FEM and Sparse Linear System Solving

Lecture 6, October 25, 2013

Sparse matrices & Fill-reducing orderings

http://people.inf.ethz.ch/tulink/FEM13.html

Tulin Kaman and Peter Arbenz
Computer Science Department, ETH Zürich
E-mail: tulin.kaman@inf.ethz.ch

- The finite element method
  - Introduction, model problems.
  - 1D problems. Piecewise polynomials in 1D.
  - 2D problems. Triangulations. Piecewise polynomials in 2D.
  - Variational formulations. Galerkin finite element method.
  - Implementation aspects.
- Direct solvers for sparse systems
  - LU and Cholesky decomposition
  - Sparse matrices
  - Fill-reducing orderings
- Iterative solvers for sparse systems
  - Stationary iterative methods, preconditioning
  - Preconditioned conjugate gradient method (PCG)
  - Incomplete factorization preconditioning
  - Multigrid preconditioning
  - Nonsymmetric problems (GMRES, BiCGstab)
  - Indefinite problems (SYMMLQ, MINRES)

Review of last week: dense matrix factorizations I

Theorem

If A is nonsingular, then one can find a row permutation P, a unit lower triangular matrix L, and an upper triangular matrix U such that is $PA = LU$.

If $PA = LU$, then instead of

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

one solves

$$P^T LU\mathbf{x} = P^T\mathbf{b} \quad \Longleftrightarrow \quad L\mathbf{y} = P^T\mathbf{b}, \qquad U\mathbf{x} = \mathbf{y}.$$

REMARK: L and U can be stored in A. (They overwrite A.)

Review of last week: dense matrix factorizations II

Theorem

If A is symmetric positive definite (SPD), then there is a lower triangular matrix L such that $A = LL^T$.

The factorization is unique if we request that the diagonal elements of L are positive.

If $A = LL^T$, then instead of

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

one solves

$$LL^T\mathbf{x} = \mathbf{b} \quad \Longleftrightarrow \quad L\mathbf{y} = \mathbf{b}, \qquad L^T\mathbf{x} = \mathbf{y}.$$

REMARK: Only the lower (upper) triangle of A is accessed.

Review of last week: Sparse Factorizations and fill-in I

Consider the LU (Cholesky) factorization of a sparse matrix A.

- The factors L and U are in general sparse again.
- The structure of the L and U factors, the locations of any non-zero entries, is different as compared to A. Matrix entries which are zero in A, but non-zero in L or U are called fill-ins.
- Triangular factors generally have more, even much more (!), nonzero entries than the original matrix.
- Fill-in requires a larger amount of memory and cause a more expensive factorization.
- By permuting (reordering) the rows and columns of A, L and U can be made more or less sparse.

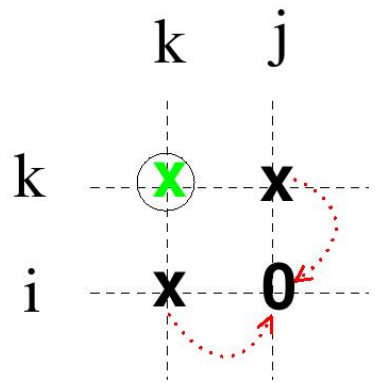
Survey of Sparse Matrix Storage Formats

- Compressed Row Storage (CRS)
- Compressed Column Storage (CCS), used by MATLAB, Rutherford-Boeing
- Block Compressed Row Storage (BCRS)
- Block Compressed Column Storage (BCCS)
- Compressed Diagonal Storage (CDS)
- Jagged Diagonal Storage (JDS)
- Skyline Storage (SKS)
- Element-by-element (FEM)

All formats have advantages/disadvantages. The best choice depends on context (matrix structure) and algorithm.

Sparse *LU* and fill-in

```
A =  1     0     0     0     0
     0     2     5     0     0
     0     3     6     0     9
     0     4     0     8     0
     0     0     7     0    10
```




```
L-eye(5) + U =
    1.0000         0         0         0         0
         0    2.0000    5.0000         0         0
         0    1.5000   -1.5000         0    9.0000
         0    2.0000    6.6667    8.0000  -60.0000
         0         0   -4.6667         0   52.0000
```

The graph connection

It is important to find fill-in reducing permutations or reorderings. The tool for predicting and reducing the amount of fill-in is graph .

- If no pivoting occurs, we can compute the structural fill-in symbolically!
- We represent the matrix as a graph and work on the graph.
- By doing this we do not consider the actual values of the matrix entries!
  (Therefore the word 'structural'.)

Adjacency graph & incidence matrix

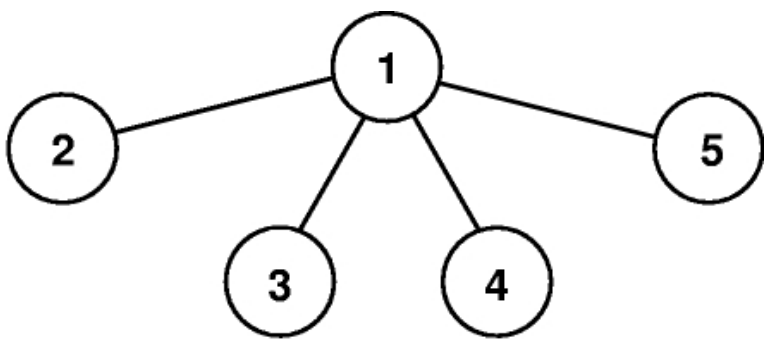
Definition

The adjacency graph $G(S) = (V, E)$ of the symmetric sparse matrix $S \in \mathcal{R}^{n \times n}$ consists of the vertices $V = \{1, \ldots, n\}$ and the edges $E = \{(i, j) : s_{ij} \neq 0\}$.

(Dual definition: incidence matrix for given undirected graph.)

