** por la izquierda de **A** crea un reordenamiento de **A** por renglón

$$\mathbf{P}\,\mathbf{A}=\begin{pmatrix}0&1&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}a&b&c&d\\e&f&g&h\\i&j&k&l\\m&n&o&p\end{pmatrix}=\begin{pmatrix}e&f&g&h\\m&n&o&p\\a&b&c&d\\i&j&k&l\end{pmatrix}.$$

Multiplicando por la derecha el reordenamiento es por columnas

$$\mathbf{A}\,\mathbf{P}=\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}=\begin{pmatrix} c & a & d & b \\ g & e & h & f \\ k & i & l & j \\ o & m & p & n \end{pmatrix}.$$

Si **A** es una matriz simétrica, dada **P** una matriz de permutación, las permutaciones (reordenamientos) de renglón del tipo

$$\mathbf{A}' \leftarrow \mathbf{P}\,\mathbf{A},$$

o de columna

$$\mathbf{A}' \leftarrow \mathbf{A}\,\mathbf{P},$$

destruyen la simetría de **A** [Golu96 p148].

Para preservar la simetría de **A** solamente podemos considerar reordenamiento de las entradas de la forma

$$\mathbf{A}' \leftarrow \mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}.$$

Siguiendo el ejemplo,

$$\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}=\begin{pmatrix}0&1&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}a&b&c&d\\e&f&g&h\\i&j&k&l\\m&n&o&p\end{pmatrix}\begin{pmatrix}0&0&1&0\\1&0&0&0\\0&0&0&1\\0&1&0&0\end{pmatrix}=\begin{pmatrix}f&h&e&g\\n&p&m&o\\b&d&a&c\\j&l&i&k\end{pmatrix}.$$

Es de notar que esta permutaciones mantienen los elementos en la diagonal. La diagonal de $\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}$ es un reordenamiento de la diagonal de $\mathbf{A}$.

Si $\mathbf{A}$ es positiva definida, entonces $\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}$ es además positiva definida para cualquier permutación de la matriz $\mathbf{P}$.

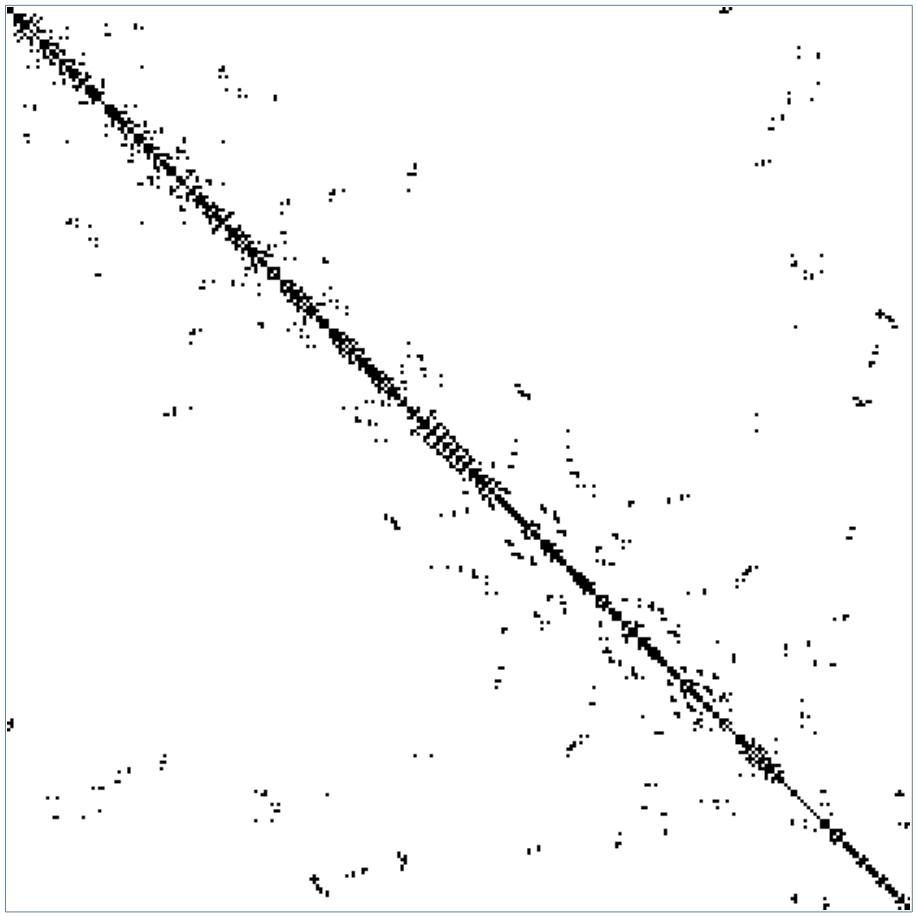
Además, $\left(\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}\right)$ tiene los mismos eigenvalores, determinante y traza que $\mathbf{A}$. Esto es estudiado en la teoría espectral de grafos.

Vamos entonces a resolver el sistema reordenado

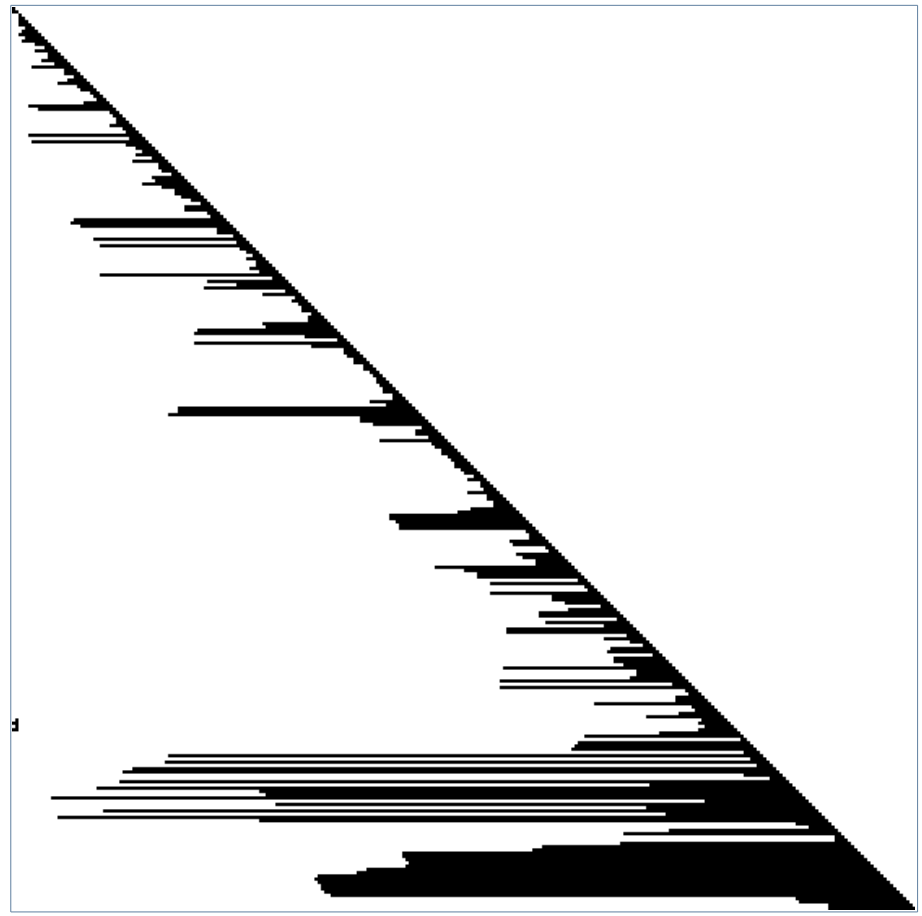
$$\left(\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}\right)\left(\mathbf{P}\,\mathbf{x}\right)=\left(\mathbf{P}\,\mathbf{b}\right).$$

Ahora veamos cuál es el beneficio de reordenar. Recordemos la notación $\eta(\mathbf{L})$, que indica el número de elementos no cero de $\mathbf{L}$.

Buscamos reordenar los renglones y las columnas de $\mathbf{A}$, de tal forma que se reduzca el número de entradas no cero de la matriz factor $\mathbf{L}$.



