CIMAT

Heuristics for Nested Dissection to Reduce Fill-in in Sparse Matrix Factorizations

Miguel Vargas-Félix, Salvador Botello-Rionda

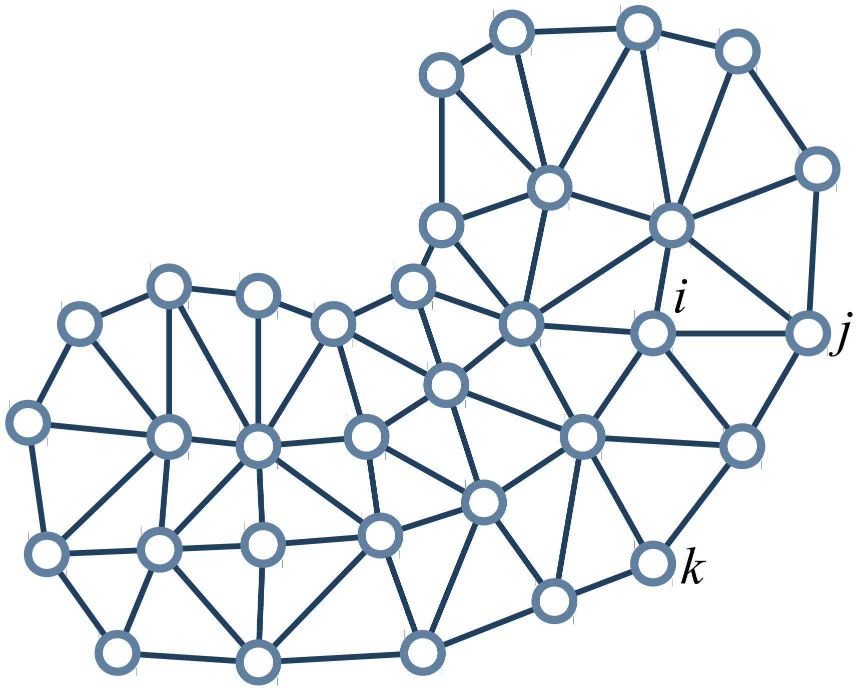
miguelvargas@cimat.mx, botello@cimat.mx

Sparse matrices

In most problems of finite element method, finite volume or isogeometric analysis we have to solve a linear system of equations

$$\mathbf{A}\,\mathbf{x}=\mathbf{b}.$$

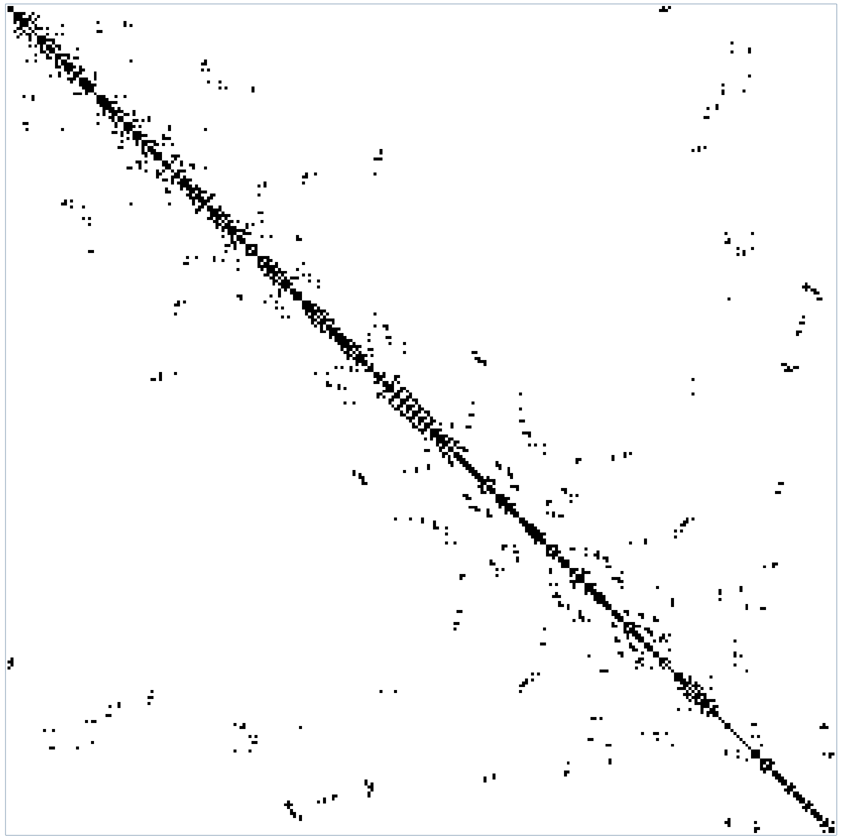
When the stiffness matrix is assembled, the relation between adjacent nodes is captured as entries in a matrix. Because a node has adjacency with only a few others, the resulting matrix has a very sparse structure.



$$\begin{pmatrix}
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & a_{i\,i} & \circ & a_{i\,j} & \circ & 0 & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & a_{j\,i} & \circ & a_{j\,j} & \circ & 0 & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\circ & 0 & \circ & 0 & \circ & a_{k\,k} & \circ & \cdots \\
\circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

Lets define the notation $\eta(\boldsymbol{A})$, it indicates the number of non-zero entries of $\boldsymbol{A}$.

For example, $\boldsymbol{A}\in\mathbb{R}^{556\times556}$ has 309,136 entries, with $\eta(\boldsymbol{A})=1810$, this means that only the 0.58% of the entries are non zero.


Black dots indicates a non zero entry in the matrix

In finite element problems all matrices have symmetric structure, and depending on the problem symmetric values or not.

Cholesky factorization for sparse matrices

For full matrices the computational complexity of Cholesky factorization $\mathbf{A}=\mathbf{L}\,\mathbf{L}^{\mathrm{T}}$ is $\mathrm{O}\left(n^3\right)$.

To calculate entries of $\mathbf{L}$

$$L_{ij}=\frac{1}{L_{jj}}\left(A_{ij}-\sum_{k=1}^{j-1} L_{ik}\,L_{jk}\right),\ \text{for } i>j$$

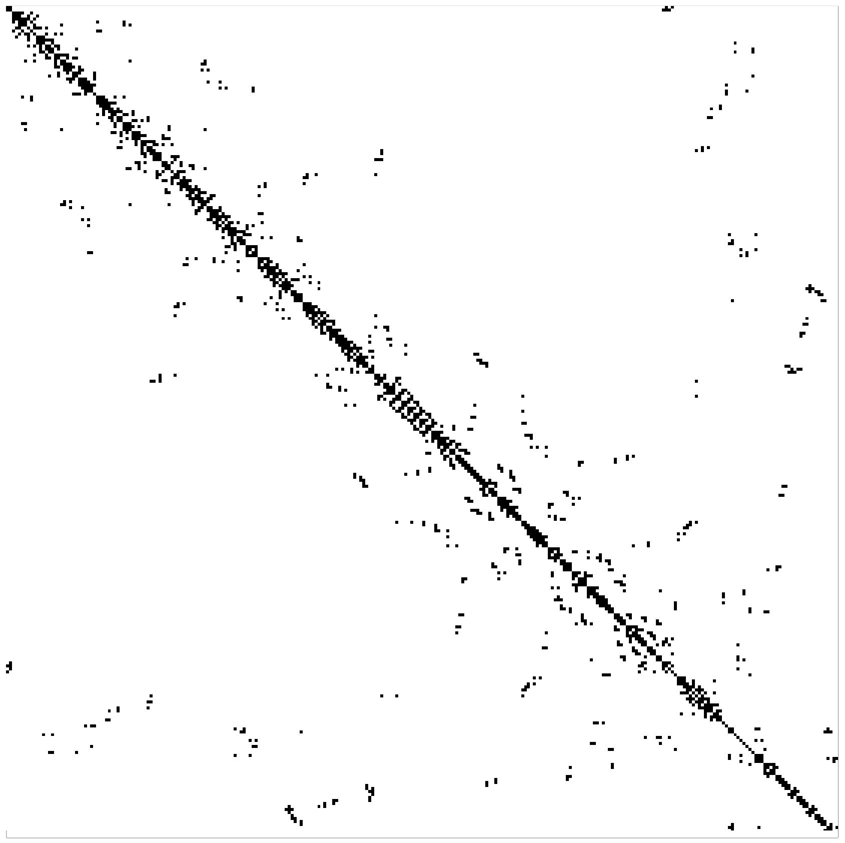
$$L_{jj}=\sqrt{A_{jj}-\sum_{k=1}^{j-1} L_{jk}^2}.$$

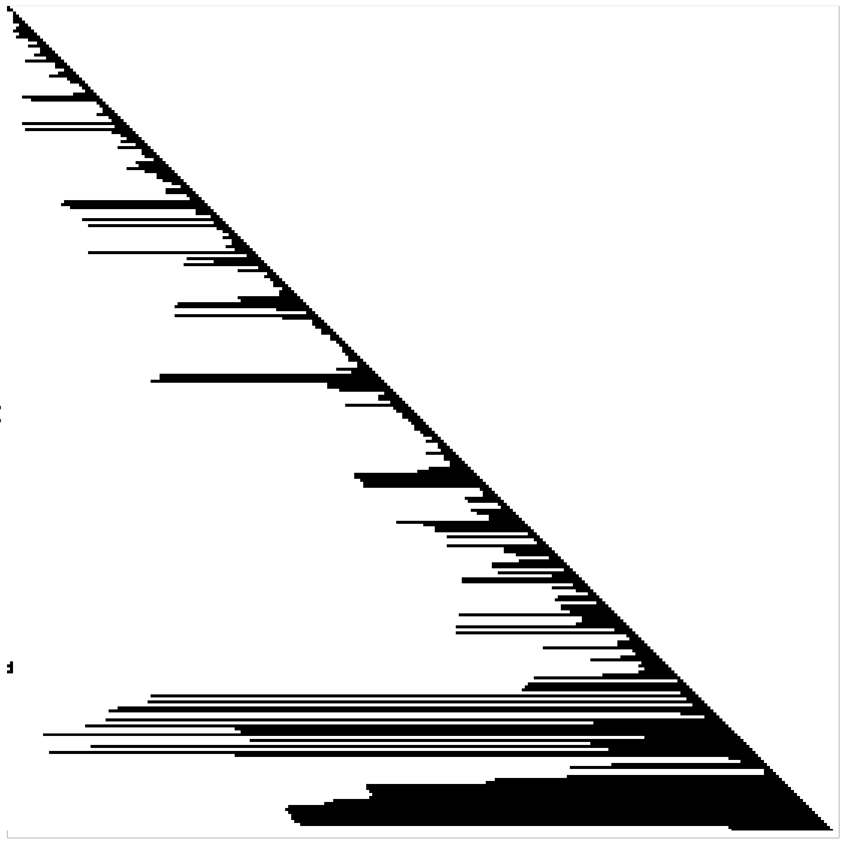
We use four strategies to reduce time and memory usage when performing this factorization on sparse matrices:

1. Reordering of rows and columns of the matrix to reduce fill-in in $\mathbf{L}$. This is equivalent to use a permutation matrix to reorder the system $\left(\mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}\right)\left(\mathbf{P}\,\mathbf{x}\right)=\left(\mathbf{P}\,\mathbf{b}\right)$.
2. Use symbolic Cholesky factorization to obtain an exact $\mathbf{L}$ factor (non zero entries in $\mathbf{L}$).
3. Organize operations to improve cache usage.
4. Parallelize the factorization.