Example: arrow matrix

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & & & \\ \times & & \times & & \\ \times & & & \times & \\ \times & & & & \times \end{bmatrix}$$

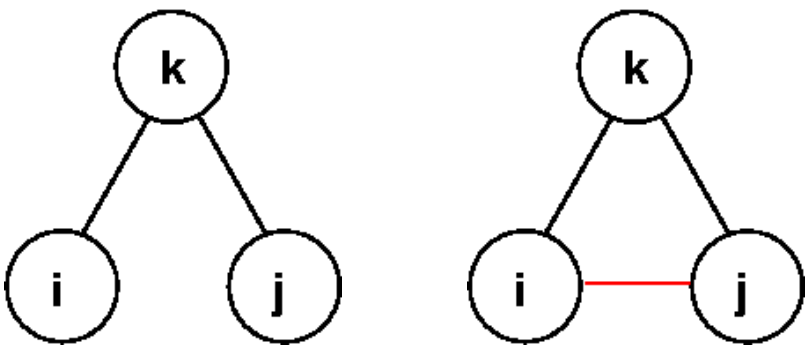


When does fill-in occur?

Let's consider the k-th step in Gaussian elimination.
Let us assume that $a_{ik}^{(k)} \cdot a_{kj}^{(k)} \neq 0$ but that $a_{ij}^{(k)} = 0$.
Then, according to

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} \cdot a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

there will be fill in.



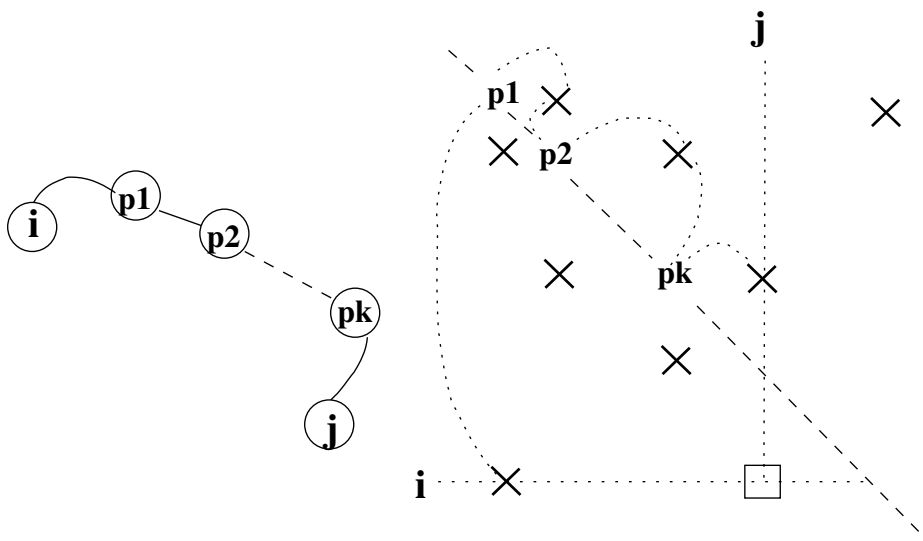
In the k-th elimination step all neighbors of node k are connected.

Cholesky fill-in path characterization

- Cholesky does not need pivoting
- Fill-in can be predicted *beforehand*!

Theorem (Fill path, D. Rose, R. Tarjan, G. Lueker)

Let the Cholesky factorization $A = LL^T$ exist. Then $l_{ij} \neq 0$ if and only if there is an edge path connecting the vertices $i, p_1, \ldots, p_k, j$ from $G(A)$, with $p_1, \ldots, p_k < \min(i, j)$.

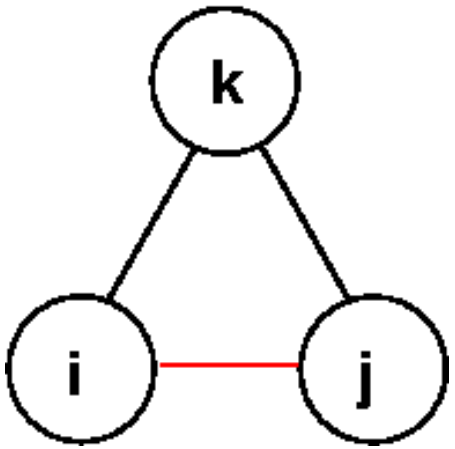


Fill-in in the Cholesky factorization

Let $k < i < j$.

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}^{(k)} \cdot l_{jk}^{(k)}$$

In the k-th elimination step there is fill if $a_{ij}^{(k)} = 0$ and $l_{ik}^{(k)} \cdot l_{jk}^{(k)} \neq 0$.