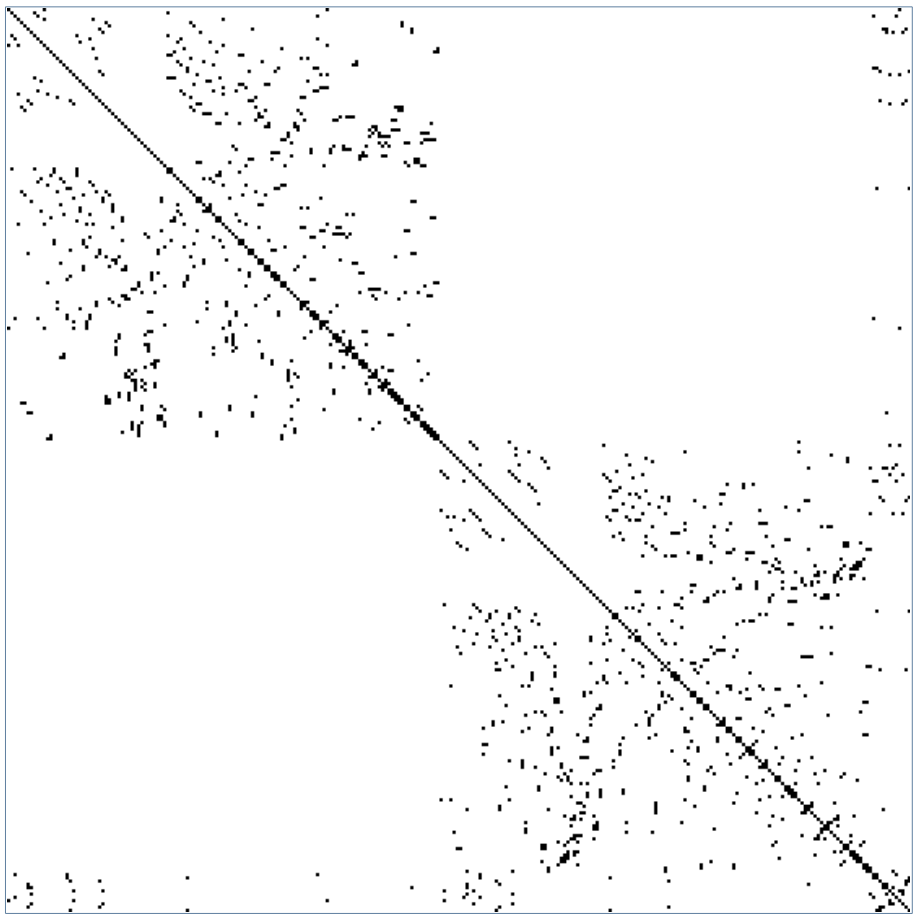
$\mathbf{A}$



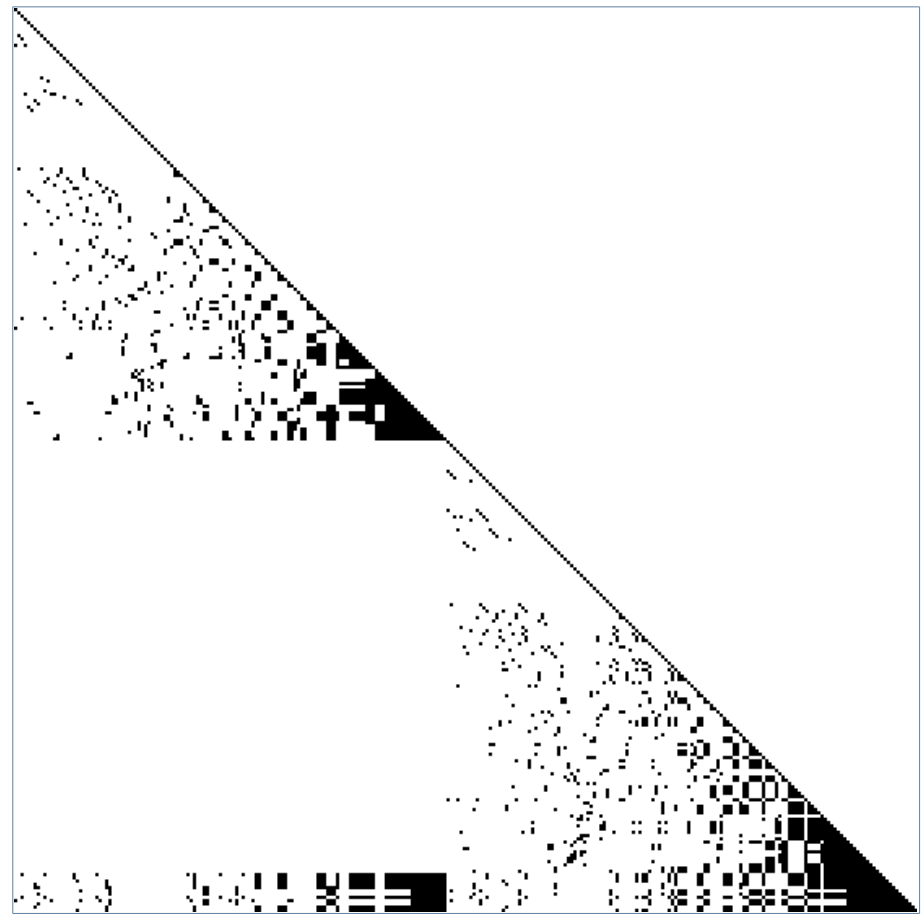
$\mathbf{L}$

En este ejemplo la matriz $\mathbf{A}$ es de tamaño 556×556, con $\eta(\mathbf{A})=1810$, la matriz $\mathbf{L}$, con $\eta(\mathbf{L})=8729$, es el resultado de la factorización Cholesky de $\mathbf{A}$.

Ahora, con un cierto reordenamiento, tenemos que la matriz de rigidez $\mathbf{A}'$ con $\eta(\mathbf{A}')=1810$ (la misma cantidad de elementos no nulos que $\mathbf{A}$) y su factorización $\mathbf{L}'$ tiene $\eta(\mathbf{L}')=3215$.



$\mathbf{A}'$



$\mathbf{L}'$

Es decir reducimos el número de entradas no cero de la factorización en

$$\frac{\eta(\mathbf{L}')=3215}{\eta(\mathbf{L})=8729}=0.368.$$

Consideraciones

La elección del reordenamiento (o permutación **P**) tendrá un efecto determinante en el tamaño de las entradas no cero de **L**.

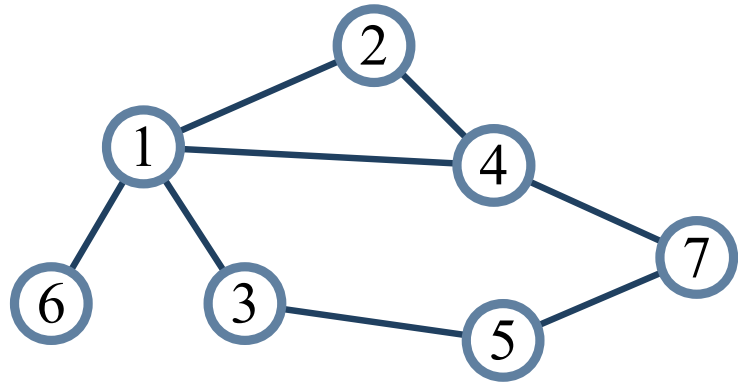
Calcular un “buen” reordenamiento de la matriz **A** que minimice las entradas no cero de **L** es un problema NP completo [Yann81].

Existen heurísticas que generan un reordenamiento aceptable en un tiempo reducido. Estas heurísticas están basadas en operaciones con **grafos**.

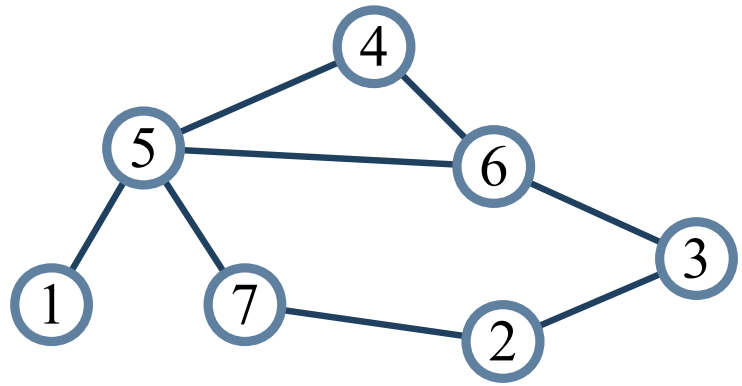
El reordenamiento del grafo de una matriz dispersa es equivalente a aplicar una matriz de permutación.

Ordenamiento de grafos

Sea $G=(V, E)$ un grafo con n vértices V junto con m aristas E,



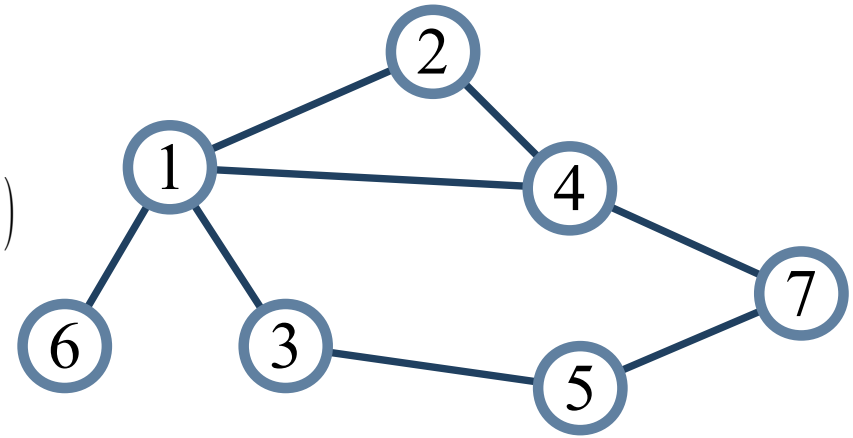
Un **ordenamiento** (o etiquetado) α de G es simplemente un mapeo del conjunto $\{1,2,\ldots,n\}$ en $\mathbf{V}$, donde n denota el número de nodos de G. El grafo ordenado (o etiquetado) por α será denotado como $G^{\alpha}=(V^{\alpha}, E^{\alpha})$.



Sea $\mathbf{A}$ una matriz dispersa, simétrica, de tamaño $n \times n$, el grafo ordenado de $\mathbf{A}$, denotado por $G^{\mathbf{A}} = \left(V^{\mathbf{A}}, E^{\mathbf{A}} \right)$, en el cual los n vértices de $G^{\mathbf{A}}$ están numerados de 1 a n, y $\left\{ v_i, v_j \right\} \in E^{\mathbf{A}}$ si y solo sí $A_{ij} = A_{ji} \neq 0$, $i \neq j$. Aquí v_i denota el nodo de $V^{\mathbf{A}}$ con etiqueta i.