Matrix reordering

We want to reorder rows and columns of $\mathbf{A}$, in a way that the number of non-zero entries of $\mathbf{L}$ are reduced. $\eta(\mathbf{L})$ indicates the number of non-zero entries of $\mathbf{L}$.

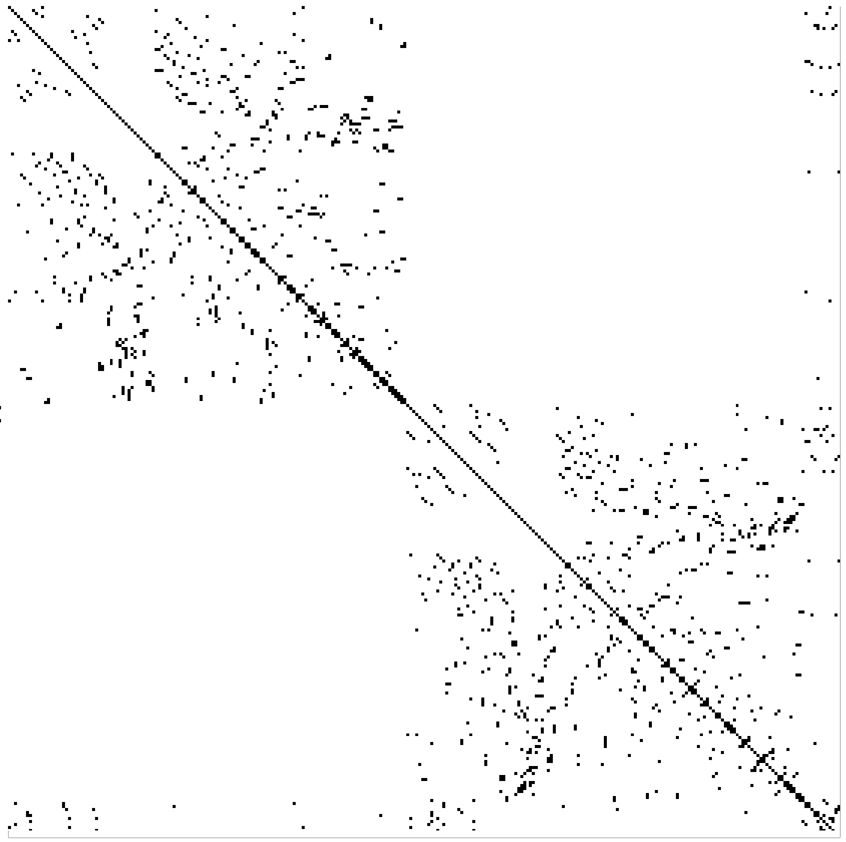
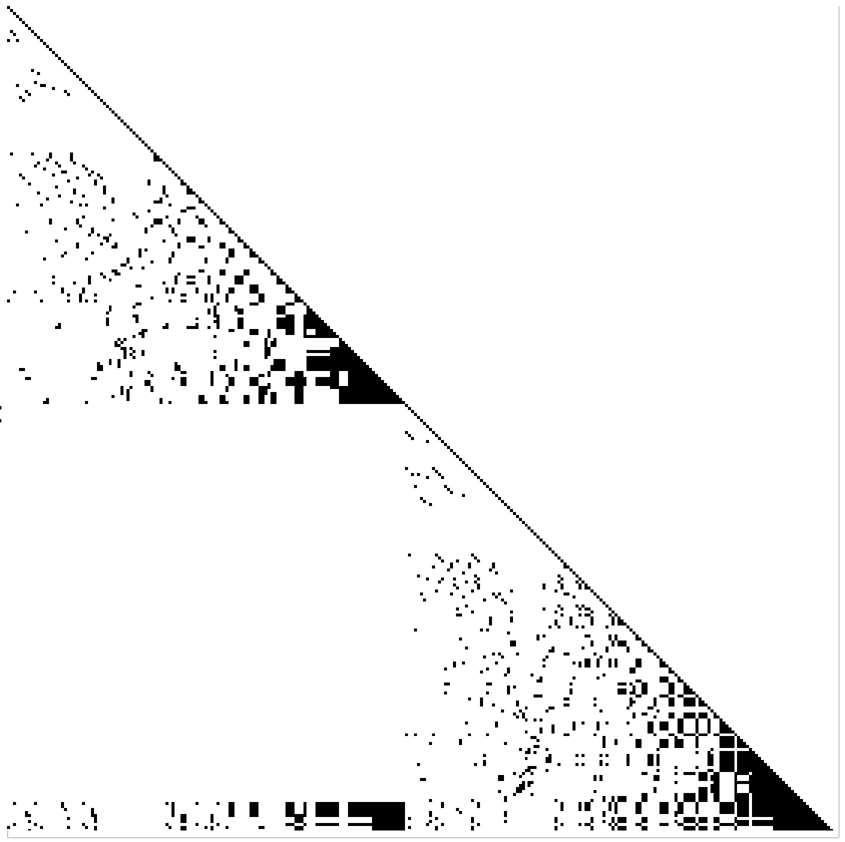
$\mathbf{A} =$ 

$\mathbf{L} =$ 

The stiffness matrix to the left $\mathbf{A} \in \mathbb{R}^{556\times 556}$, with $\eta(\mathbf{A})=1810$. To the right the lower triangular matrix $\mathbf{L}$, with $\eta(\mathbf{L})=8729$.

There are several heuristics like the minimum degree algorithm [Geor81] or a nested dissection method [Kary99].

By reordering we have a matrix $\mathbf{A}'$ with $\eta(\mathbf{A}')=1810$ and its factorization $\mathbf{L}'$ with $\eta(\mathbf{L}')=3215$. Both factorizations solve the same system of equations.

$\mathbf{A}'=$  $\mathbf{L}'=$ 

We reduce the factorization fill-in by

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

To determine a “good” reordering for a matrix $\mathbf{A}$ that minimize the fill-in of $\mathbf{L}$ is an NP complete problem [Yann81].

Symbolic Cholesky factorization

The algorithm to determine the L_{ij} entries that area non-zero is called symbolic Cholesky factorization [Gall90].

Let be, for all columns $j=1\ldots n$,

$$\boldsymbol{a}_j \overset{\text{def}}{=} \{k>j \mid A_{kj}\neq 0\},$$
$$\boldsymbol{l}_j \overset{\text{def}}{=} \{k>j \mid L_{kj}\neq 0\}.$$

The sets $\boldsymbol{r}_j$ will register the columns of $\boldsymbol{L}$ which structure will affect the column j of $\boldsymbol{L}$.

```
for $i \leftarrow 1,2,\ldots,n$
·   $\boldsymbol{r}_i \leftarrow \emptyset$

for $i \leftarrow 1,2,\ldots,n$
·   $\boldsymbol{l}_i \leftarrow \boldsymbol{a}_i$
·   for each $j\in\boldsymbol{r}_i$
·   ·   $\boldsymbol{l}_i \leftarrow \boldsymbol{l}_i\cup\boldsymbol{l}_j\setminus\{i\}$
·   if $\boldsymbol{l}_i\neq\emptyset$
·   ·   $p \leftarrow \min\{j\in\boldsymbol{l}_i\}$
·   ·   $\boldsymbol{r}_p \leftarrow \boldsymbol{r}_p\cup\{i\}$
```

$$\boldsymbol{A}=\begin{pmatrix} a_{11} & a_{12} & & & & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & & \\ & a_{32} & a_{33} & & a_{35} & \\ & a_{42} & & a_{44} & & \\ & & a_{53} & & a_{55} & a_{56} \\ a_{61} & & & & a_{65} & a_{66} \end{pmatrix} \qquad \boldsymbol{L}=\begin{pmatrix} l_{11} & & & & & \\ l_{21} & l_{22} & & & & \\ & l_{32} & l_{33} & & & \\ & l_{42} & l_{43} & l_{44} & & \\ & & l_{53} & l_{54} & l_{55} & \\ l_{61} & l_{62} & l_{63} & l_{64} & l_{65} & l_{66} \end{pmatrix}$$

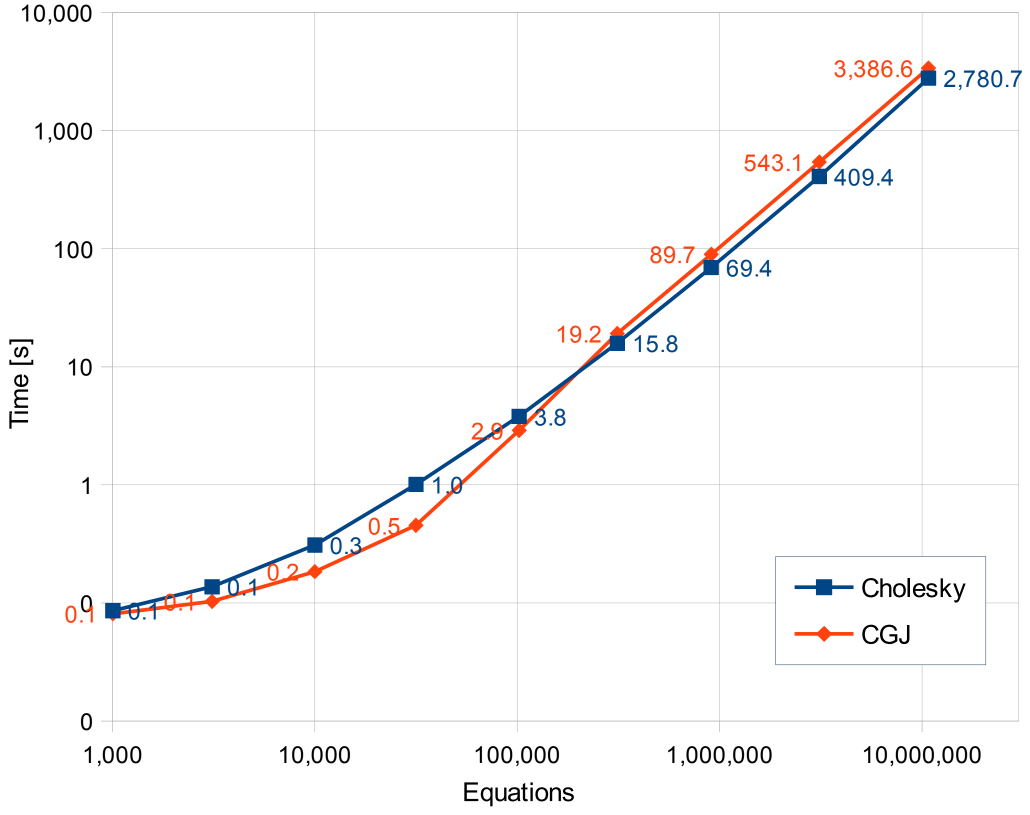
$$\boldsymbol{a}_2=\{3,4\} \qquad \boldsymbol{l}_2=\{3,4,6\}$$