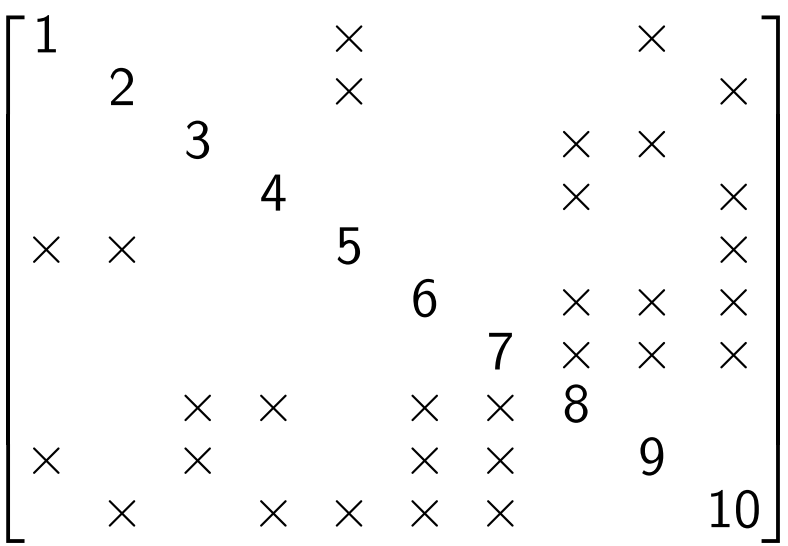


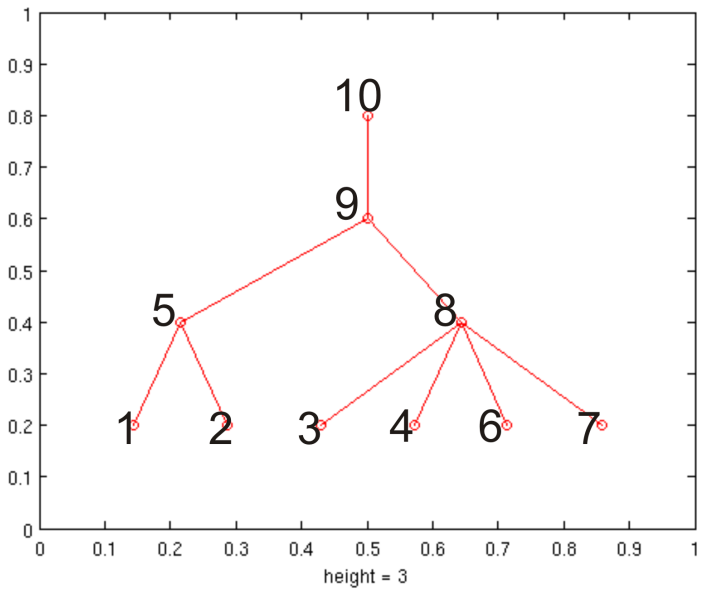
Any fill-in that is caused by l_{jk} (or the edge (j, k)) is recovered by l_{ik} and l_{ij} (or the path $j \to i \to k$).

Therefore, the first nonzero element of each column below the diagonal holds all the necessary information to compute the nonzeros of the Cholesky factorization.

The elimination tree (1)

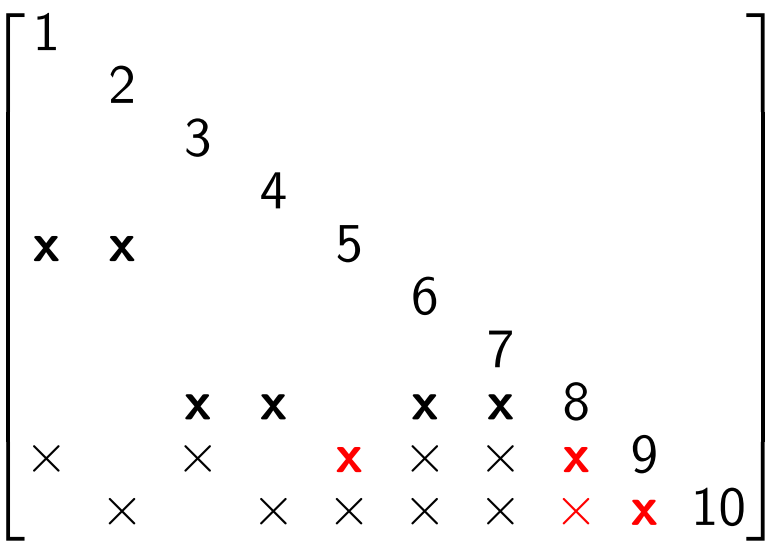
- Represents partial ordering for Cholesky factorization
- See Matlab's `etree`, `etreeplot`

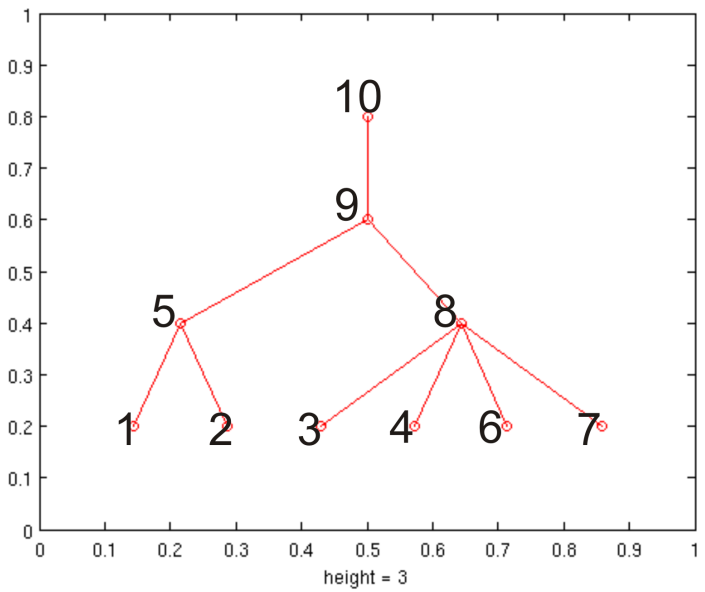




The elimination tree (2)

- Computation of elim. tree from symbolic Cholesky factor.
- Rule: *first* entry (bold) below diagonal determines ordering.





J. Liu: *The Role of Elimination Trees in Sparse Factorization*, SIAM J. Matrix Anal. **11**, 134–172 (1990).

The elimination tree (3)

- The elimination tree can be used to cheaply determine the location of the nonzeros of the Cholesky factor, column by column.
- The parent of node i in the tree is the $k > i$ where l_{ki} is the first nonzero off-diagonal element of L in column i.

Theorem (Schreiber 1982)

For a Cholesky factorization $A = LL^T$, and neglecting numerical cancellation, $l_{ki} \neq 0$ and $k > i$ imply that i is a descendant of k in the elimination tree. Equivalently, $i \to k$ is a path in the elimination tree.

LU fill-in

Pivoting makes LU more complicated than Cholesky!

Practical issues:

- Fill-in in general *not* predictable a-priori.
- Symbolic analysis is *not* enough, numerical values matter too!
- 'Dynamic' pivoting conflicts with a-priori estimated fill-in.
- A-priori estimates may be too small, may need to provide & manage additional memory during factorization.

To get an a-priori estimate on fill-in, commonly use $G(A + A^T)$. Without pivoting, this yields an upper bound.

Scaling & equilibration

'Badly scaled systems' can look artificially hard! Compare:

$$\begin{bmatrix} 10 & 10^8 \\ 10^8 & 10^{16} \end{bmatrix} \quad \text{vs.} \quad \begin{bmatrix} 10 & 1 \\ 1 & 1 \end{bmatrix}$$

Idea: apply nonsingular diagonal scaling matrices D_r, D_c to equilibrate as much as possible:

- $A \Rightarrow D_r A D_c$, here: $D_r = D_c = \text{diag}(1, 10^{-8})$,
- $Ax = b \Leftrightarrow (D_r A D_c) D_c^{-1} x = D_r b$

Scaling does influence pivoting (see above!), and thus also

- fill-in & complexity,
- condition number, accuracy.