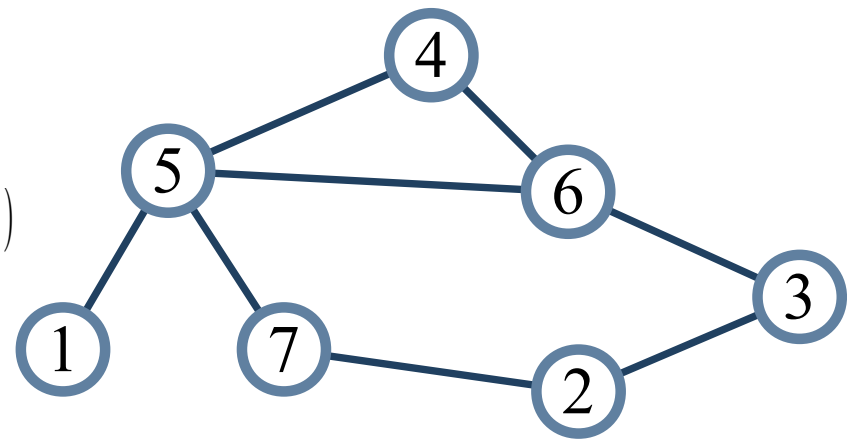
$$\mathbf{A} = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \cdot & \bullet & \cdot \\ \bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \cdot \\ \bullet & \cdot & \bullet & \cdot & \bullet & \cdot & \cdot \\ \bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet & \cdot & \bullet & \cdot & \bullet \\ \bullet & \cdot & \cdot & \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \bullet & \bullet & \cdot & \bullet \end{pmatrix}$$

$G^{\mathbf{A}} = \left(V^{\mathbf{A}}, E^{\mathbf{A}} \right)$



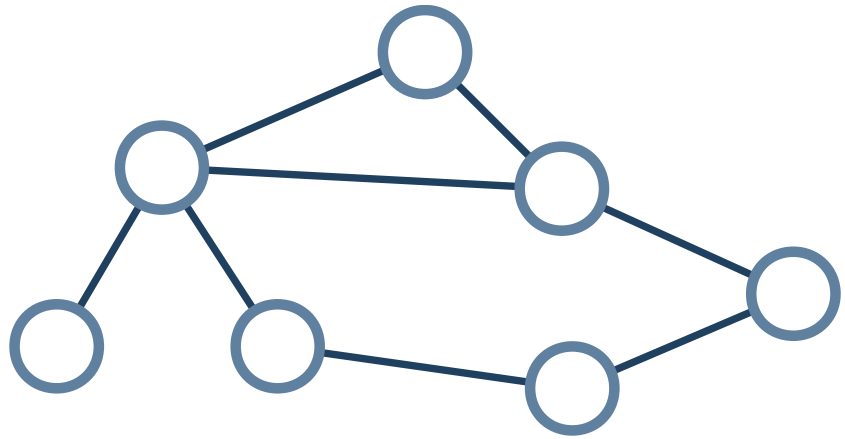
Un ordenamiento α en el grafo $G^{\mathbf{A}} = \left(V^{\mathbf{A}}, E^{\mathbf{A}} \right)$ equivale a un reordenamiento de la matriz.

$G^{\alpha} = \left(V^{\alpha}, E^{\alpha} \right)$



$$\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}} = \begin{pmatrix} \bullet & \cdot & \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \bullet & \cdot & \cdot & \cdot & \bullet \\ \cdot & \bullet & \bullet & \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \bullet & \bullet & \bullet & \cdot \\ \bullet & \cdot & \cdot & \bullet & \bullet & \bullet & \bullet \\ \cdot & \cdot & \bullet & \bullet & \bullet & \bullet & \cdot \\ \cdot & \bullet & \cdot & \cdot & \bullet & \cdot & \bullet \end{pmatrix}$$

Para cualquier matriz de permutación $\mathbf{P} \neq \mathbf{I}$, los grafos no ordenados (o etiquetados) de $\mathbf{A}$ y $\mathbf{P}\,\mathbf{A}\,\mathbf{P}^T$ son los mismos pero su etiquetado asociado es diferente.



Así, un grafo no etiquetado de $\mathbf{A}$ representa la estructura de $\mathbf{A}$ sin sugerir un orden en particular. Esta representa la equivalencia de las clases de matrices $\mathbf{P}\,\mathbf{A}\,\mathbf{P}^T$.

Regresando a nuestro problema con $\mathbf{L}$. Encontrar una "buena" permutación $\mathbf{P}$ de $\mathbf{A}$ que minimize el número de entradas no-cero en $\mathbf{L}$ equivale a encontrar un "buen" ordenamiento de su grafo [Geor81].

Vamos ahora a ver un par de estrategias basadas en manipulación de grafos que permiten minimizar el número de entradas no-cero en $\mathbf{L}$.

Adyacencia

Dos nodos $u, v \in V$ en un grafo $G=(V, E)$ son adyacentes si $\{u, v\} \in E$.

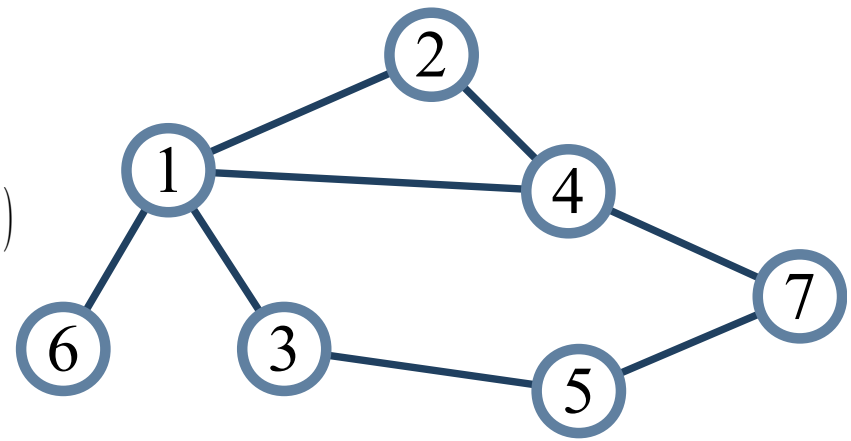
Para $v \in V$, el **conjunto adyacente** de u, denotado $\text{adj}(\{u\})$ ó $\text{adj}(u)$, es

$$\text{adj}(u) = \{v \in V \setminus \{u\} \mid \{u, v\} \in E\}.$$

Por ejemplo: $\text{adj}(4) = \{1, 2, 7\}$, $\text{adj}(6) = \{1\}$.

$$\mathbf{A} = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \cdot & \bullet & \cdot \\ \bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \cdot \\ \bullet & \cdot & \bullet & \cdot & \bullet & \cdot & \cdot \\ \bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet & \cdot & \bullet & \cdot & \bullet \\ \bullet & \cdot & \cdot & \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \bullet & \bullet & \cdot & \bullet \end{pmatrix}$$

$G=(V, E)$



Las adyacencias de un nodo equivalen a las entradas no-cero de un renglón fuera de la diagonal.

Para $U \subset V$, el **conjunto adyacente** de U, denotado como $\text{adj}(U)$, es

$$\text{adj}(U) = \{v \in V \setminus U \mid \{u, v\} \in E \text{ para algún } u \in U\}.$$

Por ejemplo: $\text{adj}(\{5, 7\}) = \{3, 4\}$.

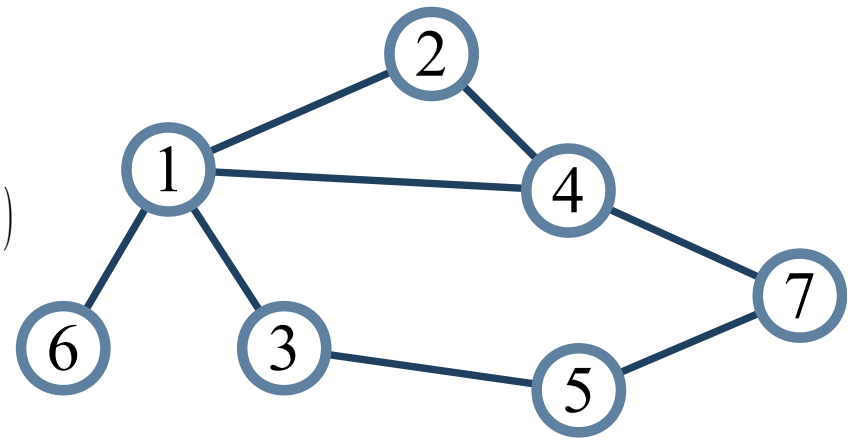
Grado

Para $\boldsymbol{U} \subset \boldsymbol{V}$, el **grado** de $\boldsymbol{U}$, denotado por $\mathrm{gr}(\boldsymbol{U})$, es simplemente el número $\left|\mathrm{adj}(\boldsymbol{U})\right|$, donde $|S|$ denota el número de miembros del conjunto S.