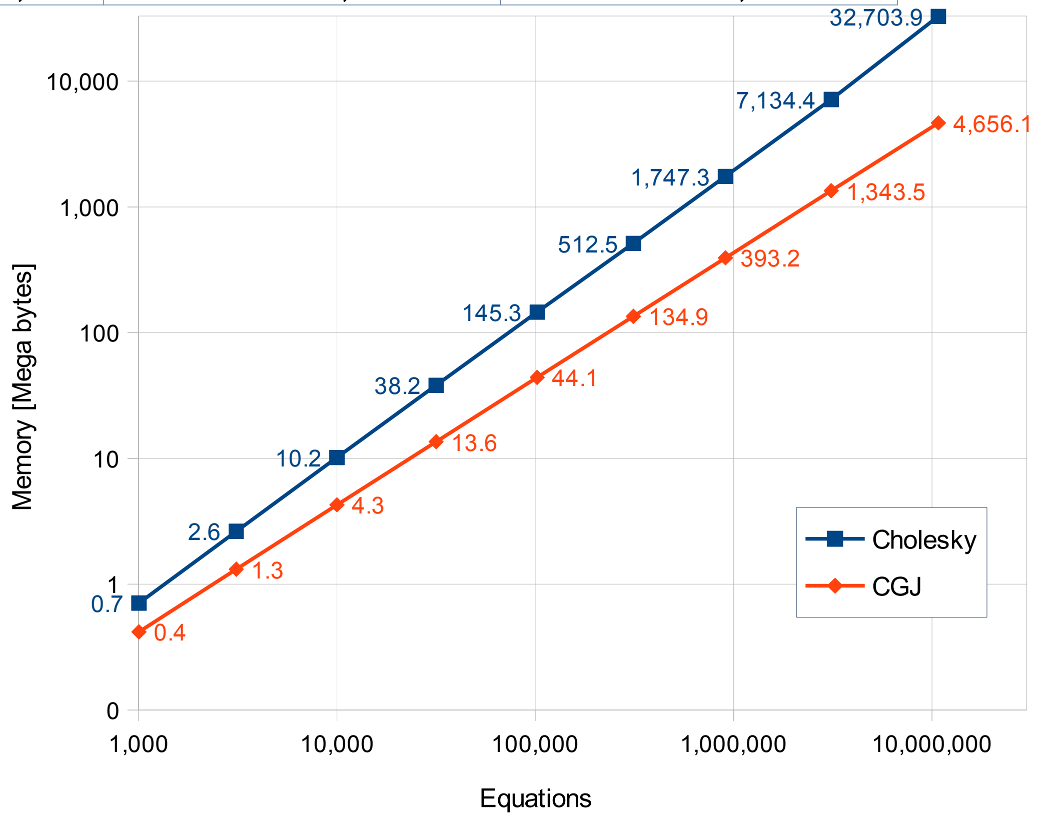
This algorithm is very efficient, its complexity in time and space has an order of $O(\eta(\boldsymbol{L}))$.

How efficient is it?

The next table shows results solving a 2D Poisson equation problem, comparing Cholesky and conjugate gradient with Jacobi preconditioning.

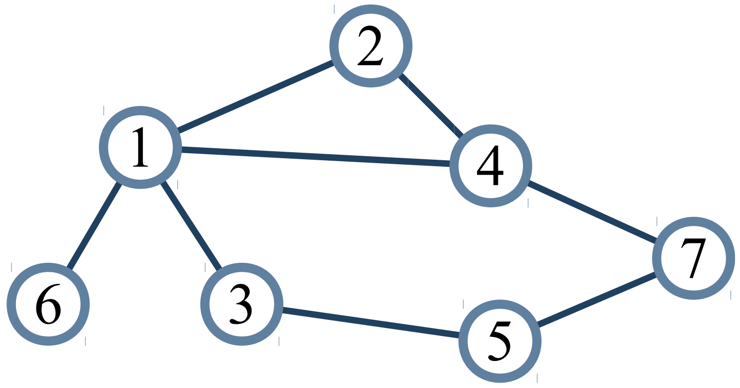
Number of equations	nnz(A)	nnz(L)	Cholesky time [s]	CGJ time [s]
1,006	6,140	14,722	0.086	0.081
3,110	20,112	62,363	0.137	0.103
10,014	67,052	265,566	0.309	0.184
31,615	215,807	1’059,714	1.008	0.454
102,233	705,689	4’162,084	3.810	2.891
312,248	2’168,286	14’697,188	15.819	19.165
909,540	6’336,942	48’748,327	69.353	89.660
3’105,275	21’681,667	188’982,798	409.365	543.110
10’757,887	75’202,303	743’643,820	2,780.734	3,386.609



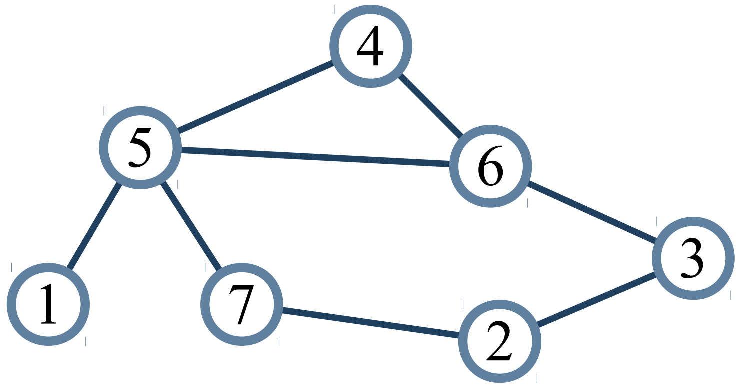


Graph reordering

Let $G=(\boldsymbol{V},\boldsymbol{E})$ be a graph with n vertex $\boldsymbol{V}$ with m edges $\boldsymbol{E}$,



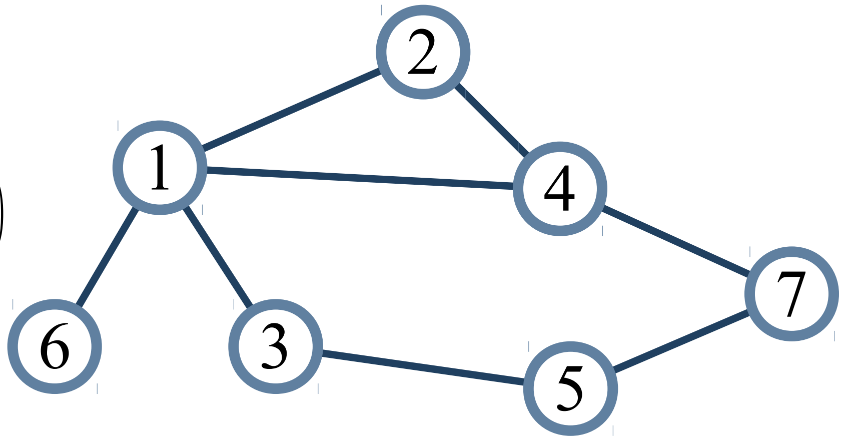
A **ordering** (or tagging) α of G is simply a mapping from the set $\{1,2,\ldots,n\}$ in **V**, where n is the number of vertex of G. The graph tagged by α will be written $G^{\alpha}=(\boldsymbol{V}^{\alpha},\boldsymbol{E}^{\alpha})$.



Let $\mathbf{A}$ be a sparse matrix, symmetric, of size $n\times n$, the ordered graph of $\mathbf{A}$, $G^{\mathbf{A}}=\left(\boldsymbol{V}^{\mathbf{A}},\boldsymbol{E}^{\mathbf{A}}\right)$, in it $\{v_i,v_j\}\in\boldsymbol{E}^{\mathbf{A}}$ if and only if $A_{ij}=A_{ji}\neq 0$, $i\neq j$. Here v_i is a vertex of $\boldsymbol{V}^{\mathbf{A}}$ with tag $\mathbf{A}$.

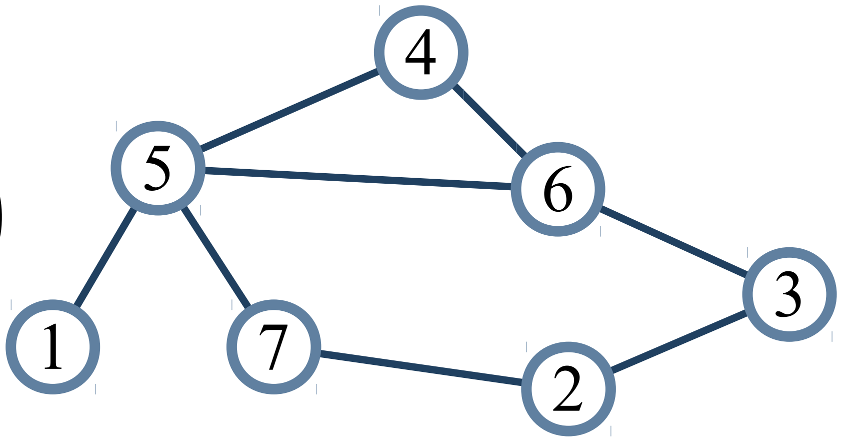
$$\mathbf{A}=\begin{pmatrix}
\bullet & \bullet & \bullet & \bullet & \cdot & \bullet & \cdot \\
\bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \cdot \\
\bullet & \cdot & \bullet & \cdot & \bullet & \cdot & \cdot \\
\bullet & \bullet & \cdot & \bullet & \cdot & \cdot & \bullet \\
\cdot & \cdot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

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



An ordering α applied to $G^{\mathbf{A}}=\left(\boldsymbol{V}^{\mathbf{A}},\boldsymbol{E}^{\mathbf{A}}\right)$ is equivalent to a apply a permutation matrix.

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



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

For the fill-in of the Cholesky factorization, to find a “good” permutation $\mathbf{P}$ for $\mathbf{A}$ that reduces the number of non-zero entries on $\mathbf{L}$ means to find a “good”ordering of its graph [Geor81].