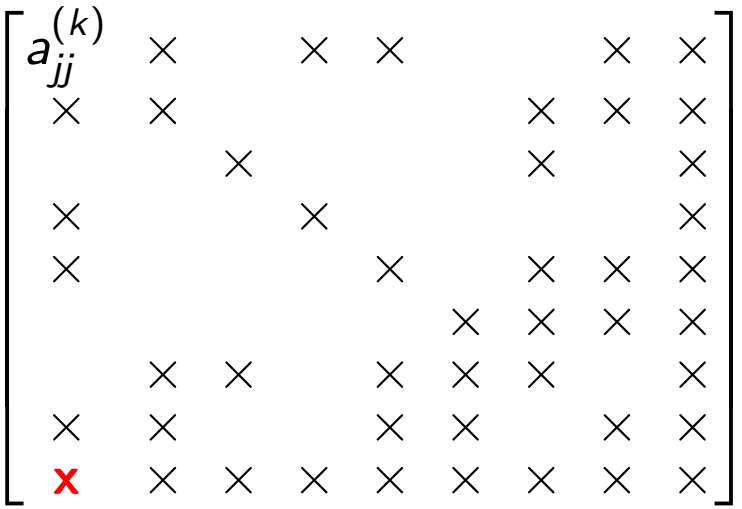
Dynamic Threshold-Pivoting (LU)

- Partial-Pivoting: good stability
- Threshold-Pivoting: tradeoff some stability for smaller fill-in

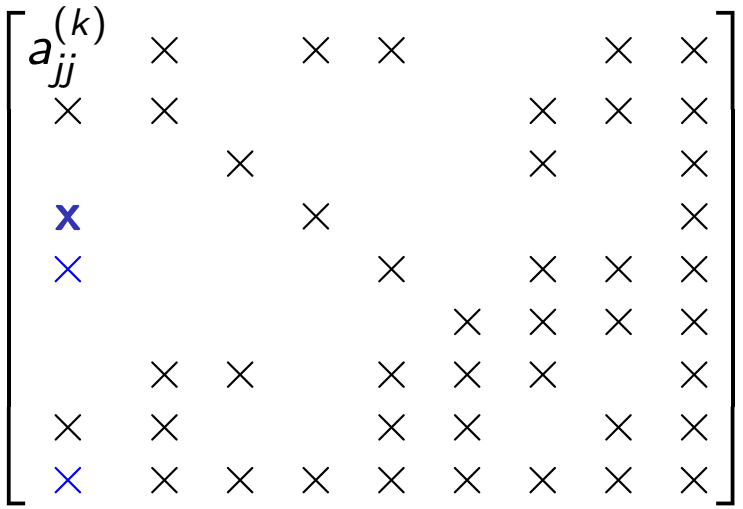
Select pivot with smallest fill-in from set

$$\{a_{ij}^{(k)} : |a_{ij}^{(k)}| \geq u \cdot \max_i |a_{ij}^{(k)}|\}, \quad 0 < u \leq 1. \ (e.g.\ u := 0.1)$$

Partial Pivoting:



Threshold Pivoting:



3 phases of direct linear solvers for sparse $Ax = b$

1. Analysis (or: symbolic factorization)
   - Analysis of the nonzero structure
   - Row- and column-permutations
   - Scaling to improve condition number
2. Numerical Factorization
   - $A = LU$ unsymmetric, with pivoting
   - $A = LL^T$ symmetric positive definite, no pivoting
3. Solve $Ax = b$ using triangular factors
   - Can also solve for multiple right-hand sides
   - If more accuracy necessary: Iterative Refinement

Band matrix

A is a $n \times n$ **band** matrix if all matrix elements are zero outside a diagonally bordered band

$$a_{i,j} = 0 \text{ if } \quad j < i - k_1 \qquad j > i + k_2; \quad k_1, k_2 \geq 0.$$

A band matrix with

- $k_1 = k_2 = 0 \Rightarrow$ Diagonal Matrix
- $k_1 = k_2 = 1 \Rightarrow$ Tridiagonal Matrix
- $k_1 = k_2 = 2 \Rightarrow$ Pentadiagonal Matrix
- $k_1 = 0, k_2 = n - 1 \Rightarrow$ Upper Triangular matrix
- $k_1 = n - 1, k_2 = 0 \Rightarrow$ Lower Triangular matrix

The bandwidth of the matrix is $k_1 + k_2 + 1$

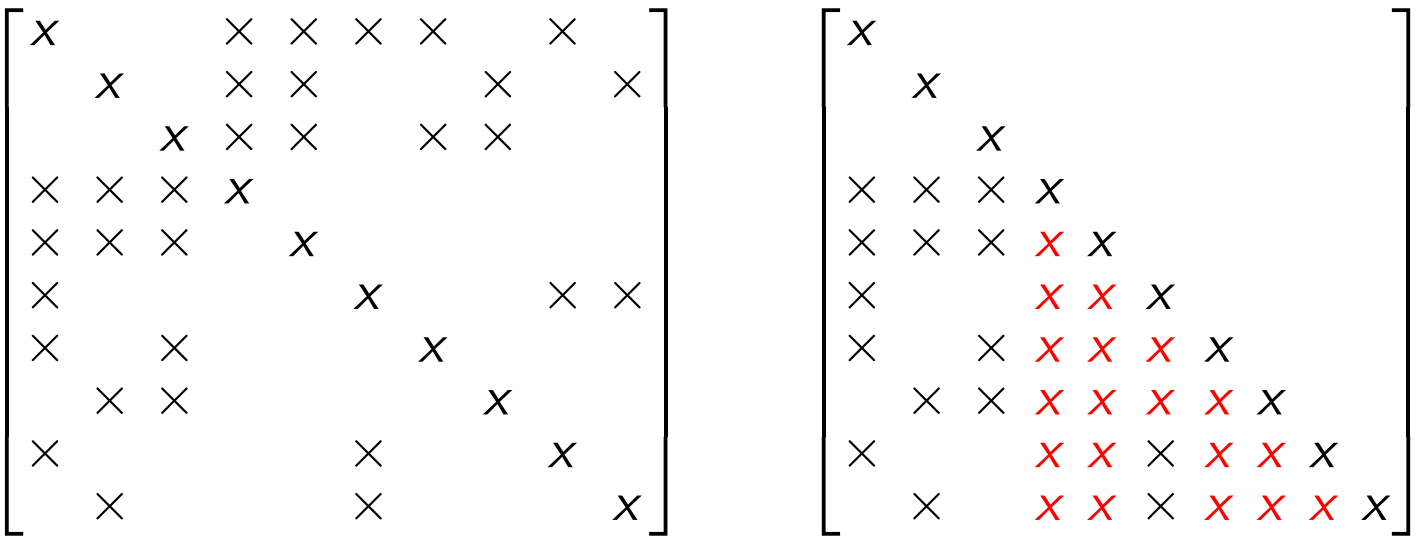
Heuristics for avoiding fill-in

1. General principles of sparse direct solvers
2. Heuristics for avoiding fill-in: Matrix Reorderings
3. Combination with Maximum-Transversal Permutation
4. Software