En el caso de que se trate de un solo elemento, consideraremos $\mathrm{gr}(\{u\}) \equiv \mathrm{gr}(u)$.

$$\mathbf{A} = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \cdot & \bullet & \cdot \\ \bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \cdot \\ \bullet & \cdot & \bullet & \cdot & \bullet & \cdot & \cdot \\ \bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet & \cdot & \bullet & \cdot & \bullet \\ \bullet & \cdot & \cdot & \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \bullet & \bullet & \cdot & \bullet \end{pmatrix}$$

$G = (\boldsymbol{V}, \boldsymbol{E})$



Por ejemplo:

$$\mathrm{adj}(4) = \{1, 2, 7\},\ \mathrm{gr}(4) = 3.$$

Por ejemplo:

$$\mathrm{adj}(\{5, 7\}) = \{3, 4\},\ \mathrm{gr}(\{5, 7\}) = 2.$$

El grado de un nodo equivale al número de entradas de un renglón fuera de la diagonal.

Algoritmo de grado mínimo

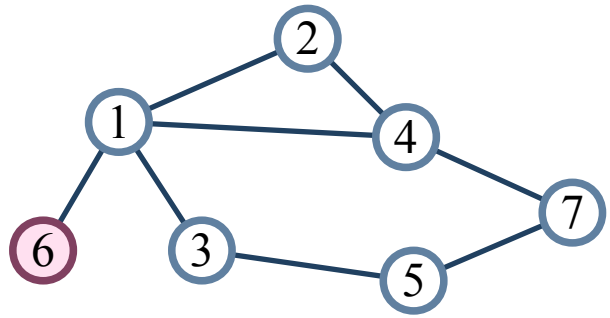
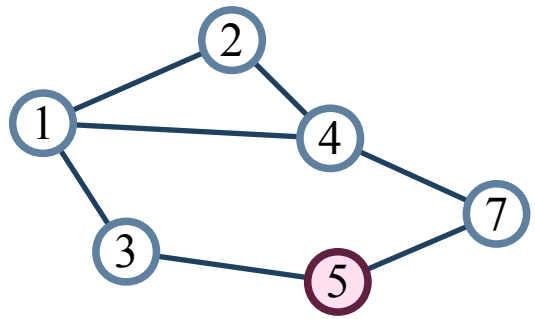
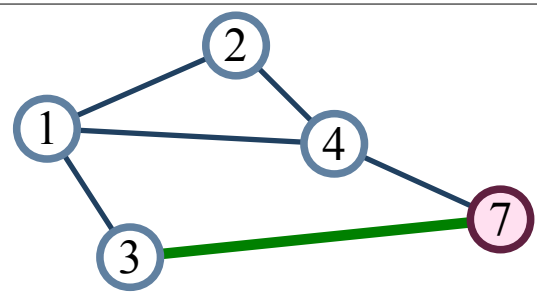
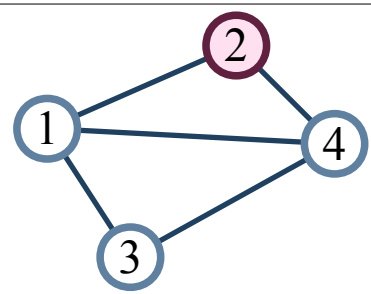
La heurística más común utilizada para realizar el ordenamiento de un grafo es el **algoritmo de grado mínimo**.

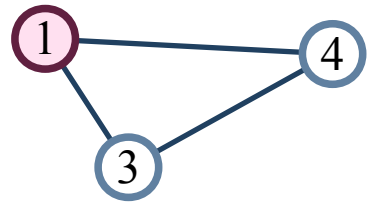
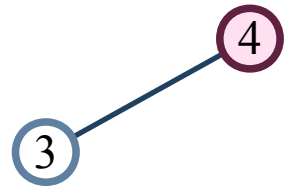
La versión básica de éste es [Geor81 p116]:

Dada un matriz $\mathbf{A}$ y su correspondiente grafo G_0
$i \leftarrow 1$
repetir
  En el grafo $G_{i-1}\left(\boldsymbol{V}_{i-1}, \boldsymbol{E}_{i-1}\right)$, elegir el vértice v_i que tenga el grado mínimo.
  Formar el grafo de eliminación $G_i\left(\boldsymbol{V}_i, \boldsymbol{E}_i\right)$ como sigue:
    Eliminar el vértice v_i de G_{i-1} y sus aristas inicidentes
    Agregar aristas al grafo tal que los nodos $\mathrm{adj}\left(v_1\right)$ sean pares adyacentes en G_i.
  $i \leftarrow i+1$
mientras $i<\left|\boldsymbol{V}\right|$

Cuando el grado mínimo se presenta en varios vértices, se elige uno de forma arbitraria.

A continuación un ejemplo de una secuecia dada por el algoritmo de grado mínimo:

i	**Grafo de eliminación** G_{i-1}	x_i	$\mathrm{gr}(x_i)$
1		6	1
2		5	2
3		7	2
4		2	2

i	**Grafo de eliminación** G_{i-1}	x_i	$\mathrm{gr}(x_i)$
5		1	2
6		4	1
7		3	0

La secuencia de eliminación es entonces 6, 5, 7, 2, 1, 4, 3. Éste es el nuevo orden para generar la matriz **A**', lo que equivale a la matriz de permutación

$$\mathbf{P}=\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Versiones más avanzadas de este algoritmo pueden consultarse en [Geor89].

Algoritmo de disección anidada

Ahora vamos a revisar brevemente el método de disección anidada, el cual funcione muy bien para para matrices resultantes de problemas de diferencias finitas y elemento finito.

La principal ventaja de este algoritmo comparado con el de grado mínimo es la velocidad y el poder predecir las necesidades de almacenamiento.

La ordenación producida es similar a la del algoritmo de grado mínimo.

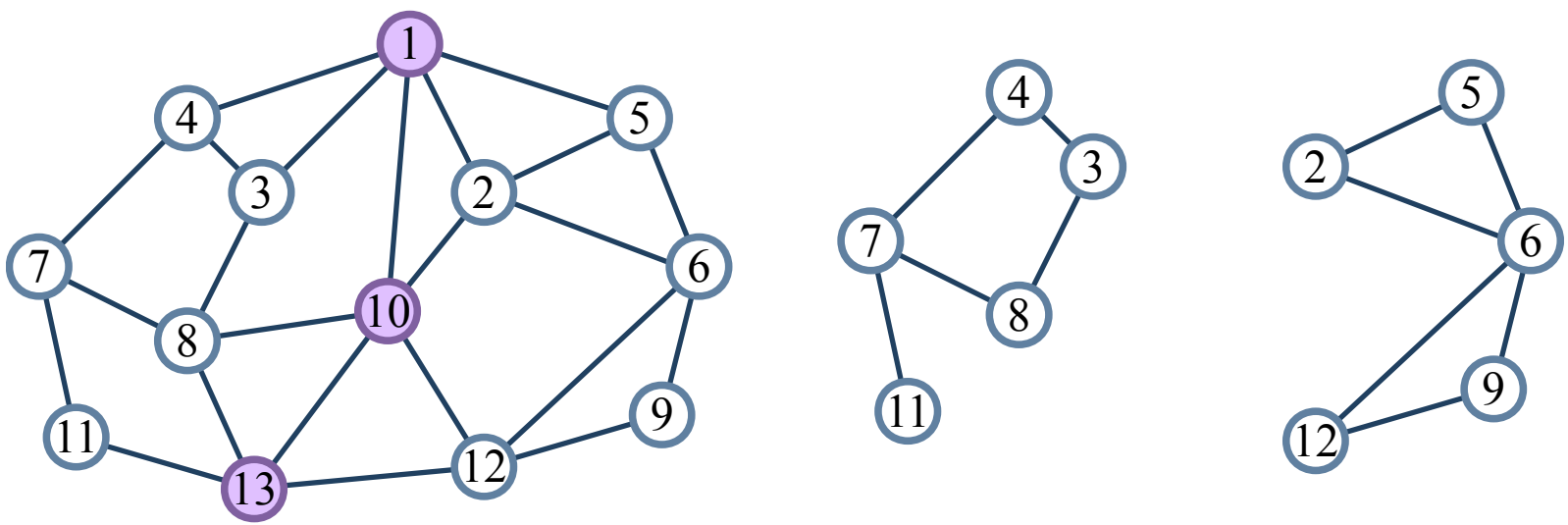
Separador

Consideremos un conjunto separador $\boldsymbol{S} \subset \boldsymbol{V}$ en $G=(\boldsymbol{V},\boldsymbol{E})$, cuya remoción de G desconecta el grafo en dos conjuntos de vértices $\boldsymbol{R} \subset \boldsymbol{V}$ y $\boldsymbol{S} \subset \boldsymbol{V}$ no vacios, tal que

$$\boldsymbol{R} \cap \boldsymbol{S} = \emptyset,\ \boldsymbol{S} \cap \boldsymbol{T} = \emptyset,\ \boldsymbol{R} \cap \boldsymbol{T} = \emptyset \text{ y } \boldsymbol{R} \cup \boldsymbol{S} \cup \boldsymbol{T} = \boldsymbol{V}$$

En el siguiente ejemplo $\boldsymbol{V}=\{1,2,\ldots,13\}$, sea $\boldsymbol{S}=\{1,10,13\}$ un separador de $G=(\boldsymbol{V},\boldsymbol{E})$, el cual, al ser removido del grafo obtenemos

$$\boldsymbol{R}=\{4,3,7,8,11\} \text{ y } \boldsymbol{T}=\{2,5,6,9,12\}$$



Esta procedimiento aplicado de forma de forma recursiva se conoce como algoritmo de disección anidada generalizado [Lipt79].