Nested dissection algorithm

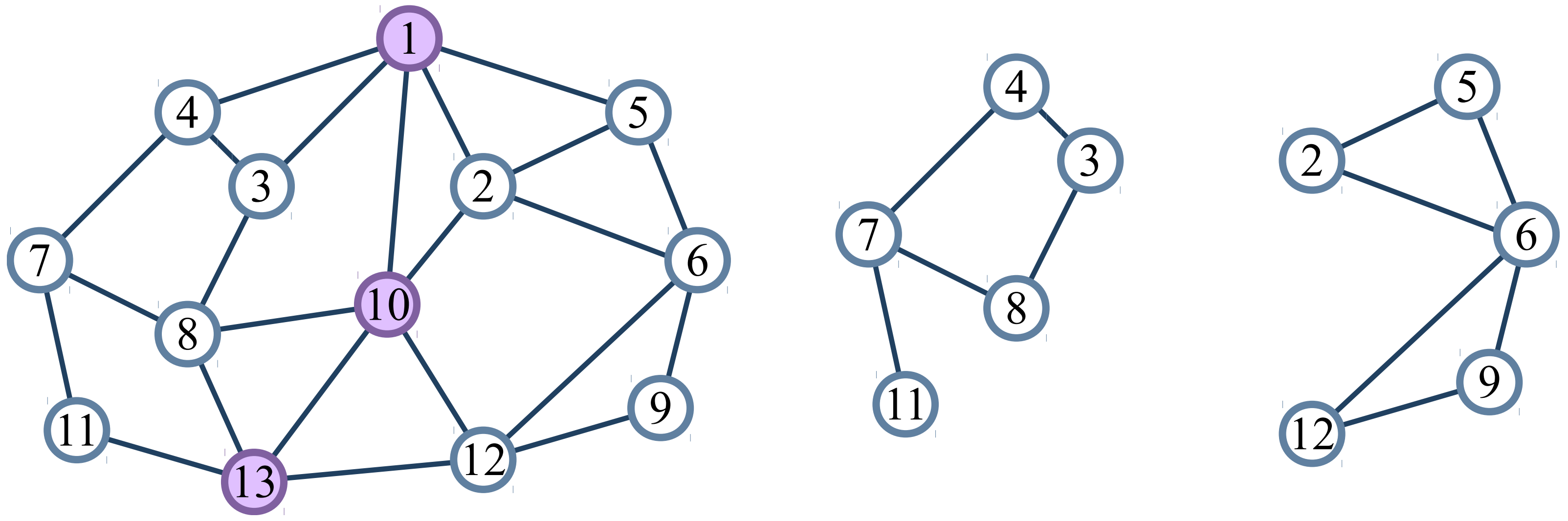
Separator

Lets consider a separator set $\boldsymbol{S} \subset \boldsymbol{V}$ in $G = (\boldsymbol{V}, \boldsymbol{E})$, when removed from G the graph is disconnected in two non-empty sets of vertex $\boldsymbol{R} \subset \boldsymbol{V}$ and $\boldsymbol{S} \subset \boldsymbol{V}$, also

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

In the next example $\boldsymbol{V} = \{1, 2, \ldots, 13\}$, let $\boldsymbol{S} = \{1, 10, 13\}$ be a separator of $G = (\boldsymbol{V}, \boldsymbol{E})$, when removed from the graph we obtain

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



This procedure applied in a recursive way is known as the generalized nested dissection algorithm [Lipt79].

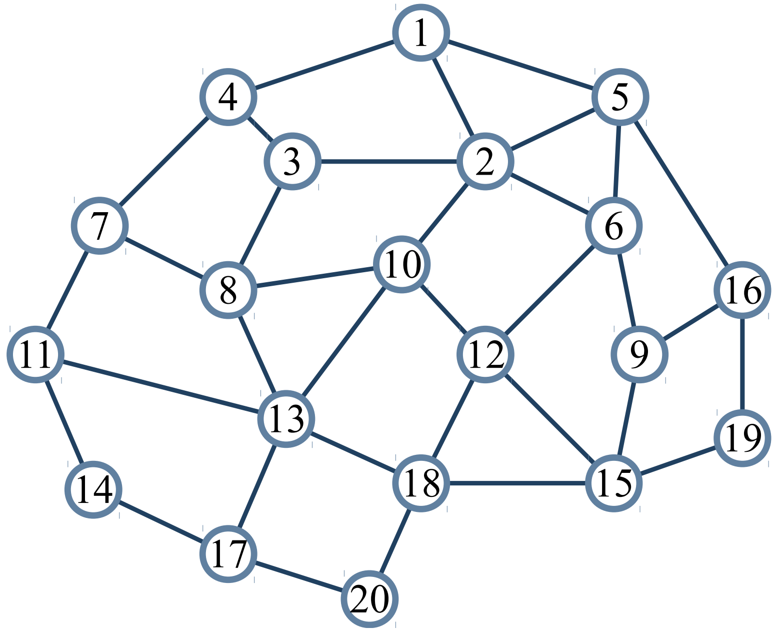
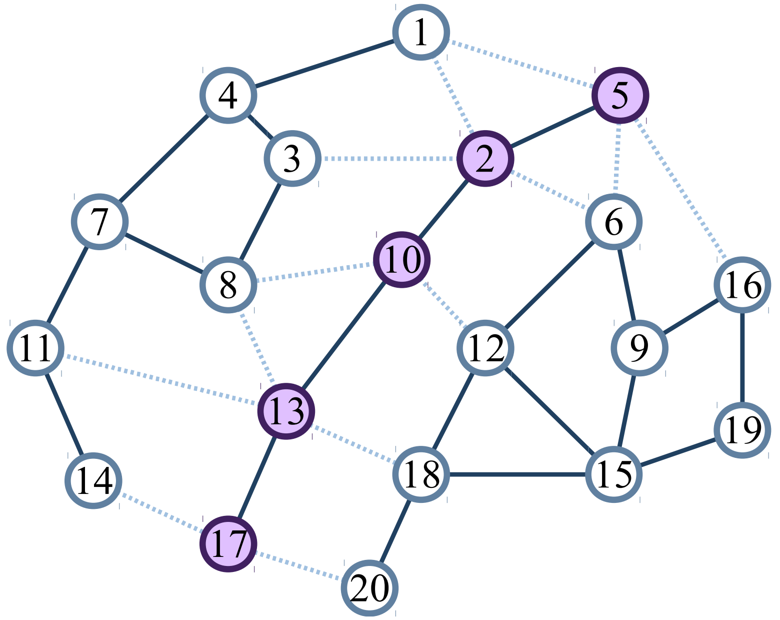
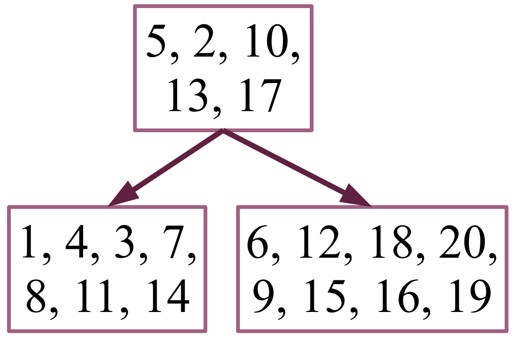
The idea is to split the graph trying to make $\boldsymbol{R}$ and $\boldsymbol{T}$ of the same size with a small separator. This is an divide-and-conquer recursive method.

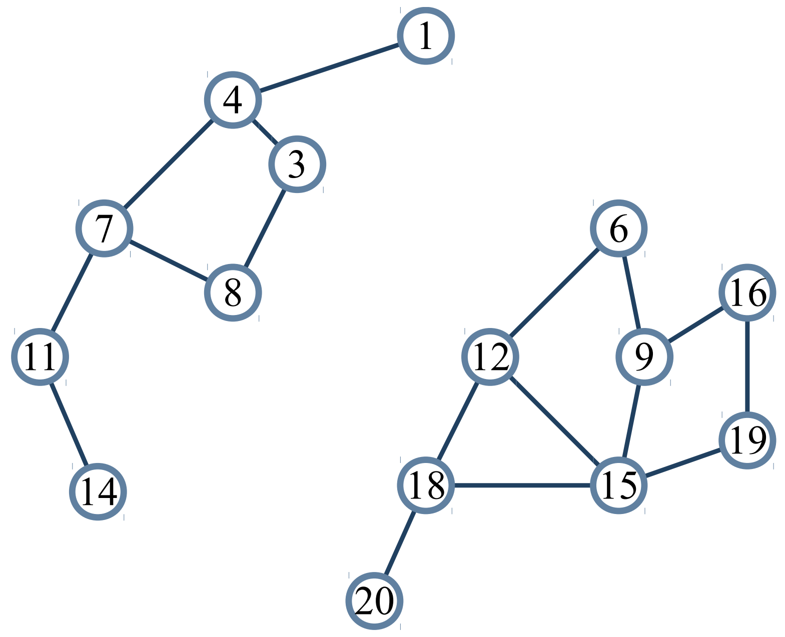
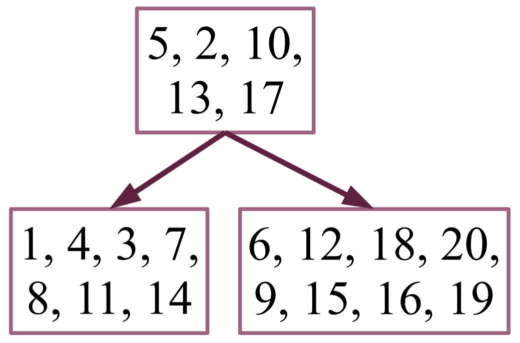
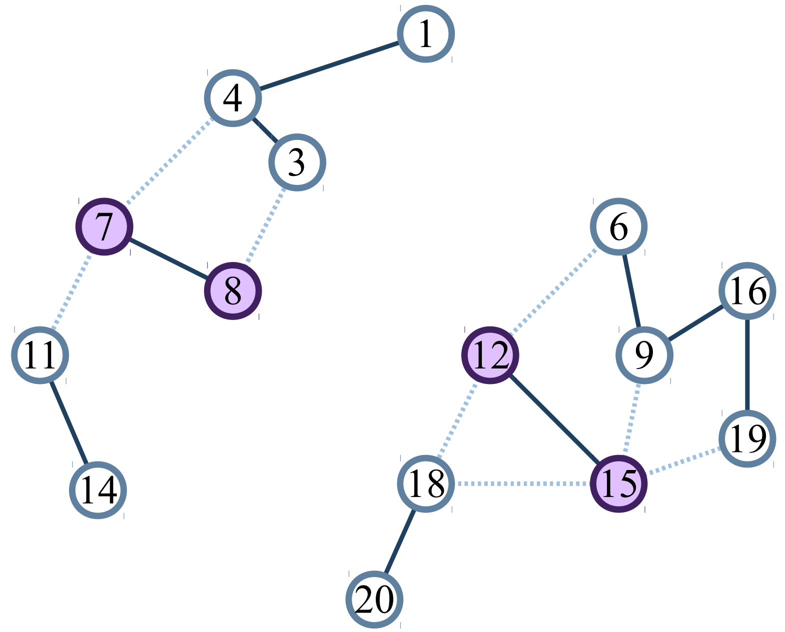
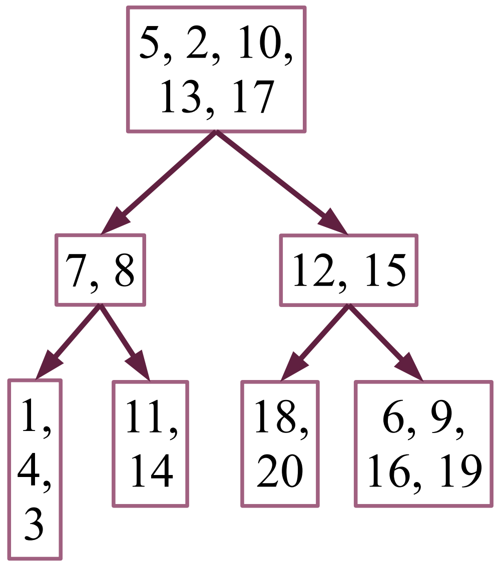
```
Dissection $G(\boldsymbol{V}, \boldsymbol{E})$
If $|G| < \beta$
·   The vertexes $\boldsymbol{V}$ are ordered with any order
    (the minimum degree algorithm can be used)
else
·   Search for a separator set $\boldsymbol{S} \subset \boldsymbol{V}$ such that after removing it from $G$
    we get two sets of disconnected vertexes $\boldsymbol{R}$ and $\boldsymbol{T}$, with $|\boldsymbol{R}| \approx |\boldsymbol{T}|$.
·   Remove all edges from $\boldsymbol{E}$ that have connections with entries in $\boldsymbol{S}$.
·   Dissection $G(\boldsymbol{R}, \boldsymbol{E})$
·   Dissection $G(\boldsymbol{T}, \boldsymbol{E})$
·   Order the vertexes, putting the ones in $\boldsymbol{S}$ at the end of the order.
```