Heuristics for avoiding fill-in

1. For reducible matrices
   - DulmageMendelsohn (Matlab's DMPERM): permute A to block-triangular form (if possible)
2. Local Greedy-algorithms
   - Reverse Cuthill-McKee (Matlab's SYMRCM): reduce bandwidth of A
   - Approximate Minimum Degree (Matlab's SYMAMD): local minimization of fill-in
3. Global, Domain-Decomposition inspired reordering
   - Nested Dissection: divide and conquer heuristic

A sample matrix & fill-in



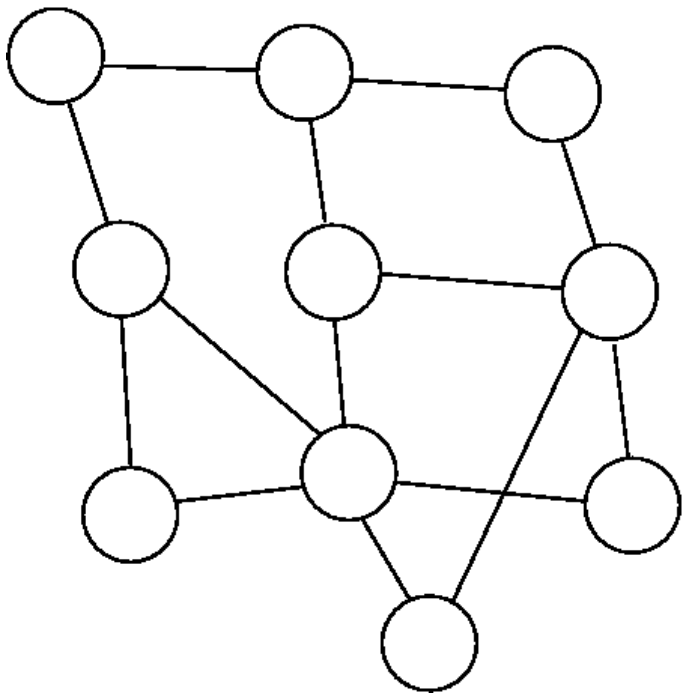
- Elimination of an unknown ⇒ clique among neighbors
- Matrix looks like lots of 'up arrows' ⇒ submatrix $A(4:10, 4:10)$ *completely filled in*!
- Can we do better than 19 fill-ins?

Reverse Cuthill-McKee algorithm I

- The Cuthill-McKee algorithm was designed to reduce the bandwidth of sparse symmetric matrices.
  REF.: E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In: Proc. 24th Nat. Conf. ACM, pp. 157–172, 1969.
- The reverse Cuthill-McKee algorithm (RCM) is the same algorithm but with the resulting index numbers reversed. In general, a better solution (less fill-in).
- The reverse Cuthill-McKee algorithm algorithm is often (but need not be) used in connection with the skyline storage scheme. (stores only the entries from the first nonzero entry to the last nonzero entry)

Reverse Cuthill-McKee algorithm II

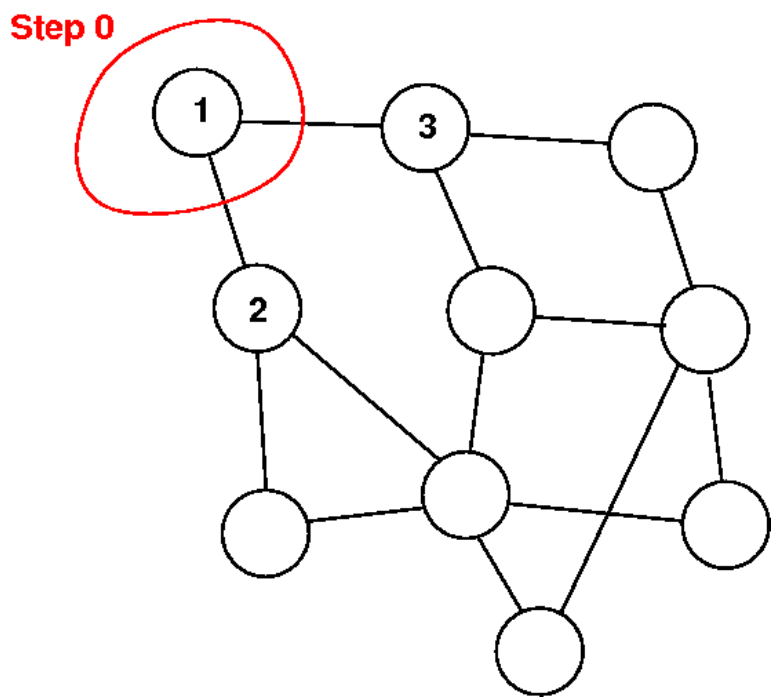
- **Step 0:** Choose a vertex with minimum degree
  ($\Rightarrow$ Greedy: corresponds often to locally minimum fill-in).



The degree of a vertex in a graph is the number of connecting edges.