La idea es tratar de dividir el grafo buscando que $\boldsymbol{R}$ y $\boldsymbol{T}$ sean de igual tamaño con un separador pequeño. Este es un algoritmo recursivo que emplea la estrategia de divide y vencerás, a continuación describimos una versión simplificada del algoritmo.

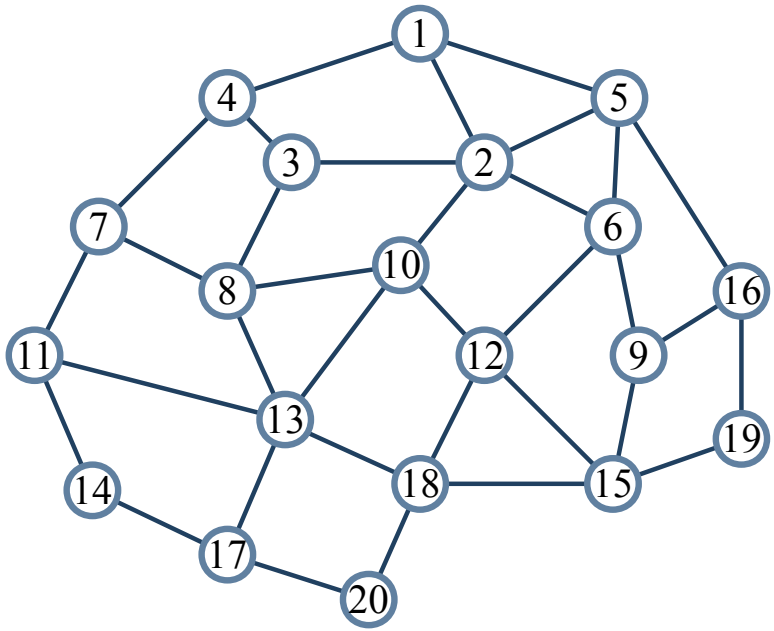
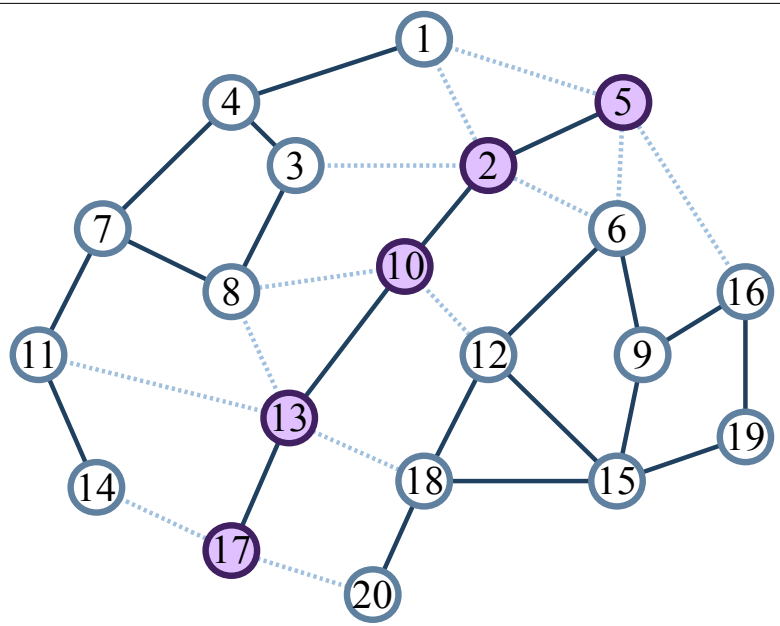
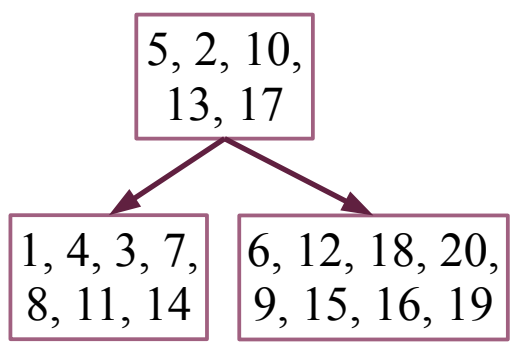
Disección $G(\boldsymbol{V}, \boldsymbol{E})$
Si $|G|<\beta$
- Los nodos de $\boldsymbol{V}$ son ordenados arbitrariamente (puede utilizarse el algoritmo de grado mínimo)

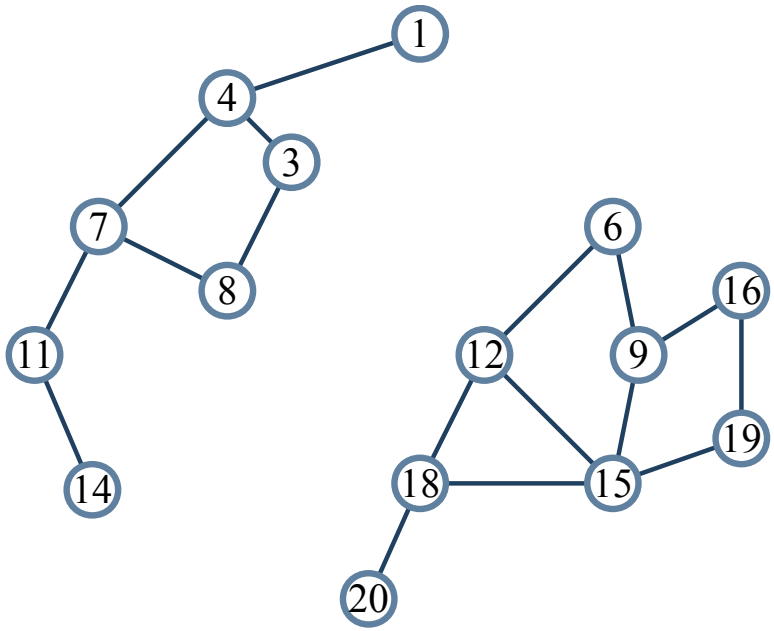
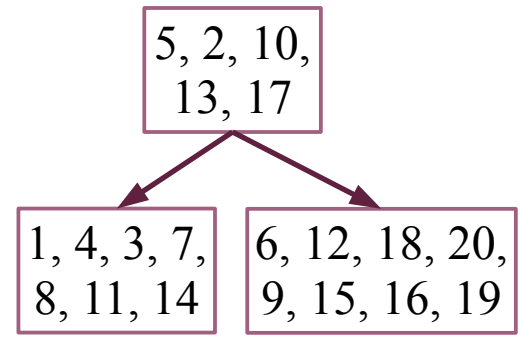
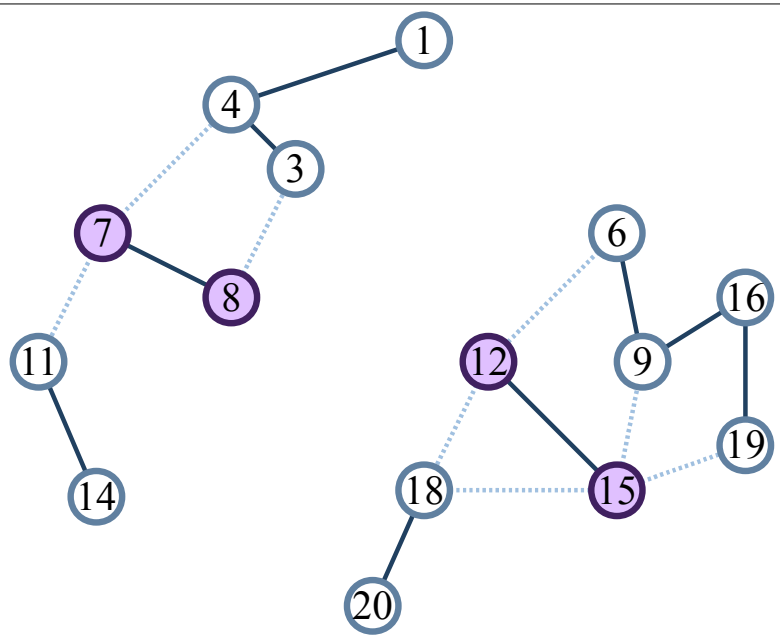
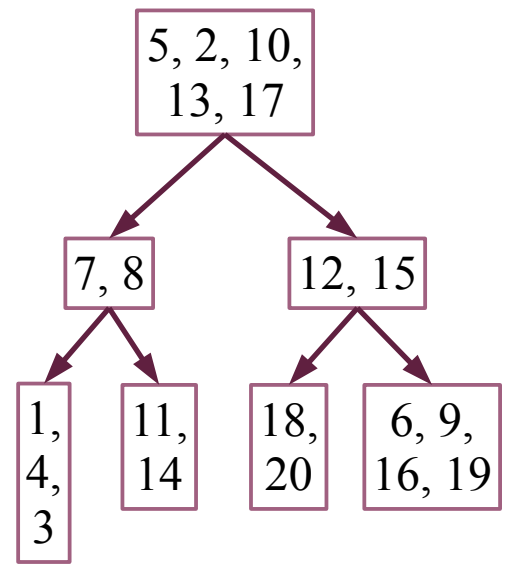
si_no
- Buscar un separador $\boldsymbol{S} \subset \boldsymbol{V}$ tal que al removerlo de G queden dos conjuntos de vértices disconexos $\boldsymbol{R}$ y $\boldsymbol{T}$ tal que $|\boldsymbol{R}| \approx |\boldsymbol{T}|$.
- Eliminar de $\boldsymbol{E}$ todas las aristas con conexiones con $\boldsymbol{S}$.
- Disección $G(\boldsymbol{R}, \boldsymbol{E})$
- Disección $G(\boldsymbol{T}, \boldsymbol{E})$
- Ordenar los nodos de $\boldsymbol{S}$ con los números más grandes.