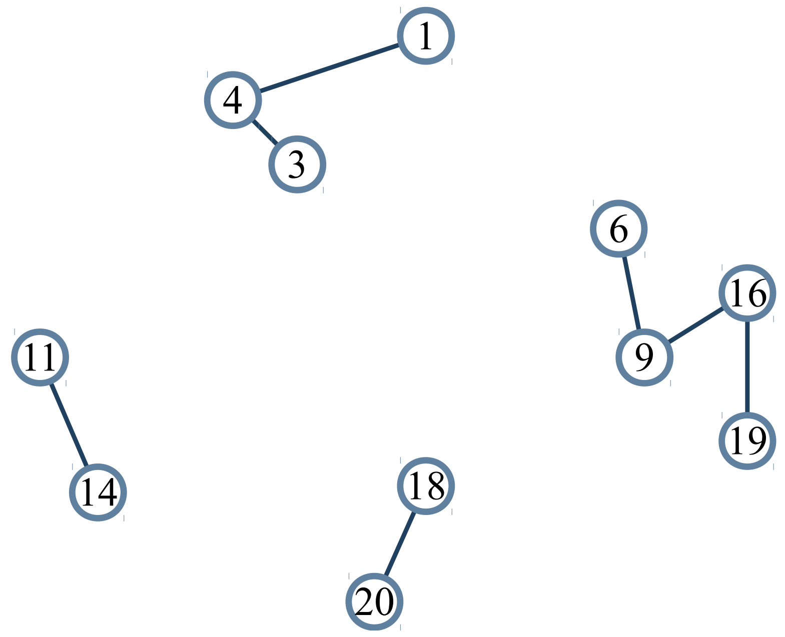
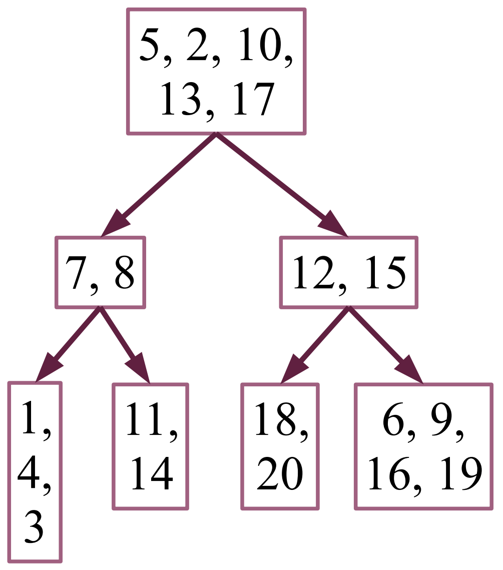
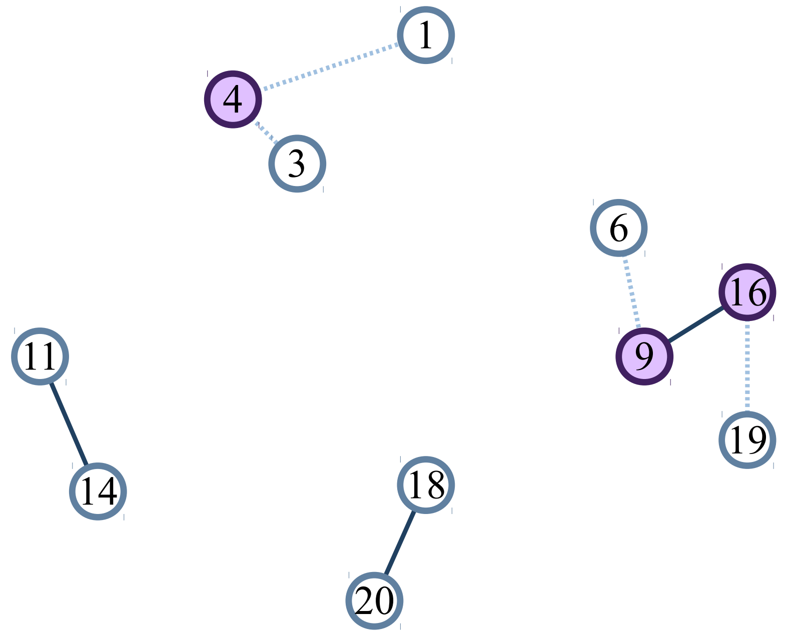
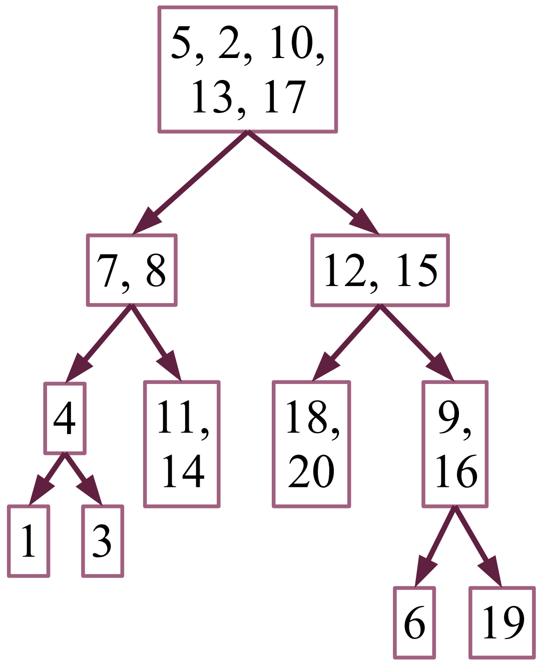
This algorithm is recursive and the resulting order of vertexes of G produces an efficient Cholesky factorization.

An improved version of this algorithm is presented in [Kary99].

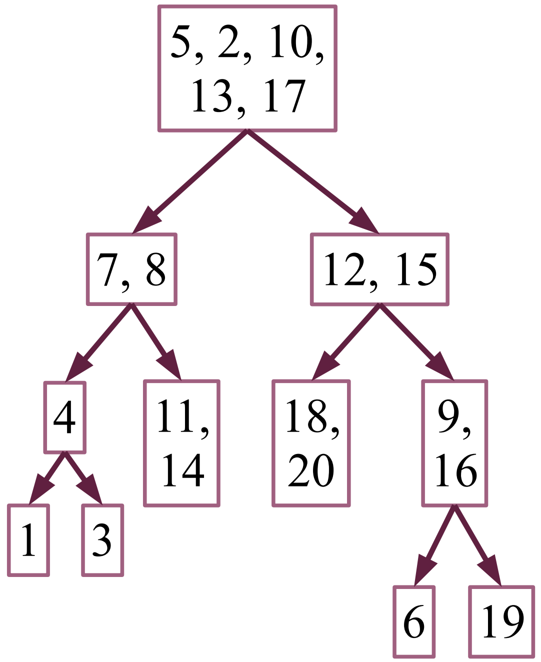
This is an exaple of the nested dissection algorithm:

Graph	Dependency tree
	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
	 5, 2, 10, 13, 17 1, 4, 3, 7, 8, 11, 14 6, 12, 18, 20, 9, 15, 16, 19

Graph	Dependency tree
	
	

Graph	Dependency tree
	 5, 2, 10, 13, 17 7, 8 12, 15 1, 4, 3 11, 14 18, 20 6, 9, 16, 19
	 5, 2, 10, 13, 17 7, 8 12, 15 4 11, 14 18, 20 9, 16 1 3 6 19

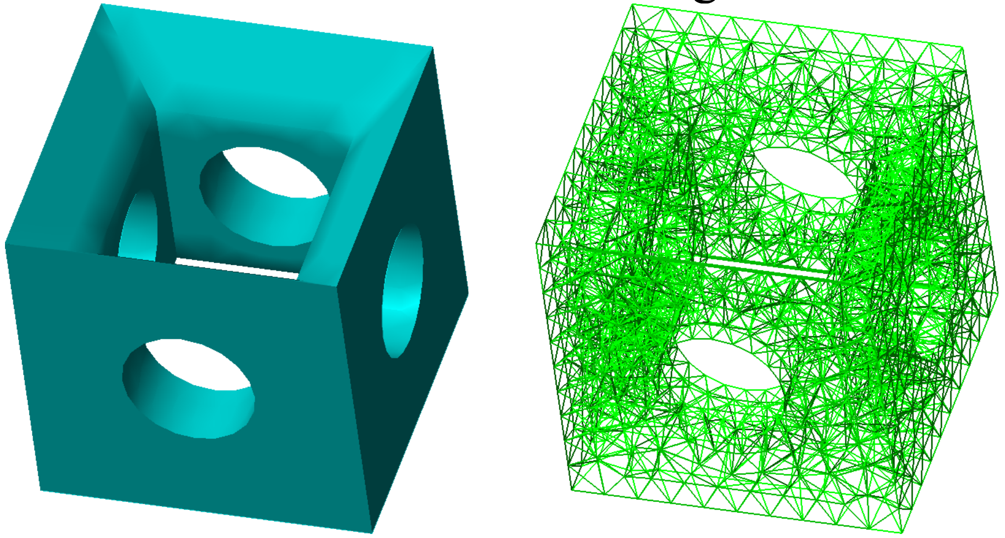
Fron the final dependency tree an ordering is creating numerating the vertexes filling the root at last up to the leaves.