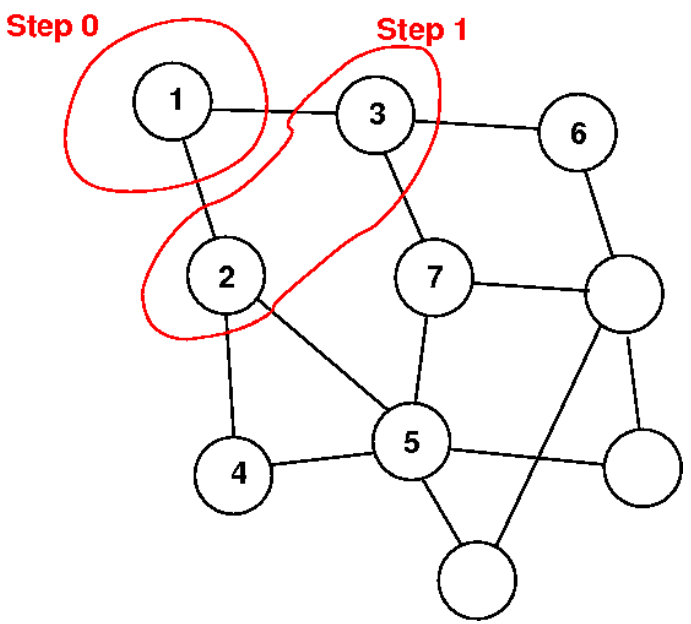
Reverse Cuthill-McKee algorithm III

- Step 0: Choose a vertex with minimum degree.
- **Step 1:** Number all neighbors of vertex 1 by increasing degree.



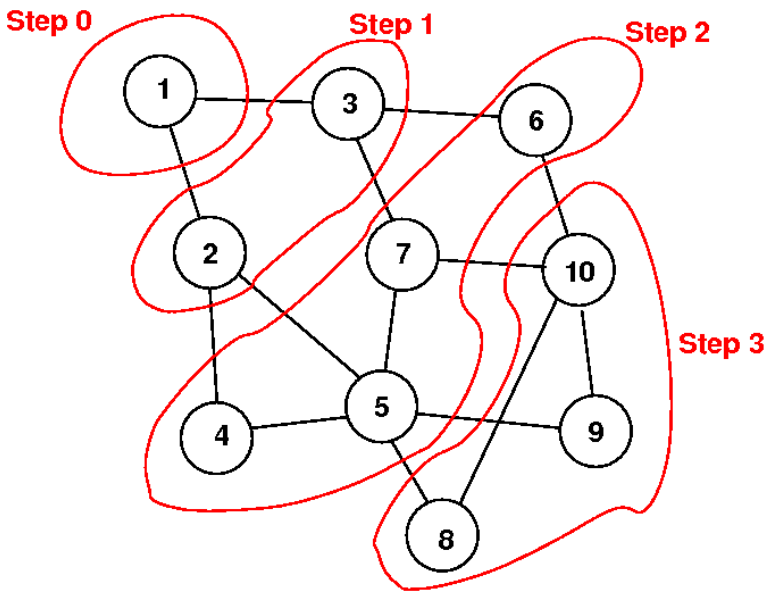
Reverse Cuthill-McKee algorithm IV

- Step 0: Choose a vertex with minimum degree.
- Step 1: Number all neighbors of vertex 1 by increasing degree.
- **Step 2:** For *each* vertex from step 1, number all its remaining neighbors by increasing degree. etc.



Reverse Cuthill-McKee algorithm V

$$\begin{bmatrix}
1 & \times & \times & & & & & & & \\
\times & 2 & & \times & \times & & & & & \\
\times & & 3 & & & \times & \times & & & \\
 & \times & & 4 & \times & & & & & \\
 & \times & & \times & 5 & & \times & \times & \times & \\
 & & \times & & & 6 & & & & \times \\
 & & \times & & \times & & 7 & & & \times \\
 & & & & \times & & & 8 & & \times \\
 & & & & \times & & & & 9 & \times \\
 & & & & & \times & \times & \times & \times & 10
\end{bmatrix}$$



'Reverse' Cuthill-McKee (RCM, George 1971, Jennings 1977):
Empirical finding: bandwidth often better when order reversed

- What is the fill-in incurred in CM and RCM?
- Try to find out yourself!

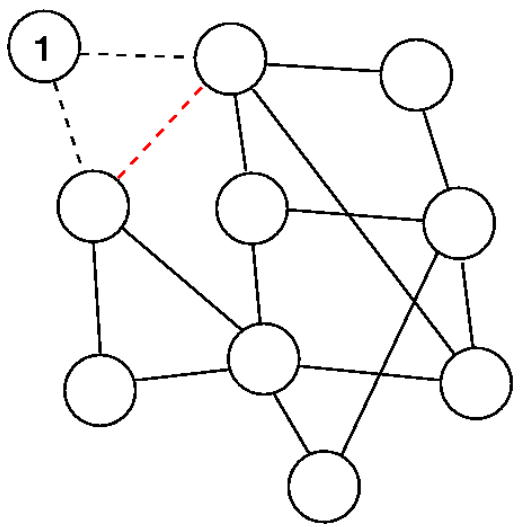
Minimum Degree Algorithm I

- The Minimum Degree Algorithm is a heuristic for finding a permutation P such that PAP^T has fewer nonzeros in its factorization than A.
- A version of the minimum degree algorithm was implemented in the MATLAB function `symamd`, but has now been superseded by a symmetric approximate multiple minimum degree function `symamd`, which is faster.
- The algorithm is greedy in that it trys to do the best in each iteration step. It selects the sparsest pivot row/column as the next pivot row.