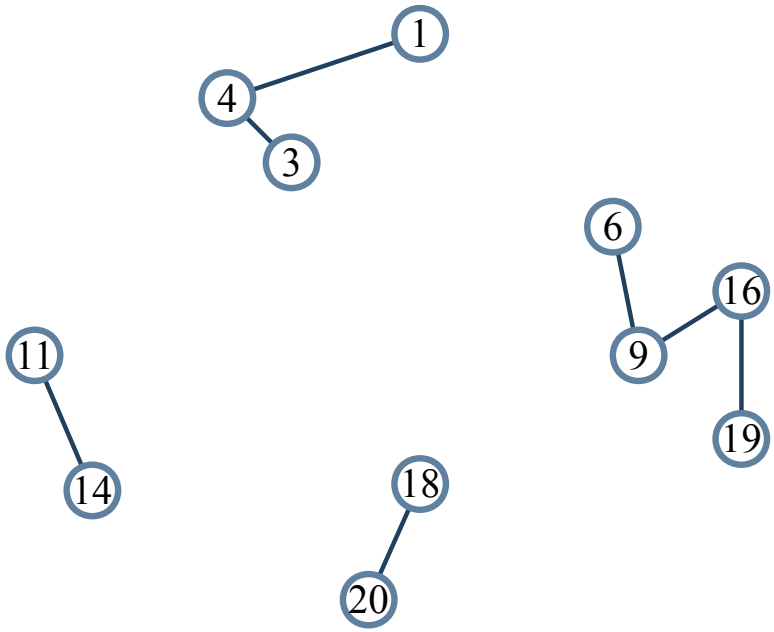
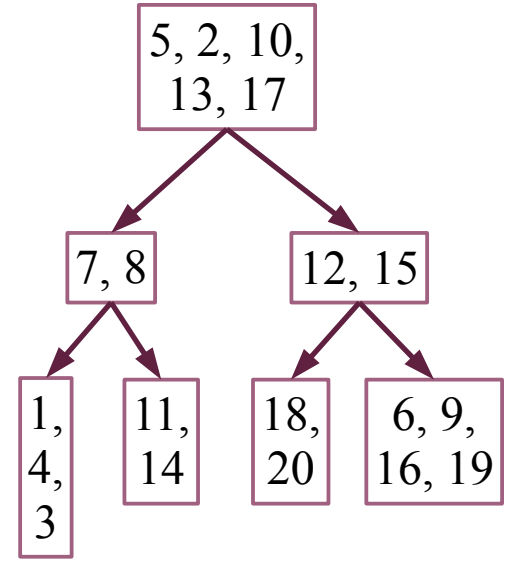
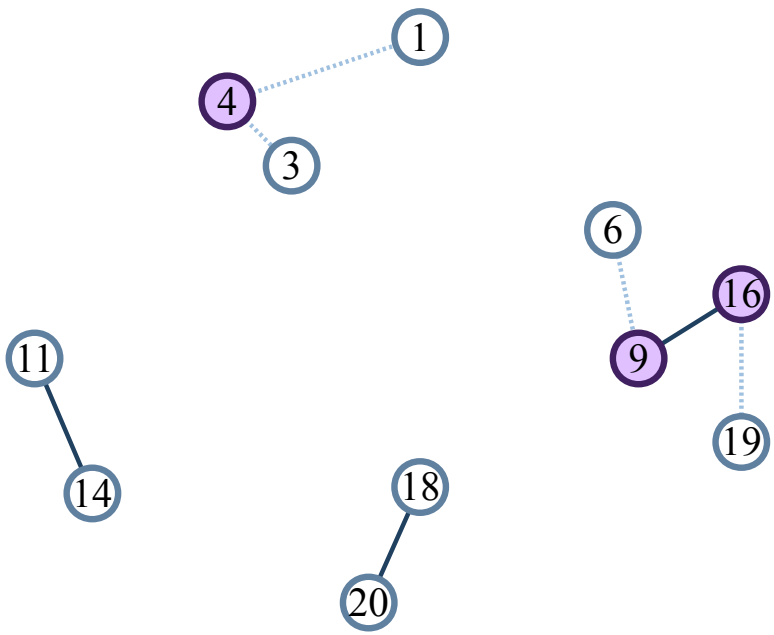
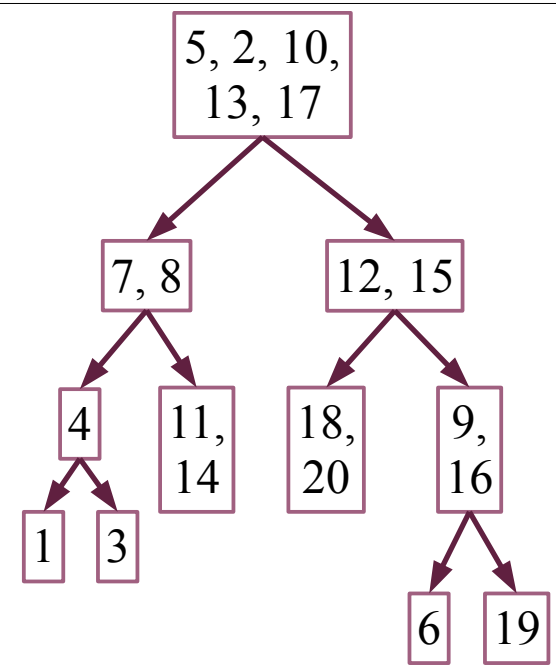
El algoritmo es recursivo y numera los nodos de G tal que la eliminación gaussiana rala (factorización Cholesky rala) es eficiente.

Una versión mejorada del algoritmo se puede consultar en [Kary99].

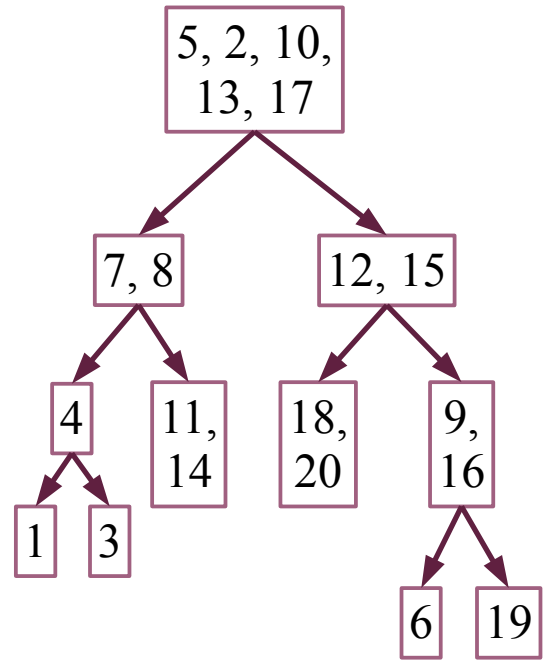
A continuación un ejemplo de una secuecia dada por el algoritmo de disección anidada:

Grafo	Árbol de dependenicas
	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
	

Grafo	Árbol de dependenicas
	
	

Grafo	Árbol de dependencias
	
	

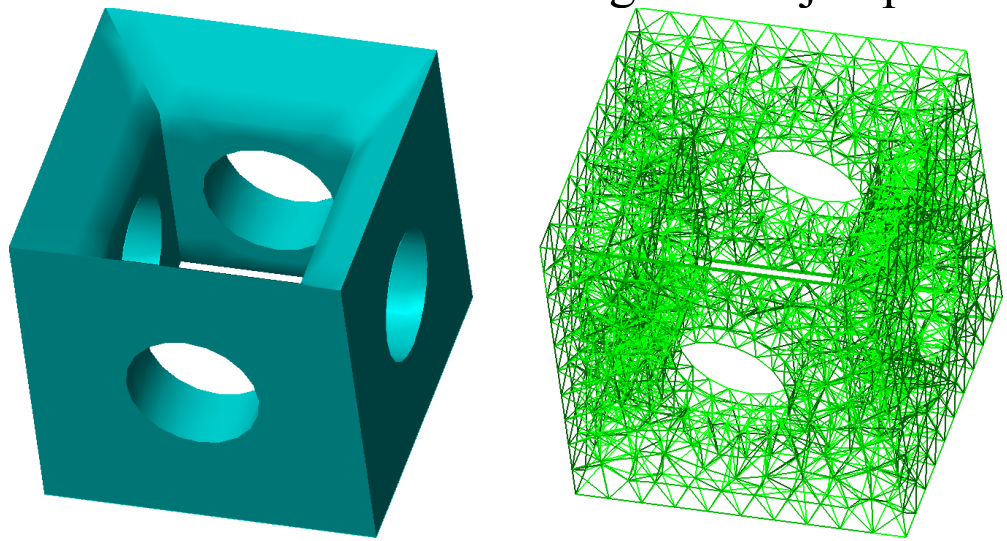
A partir del árbol de dependencias final se crea la secuencia para la renumeración de los nodos, enumerando de las hojas hasta la raíz de forma recursiva.