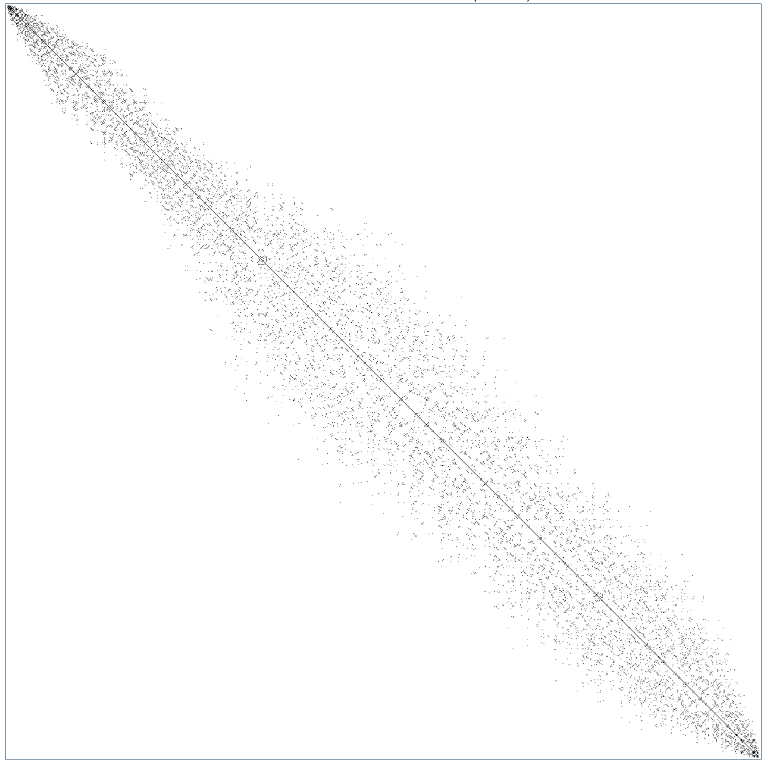
The elimination sequence is 6, 19, 9, 16, 18, 20, 12, 15, 1, 3, 4, 11, 14, 7, 8, 5, 2, 10, 13, 17. This ordering is used to generate **A**', the equivalent **P** is

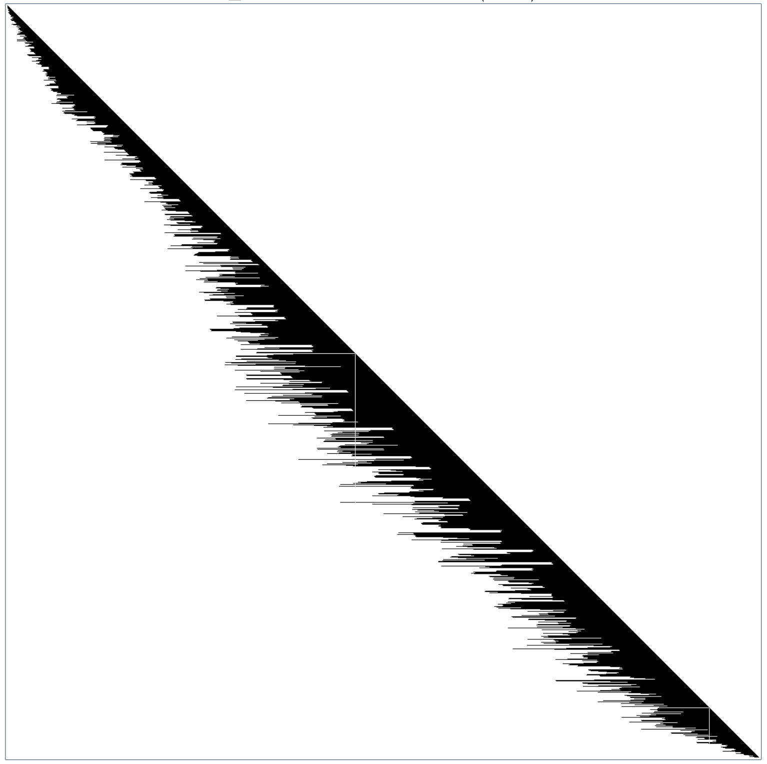
$$\mathbf{P}=\begin{pmatrix}
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\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

This is an example of a finite element mesh reordered using nested dissection

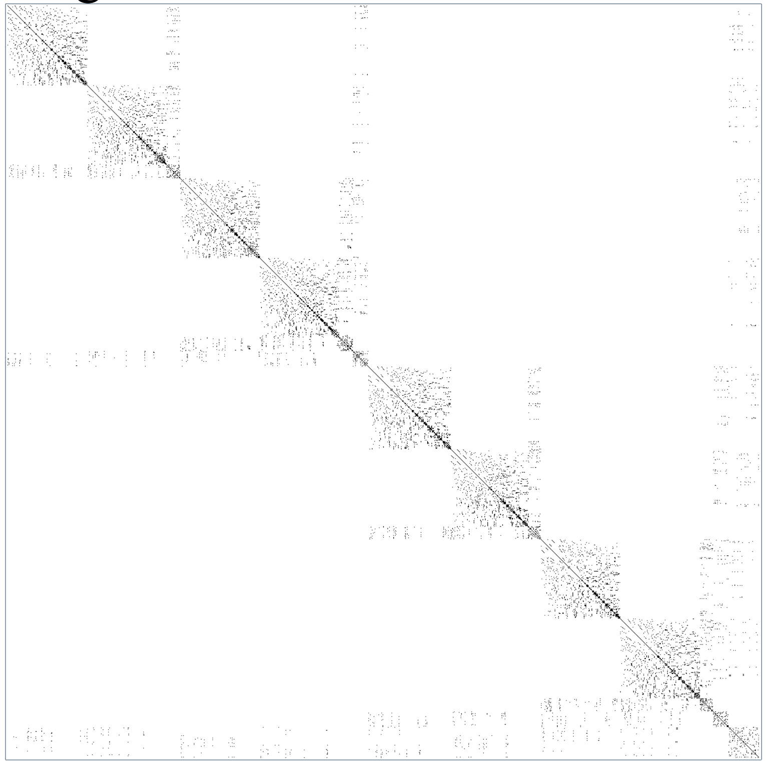


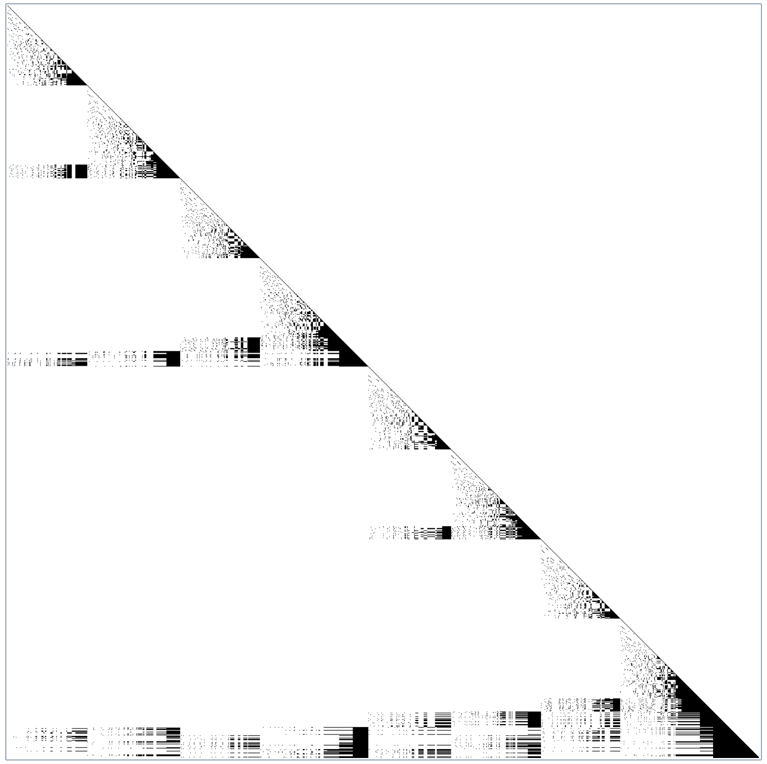
$\mathbf{A}$ has a size $1{,}263 \times 1{,}263$, with $\eta(\mathbf{A}) = 14{,}131$. The factorization produces $\eta(\mathbf{L}) = 128{,}476$

$\mathbf{A} =$ 

$\mathbf{L} =$ 

Reordering using nested dissection

$\mathbf{A}^r =$ 

$\mathbf{L}^r =$ 

The factorization produces $\eta(\mathbf{L}^r) = 39{,}465$. For this case

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

Heuristics to obtain “good” separators

Basically we have to goals:

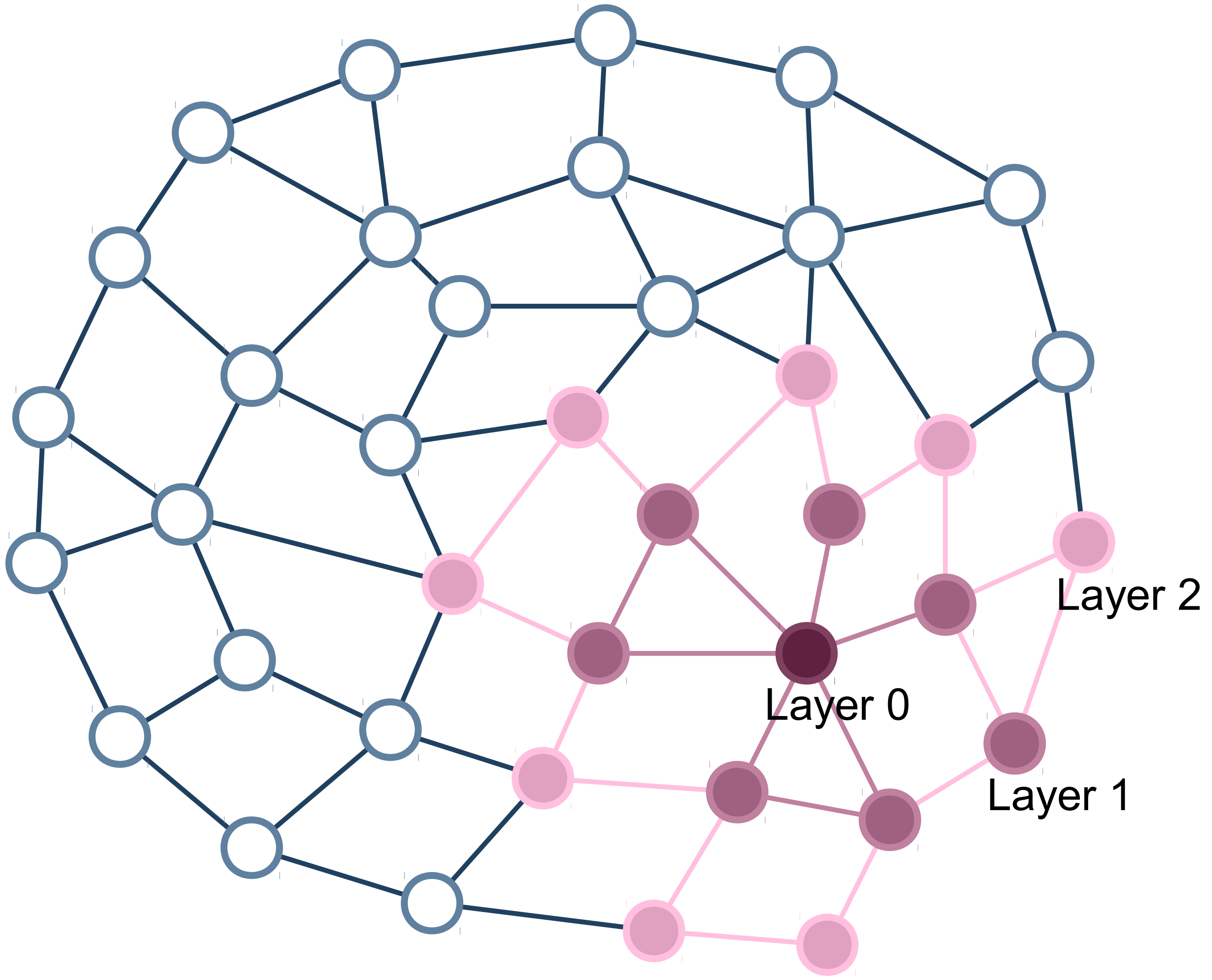
a) Split the graph in two approximatelly equal sub-graphs.

b) Split the graph using the minimum number of vertexes in the separator.