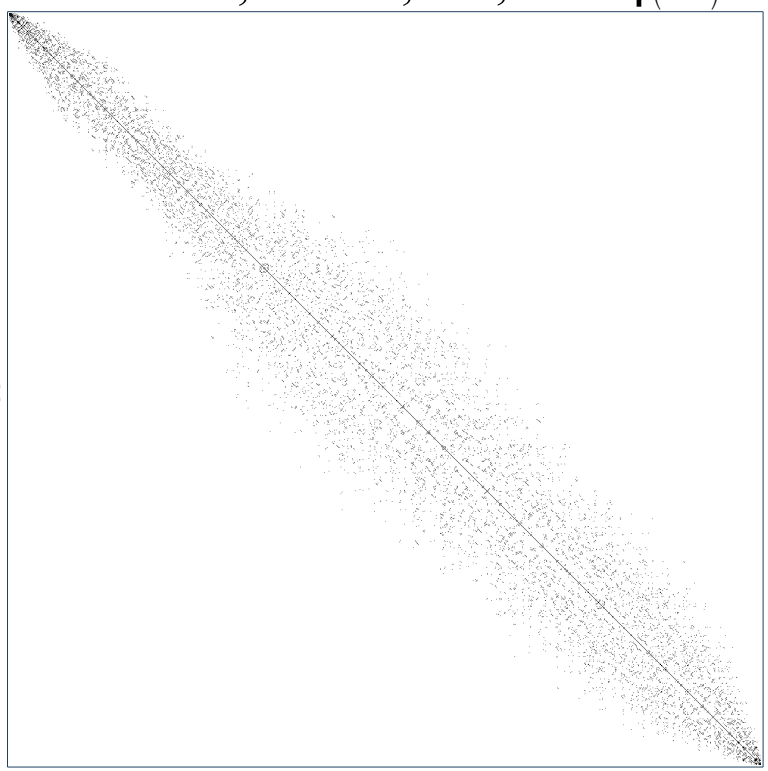
La secuencia de eliminación es entonces 6, 19, 9, 16, 18, 20, 12, 15, 1, 3, 4, 11, 14, 7, 8, 5, 2, 10, 13, 17. Éste es el nuevo orden para generar la matriz **A**', lo que equivale a **P** igua a

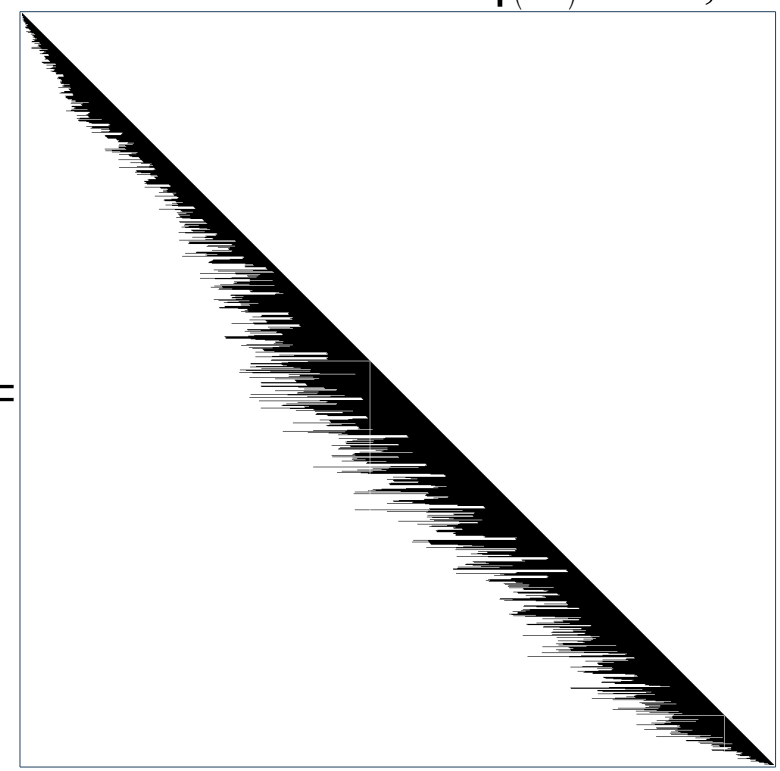
$$\mathbf{P}=\begin{pmatrix}
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0
\end{pmatrix}.$$

Este algoritmo produce reordenamientos como en el siguiente ejemplo de elemento finito

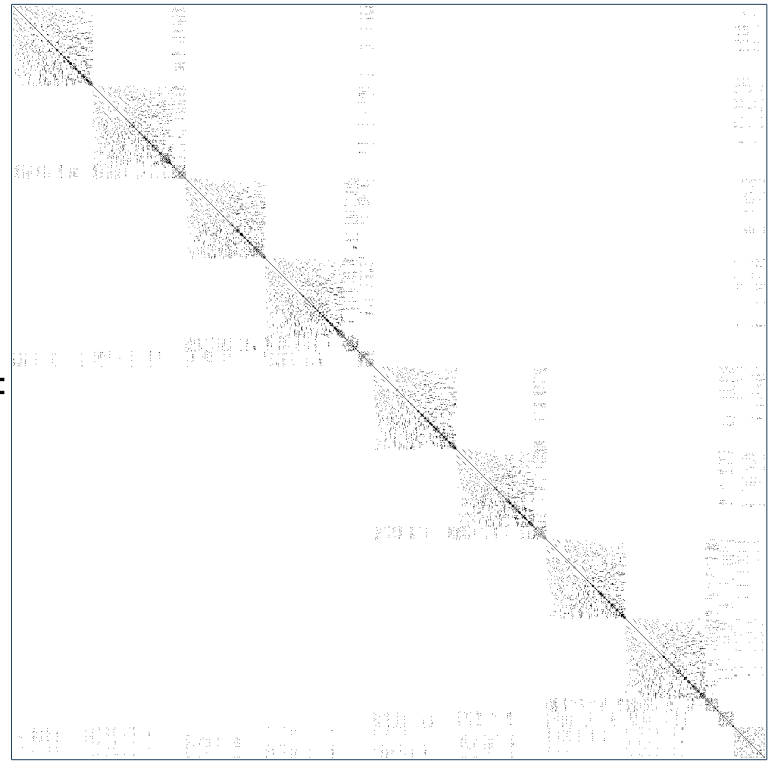


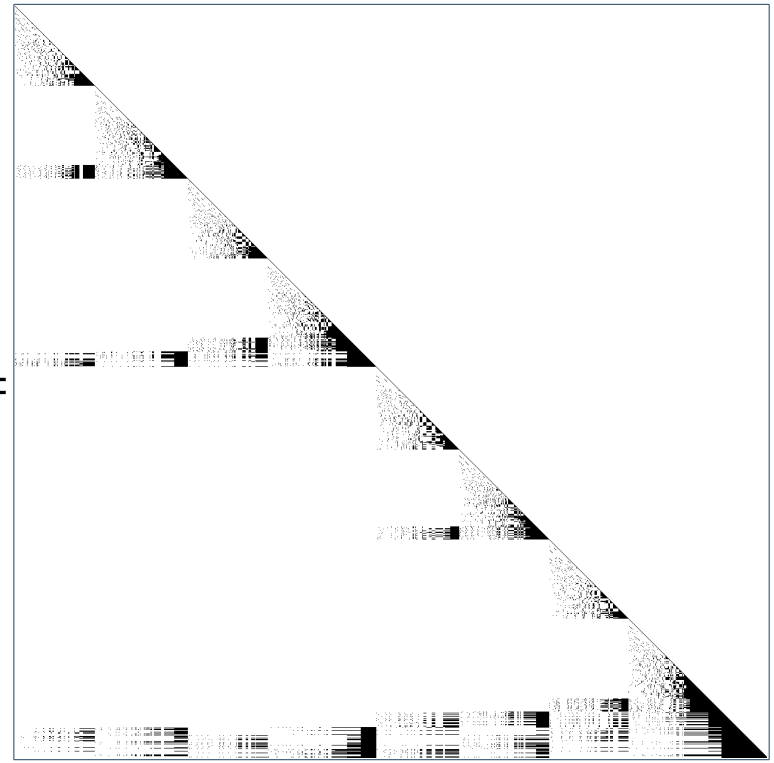
Se tiene $\mathbf{A}$ de tamaño $1{,}263 \times 1{,}263$, con $\eta(\mathbf{A}) = 14{,}131$. La factorización da $\eta(\mathbf{L}) = 128{,}476$

$\mathbf{A} =$ 

$\mathbf{L} =$ 

El reordenamiento utilizando disección anidada es

$\mathbf{A}^{\mathrm{r}} =$ 

$\mathbf{L}^{\mathrm{r}} =$ 

La factorización produce $\eta\left(\mathbf{L}^{\mathrm{r}}\right) = 39{,}465$. Es decir

$$\frac{\eta\left(\mathbf{L}^{\mathrm{r}}\right) = 39{,}465}{\eta\left(\mathbf{L}\right) = 128{,}476} = 0.3072.$$

Usando la librería METIS para reordenar matrices

Es una librería para particionar grafos y además tiene rutinas para reordenar matrices:
http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

Vamos a utlizar la función **METIS_NodeND**.

void **METIS_NodeND**(int* *n*, idxtype* *xadj*, idxtype* *adjncy*, int* *numflag*, int* *options*, idxtype* *perm*, idxtype* *iperm*)

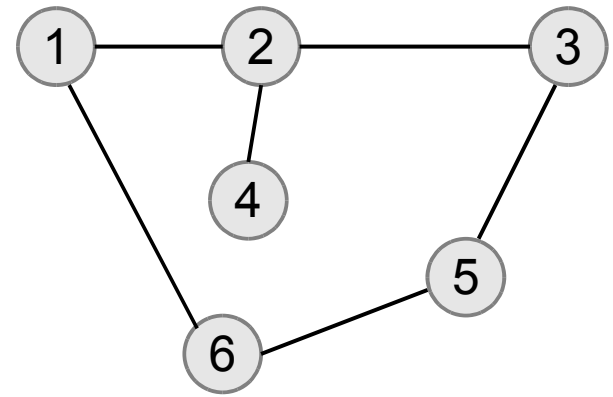
Esta funcion calcula un reordenamiento que reduce el fill-in de la factorización de una matriz rala, utiliza el algoritmo multinivel de disección anidada.

n	Number of nodes in the graph
xadj, adjncy	The adjacency structure of the graph.
numflag	Used to indicate which numbering scheme is used for the adjacency structure of the graph. numflag can take the following two values: 0 C-style numbering is assumed that starts from 0 1 Fortran-style numbering is assumed that starts from 1
options	This is an array of 8 integers that is used to pass parameters for the various phases of the algorithm. If options[0]=0 then default values are used.
perm, iperm	These are vectors, each of size n. Upon successful completion, they store the fill-reducing permutation and inverse-permutation.

Ejemplo

	1	2	3	4	5	6
1	*	*				*
2	*	*	*	*		
3		*	*		*	
4		*		*		
5			*		*	*
6	*				*	*

Matriz A



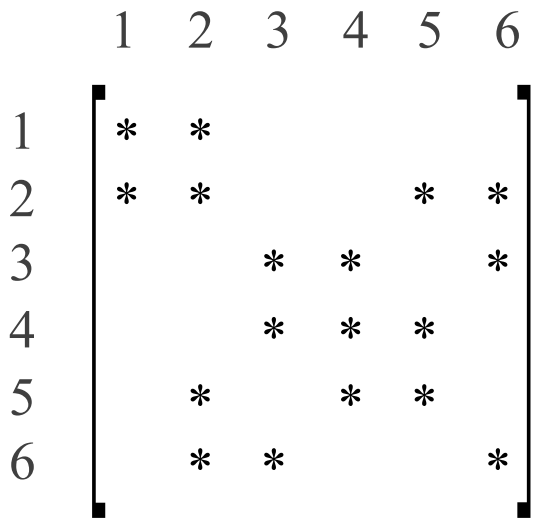
Grafo G^A

```
int n = 6;
int xadj[7] = {1, 3, 6, 8, 9, 11, 13};
int adjncy[12] = {2, 6, 1, 3, 4, 2, 5, 2, 3, 6, 1, 5};
int numflag = 1;
int options[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int perm[6];
int iperm[6];
METIS_NodeND(&n, xadj, adjncy, &numflag, options, perm, iperm);
```

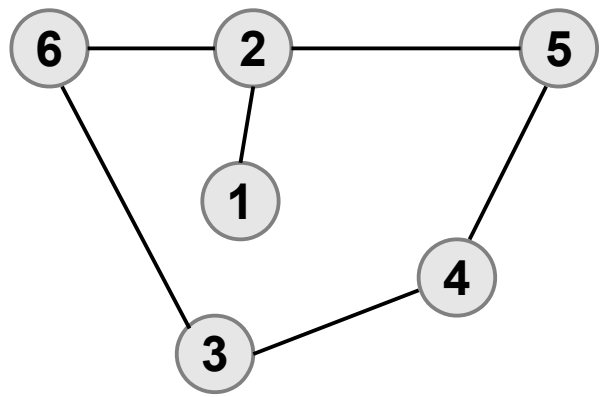
Resultado

perm = {4, 2, 6, 5, 3, 1}

iperm = {6, 2, 5, 1, 4, 3}



Matriz $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}$



Grafo $G^{\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{P}^{\mathrm{T}}}$

Vamos a utilizar *perm* e *iperm* para reordenar la matriz $\boldsymbol{A}$ y el vector $\boldsymbol{b}$.

Reordenamiento

Para reordenar la matriz rala $\mathbf{A}$ a partir de *perm* e *iperm*, para obtener $\mathbf{A}^{\mathrm{r}} = \mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}$, podemos utilizar el siguiente algoritmo:

para $i \leftarrow 1 \ldots n$
  $j \leftarrow perm_i$
  Reservar espacio en $\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)$, tal que $\left|\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)\right| = \left|\boldsymbol{J}_j\left(\mathbf{A}\right)\right|$
  para $k \leftarrow 1, 2 \ldots \left|\boldsymbol{J}_j\left(\mathbf{A}\right)\right|$
    $l \leftarrow \boldsymbol{J}_j^k\left(\mathbf{A}\right)$
    $m \leftarrow iperm_l$
    $\boldsymbol{J}_i^k\left(\mathbf{A}^{\mathrm{r}}\right) \leftarrow m$
    $\boldsymbol{V}_i^k\left(\mathbf{A}^{\mathrm{r}}\right) \leftarrow \boldsymbol{V}_j^k\left(\mathbf{A}\right)$
  fin_para
  Los índices de $\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)$ no estarán en orden ascendente, reordenar $\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)$ y $\boldsymbol{V}_i\left(\mathbf{A}^{\mathrm{r}}\right)$.
fin_para

Hay que reordenar $\boldsymbol{b}$ para obtener un vector $\boldsymbol{b}^{\mathrm{r}} = \boldsymbol{P}\,\boldsymbol{b}$.

para $i \leftarrow 1, 2 \ldots n$
    $j \leftarrow perm_i$
    $b_i^{\mathrm{r}} \leftarrow b_j$
fin_para

Podemos entonces resolver el sistema

$$\mathbf{A}^{\mathrm{r}}\,\mathbf{x}^{\mathrm{r}} = \mathbf{b}^{\mathrm{r}}.$$

Finalmente hay que reordenar $\boldsymbol{x}^{\mathrm{r}}$ con la permutación inversa para obtener $\mathbf{x} = \mathbf{P}^{\mathrm{T}}\,\mathbf{x}^{\mathrm{r}}$.

para $i \leftarrow 1, 2 \ldots n$
    $j \leftarrow iperm_i$
    $x_i \leftarrow x_j^{\mathrm{r}}$
fin_para

En Linux/Unix es más simple si utilizan el manejador de paquetes de su distribución (synamptic, apt-get, etc).

La librería METIS está diseñada para trabajar con C, para utilizarla con C++ es neceario usar:

```
extern "C"
{
    #include <metis.h>
}
```

Para compilar:

```
g++ -o ejecutable codigo.cpp -lmetis
```

Código de ejemplo completo

```
extern "C"
{
   #include <metis.h>
}
#include <stdio.h>

int main(void) {
   int n = 6;
   int xadj[7] = {1, 3, 6, 8, 9, 11, 13};
   int adjncy[12] = {2, 6, 1, 3, 4, 2, 5, 2, 3, 6, 1, 5};
   int numflag = 1;
   int options[8] = {0, 0, 0, 0, 0, 0, 0, 0};
   int perm[6];
   int iperm[6];
   METIS_NodeND(&n, xadj, adjncy, &numflag, options, perm, iperm);
   printf("node perm iperm\n");
   for (int i = 0; i < n; ++i)
      printf("   %i    %i     %i\n", i + 1, perm[i], iperm[i]);
   return 0;
}
```

node	perm	iperm
1	4	2
2	1	5
3	5	4
4	3	1
5	2	3
6	6	6

Como instalar METIS desde el código fuente

Bajar el código fuente de METIS de:
http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/OLD/metis-4.0.3.tar.gz

Ir al directorio donde se bajó METIS y ejecutar:

```
tar xzf metis-4.0.3.tar.gz
cd metis-4.0.3
make
mkdir -p ~/metis
cp Lib/*.h ~/metis/.
cp libmetis.a ~/metis/.
```

Esto compila e instala el METIS en su directorio *home*.

Para compilar el ejemplo anterior hay que usar:

```
g++ -o test -I ~/metis test.cpp -L ~/metis -lmetis
```

Los parámetos extra son:

-I ~/metis	Dónde buscar los headers de metis
-L ~/metis	Dónde buscar la librería de metis
-lmetis	Enlazar la librería metis con el código

¿Preguntas?

miguelvargas@cimat.mx

Referencias

[Geor81] A. George, J. W. H. Liu. Computer solution of large sparse positive definite systems. Prentice-Hall, 1981.

[Geor89] A. George, J. W. H. Liu. The evolution of the minimum degree ordering algorithm. SIAM Review Vol 31-1, pp 1-19, 1989.

[Golu96] G. H. Golub, C. F. Van Loan. Matrix Computations. Third edidion. The Johns Hopkins University Press, 1996.

[Heat91] M T. Heath, E. Ng, B. W. Peyton. Parallel Algorithms for Sparse Linear Systems. SIAM Review, Vol. 33, No. 3, pp. 420-460, 1991.

[Kary99] G. Karypis, V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing, Vol. 20-1, pp. 359-392, 1999.

[Lipt79] R. J. Lipton, D. J. Rose, R. E. Tarjan. Generalized Nested Dissection, Computer Science Department, Stanford University, 1997.

[Yann81] M. Yannakakis. Computing the minimum fill-in is NP-complete. SIAM Journal on Algebraic Discrete Methods, Volume 2, Issue 1, pp 77-79, March, 1981.