We will try to locate the “center” of the graph and then we will try to generate a plane that cuts the graph in approximately two equal parts.

The difficult part is that this has to be done without knowing the coordinates of nodes of the mesh.

Generating ‘balls’

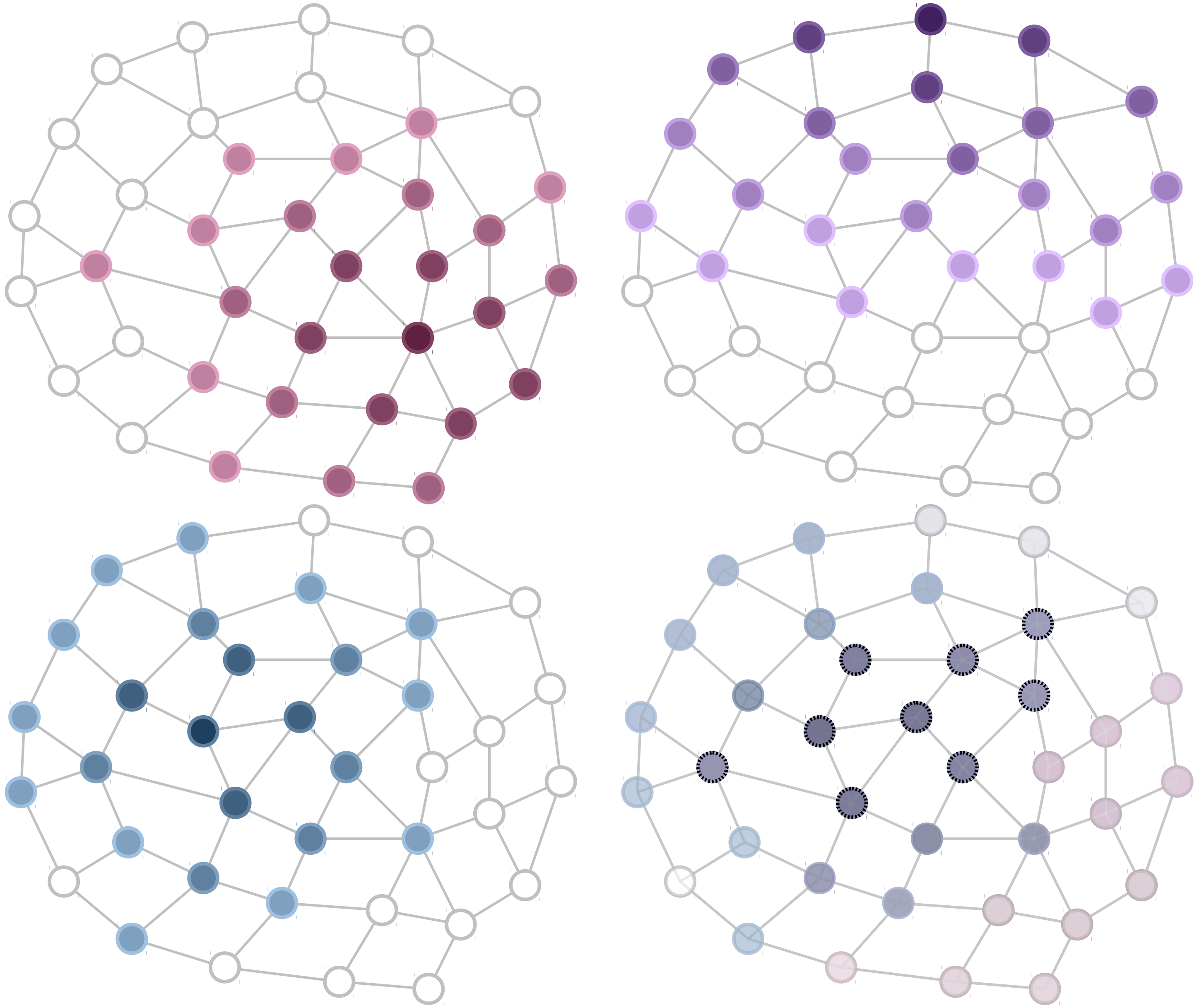


Balls are created selecting a vertex randomly, lets call it ‘layer 0’. Next a layer of all adjacent vertexes to ‘layer 0’ is added, forming ‘layer 1’. Same process to form ‘layer 2’, etc.

This procedure works for 3D meshes.

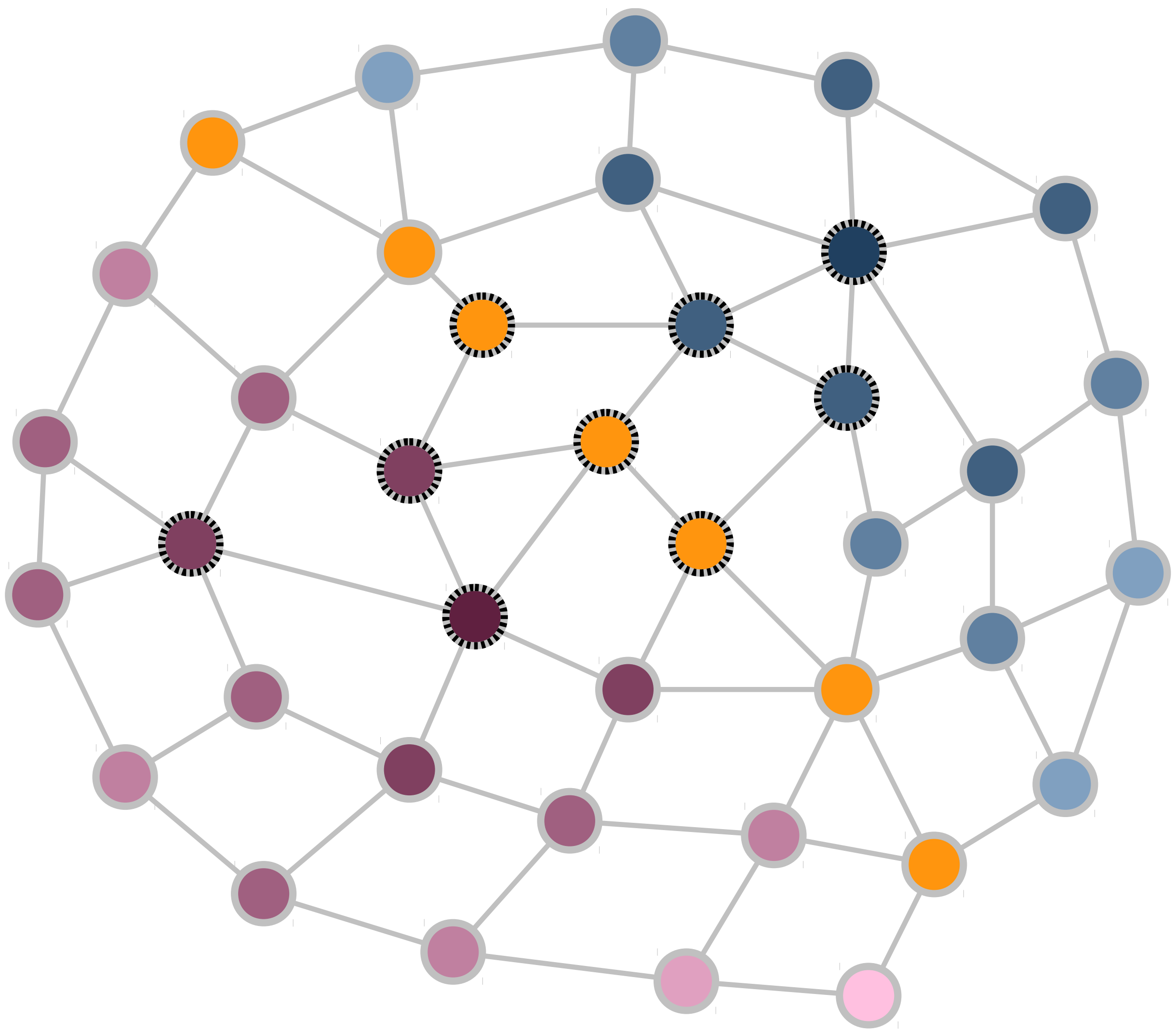
Heuristic to locate the center of the graph

This is done creating several balls, each one has $\frac{n}{2}+\alpha$ vertexes. The intersection of all balls will be choosen as the center of the graph.



Separator creation

Two nodes of the ‘center’ of the graph are randomly selected, they are used as center of balls.



Layers of both balls are grown at the same time. When a vertex is part of the layer of both balls it is maked as a separator and removed from the layers.

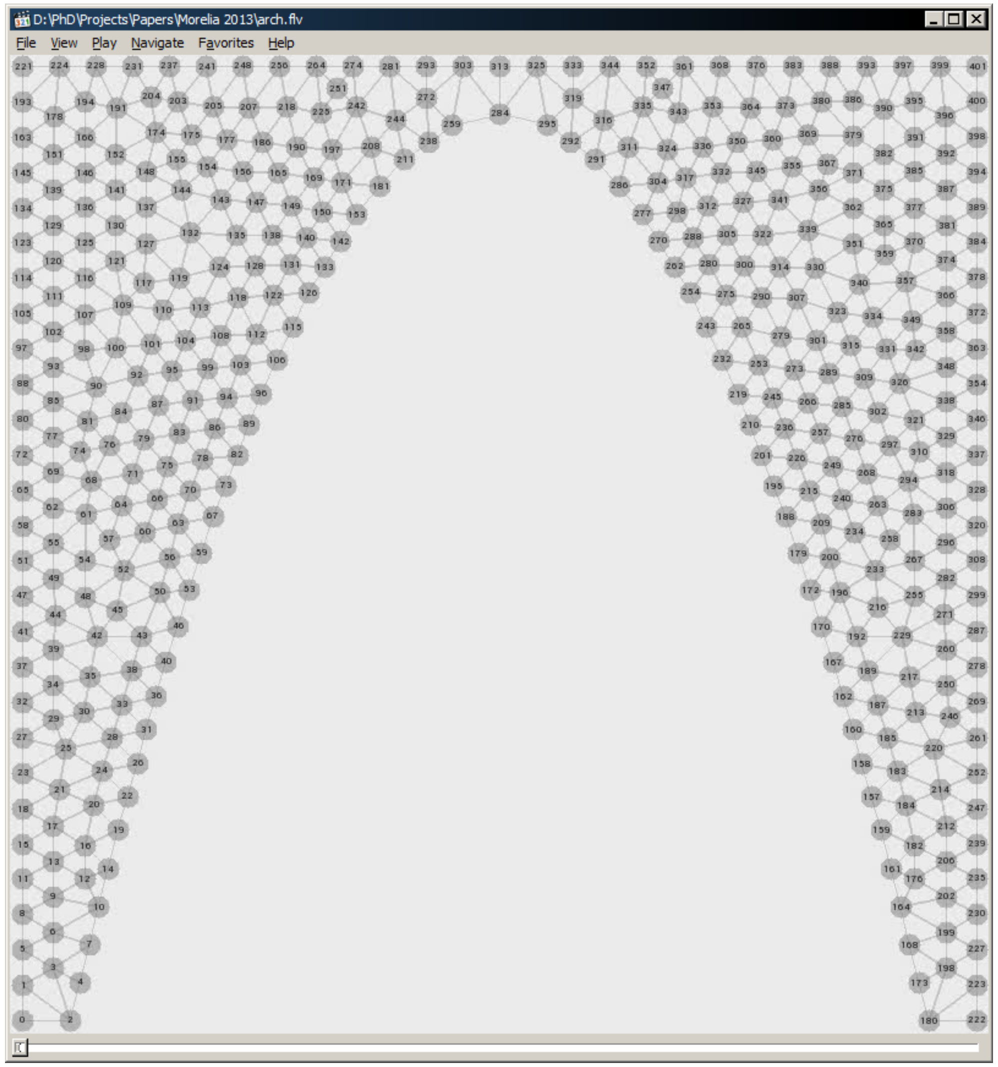
Several separators are created, the best one is used. It is choosen the one that has the minimun fitness. The fitness function is:

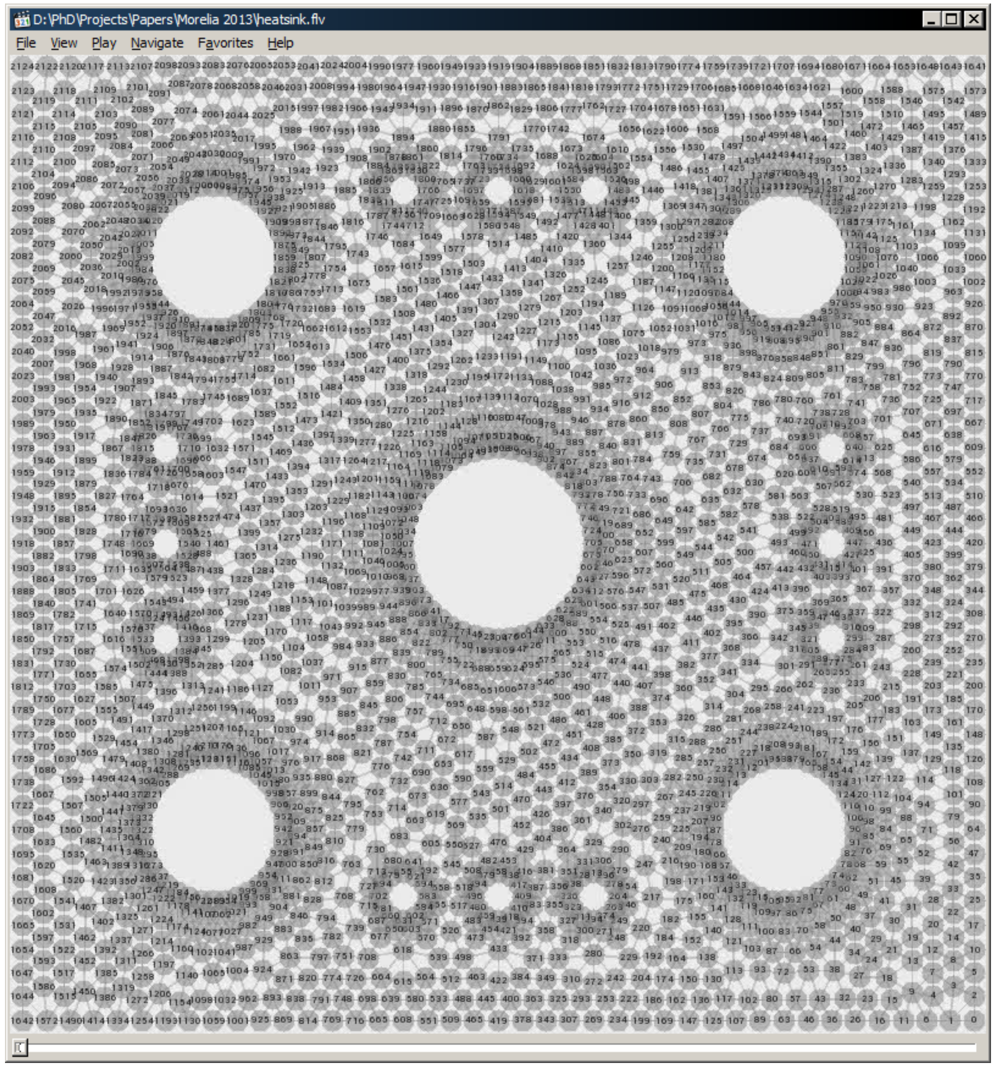
$$f = s\left(\frac{n_{\min}}{n_{\max}}\right),$$

where s is the separator size, $n_{\min}$ is the number of nodes of the partition with less nodes, and $n_{\max}$ is the number of nodes of the partition with more nodes.

Examples

These are examples of the recursive evolution of the nested dissection.

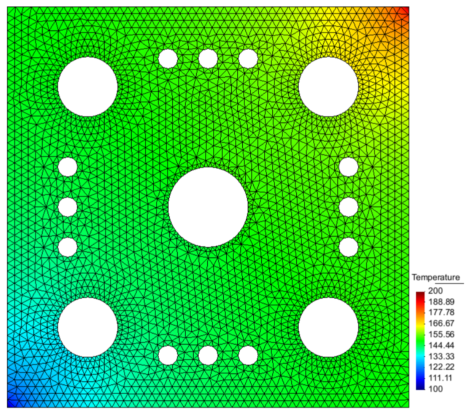

D:\PhD\Projects\Papers\Morelia 2013\arch.flv
File View Play Navigate Favorites Help


D:\PhD\Projects\Papers\Morelia 2013\heatsink.flv
File View Play Navigate Favorites Help

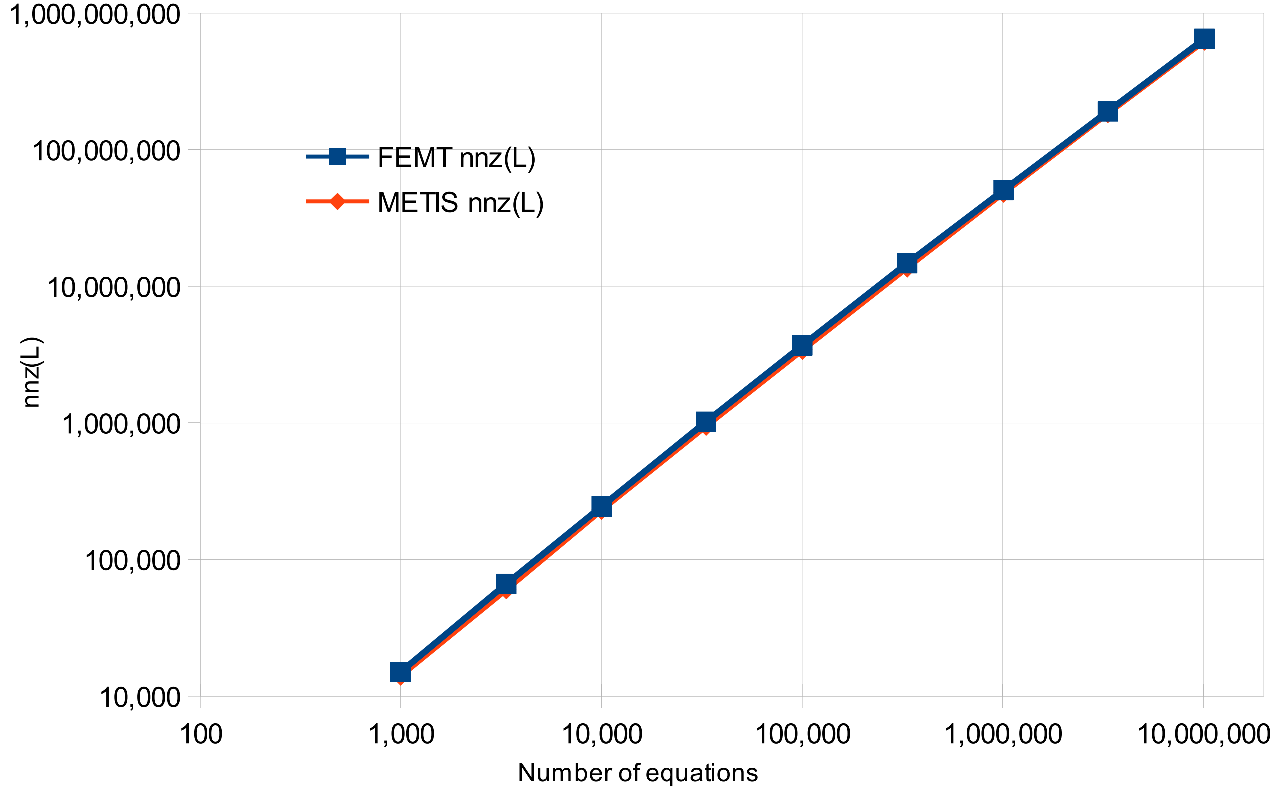
Numerical experiments

We tested these heuristics against the METIS library [Kary99].

n	nnz(A)	FEMT nnz(L)	METIS nnz(L)	Difference
998	6,490	15,760	14,149	6.4%
3,361	22,311	66,280	60,492	9.5%
10,013	67,819	255,221	231,332	5.7%
33,258	228,502	1,052,032	952,013	7.1%
100,529	695,927	3,843,843	3,434,915	7.6%
334,643	2,328,077	15,590,679	13,760,163	7.3%
1,008,572	7,034,658	52,843,928	48,576,349	4.0%
3,343,042	23,354,982	190,399,689	185,091,059	2.9%
10,136,296	70,900,872	648,691,337	629,026,259	3.1%



Contour Fill of Temperature.



Future work

Improve speed. Dynamically set the number of separators tried by the number of nodes.

Improve speed by parallelizing the creation of separators.

Improve the fitness function.

Use nested dissection as a partitioning strategy [Kary99].

Questions?

miguelvargas@cimat.mx

References

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

[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.

[Gall90] K. A. Gallivan, M. T. Heath, E. Ng, J. M. Ortega, B. W. Peyton, R. J. Plemmons, C. H. Romine, A. H. Sameh, R. G. Voigt. Parallel Algorithms for Matrix Computations. SIAM, 1990.