Minimum Degree Algorithm II

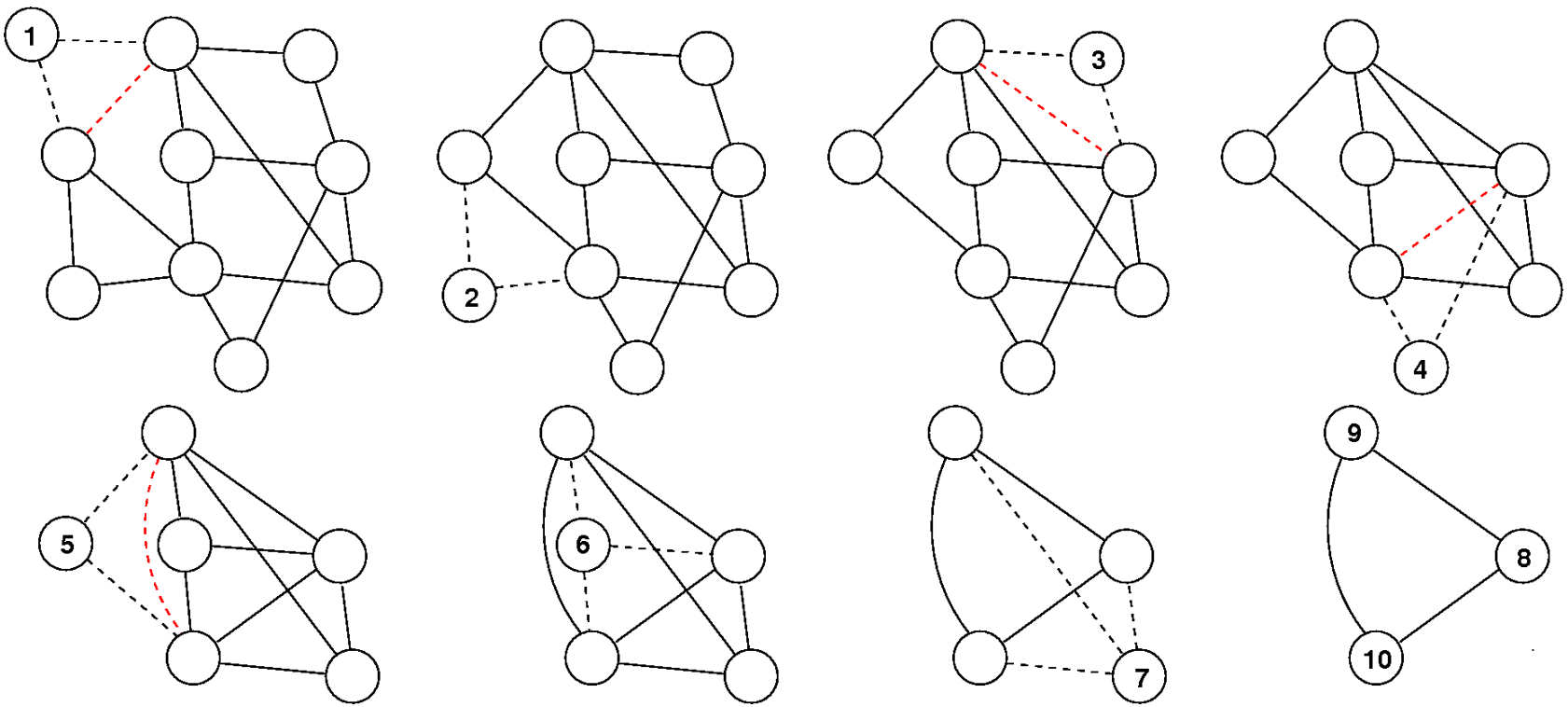
Include fill-in and update the vertex degree after elimination of a variable.

- Step 1: Choose among n vertices one with minimum degree as 1. Connect all neighbors of 1 with each other to a 'clique,' (fill-in). Remove vertex 1 and all its edges from graph.
- Step 2: repeat with reduced graph of $n-1$ vertices, etc.

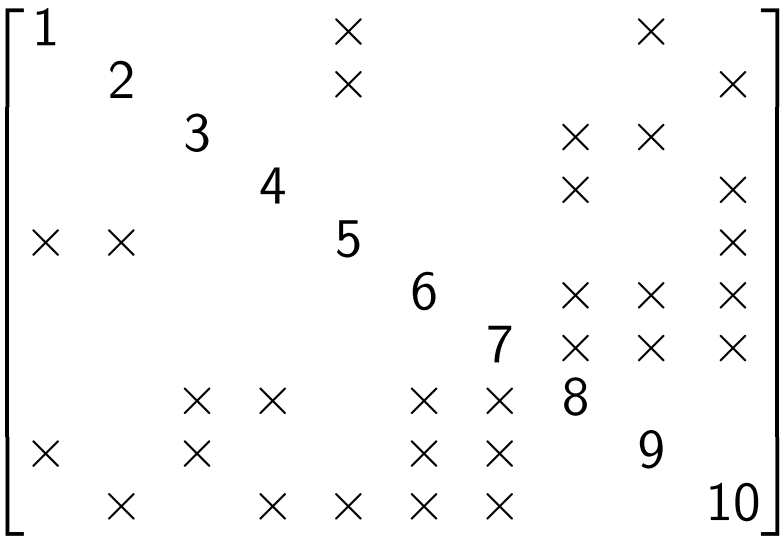


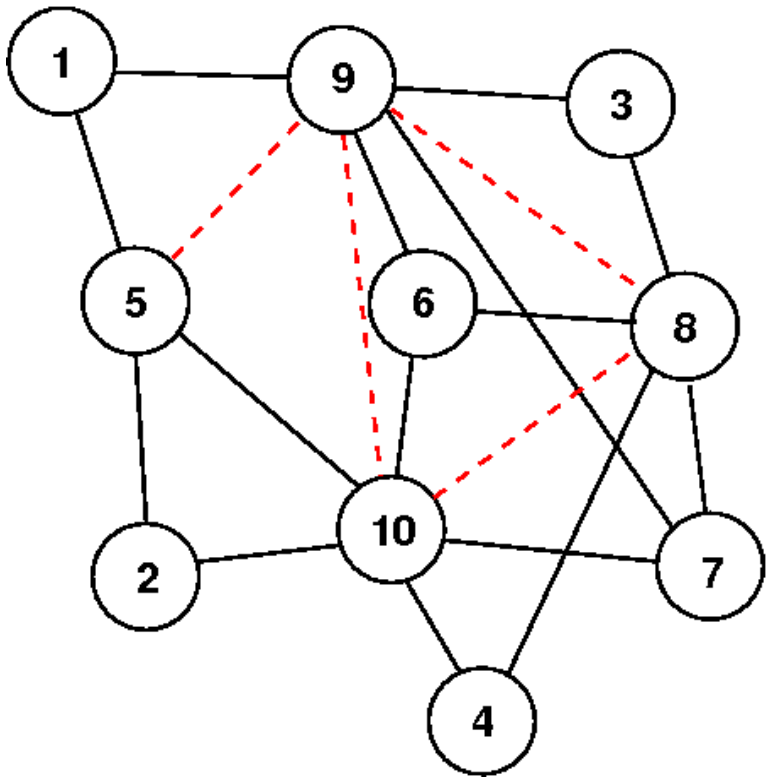
Minimum Degree Algorithm III

Complete execution for sample graph:



Minimum Degree Algorithm IV

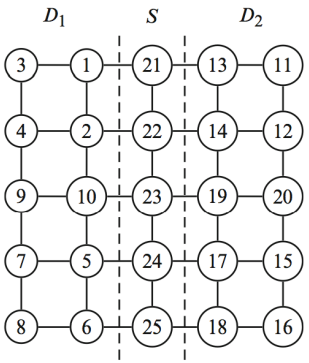




Nested Dissection I

- The nested dissection matrix reordering strategy tries to reorder the matrix **A**, so that the fill-in is kept within the certain matrix blocks in the factorisation.
- The graph of **A** is dissected into smaller subgraphs by separators,
- A separator **S** between two subgraph D_1 and D_2 is the set of vertices containing all paths between between D_1 and D_2.
- The rationale for making this dissection is that there can not be any fill-in L_{ij} with vertex i in D_1 and j in D_2.

Nested Dissection II



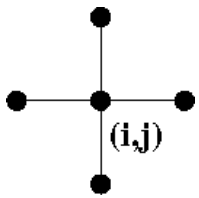
$$L = \begin{bmatrix} L_{11} & 0 & 0 \\ 0 & L_{22} & 0 \\ L_{S1} & L_{S2} & L_{SS} \end{bmatrix}$$

where the diagonal blocks $L_{ii}, i = 1, 2$, and L_{SS} stems from the vertex sets D_i and S, and the off-diagonal blocks L_{Si} stems from the edge intersection of these sets.
Due to the nested dissection any fill-in must occur in L_{ii} or L_{SS}.

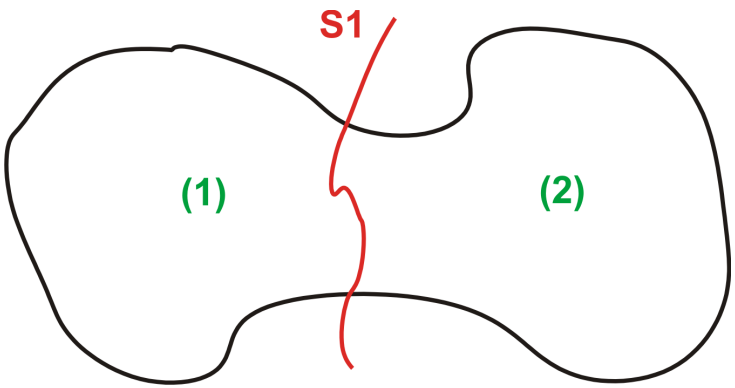
Domain Decomposition 1/3: Nested Dissection

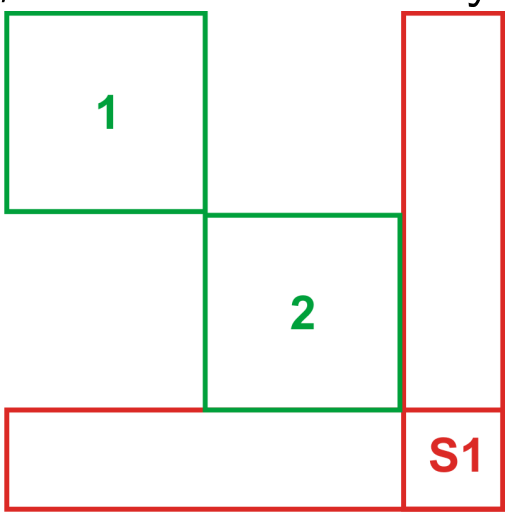
Approximation of 2D-Poisson equation using FD:

$$-\Delta u(\mathbf{x}_{i,j}) = f(\mathbf{x}_{i,j}),\ \ 0 < i,j < n{+}1$$



Numbering: first the two sub-domains, then the boundary points

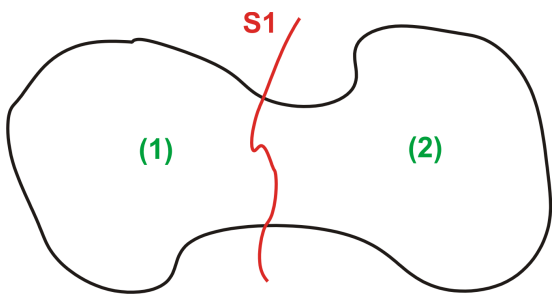


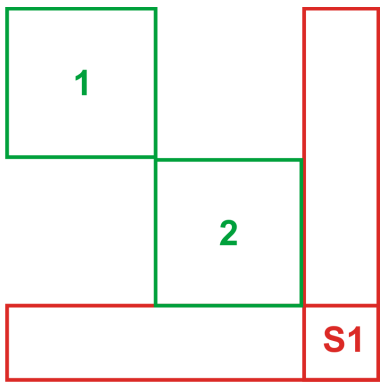


Only fill-in *within* the blocks of the 'block-bordered' matrix!

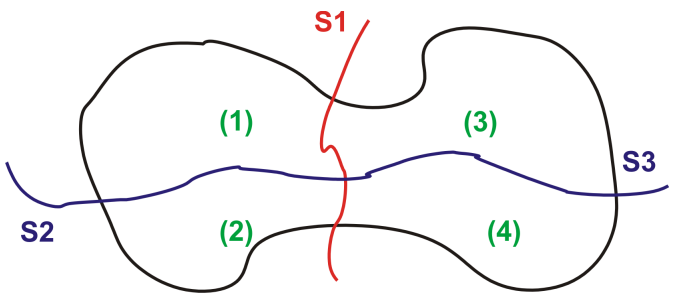
Domain Decomposition 2/3: Nested Dissection (recursive)

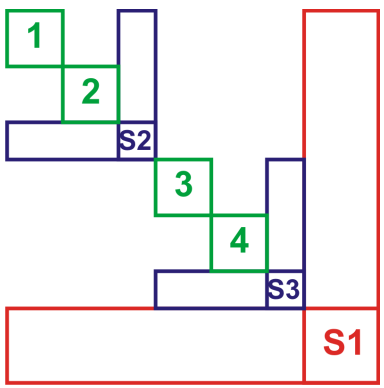
Numbering: first the two sub-domains, then the boundary points





Recursively:

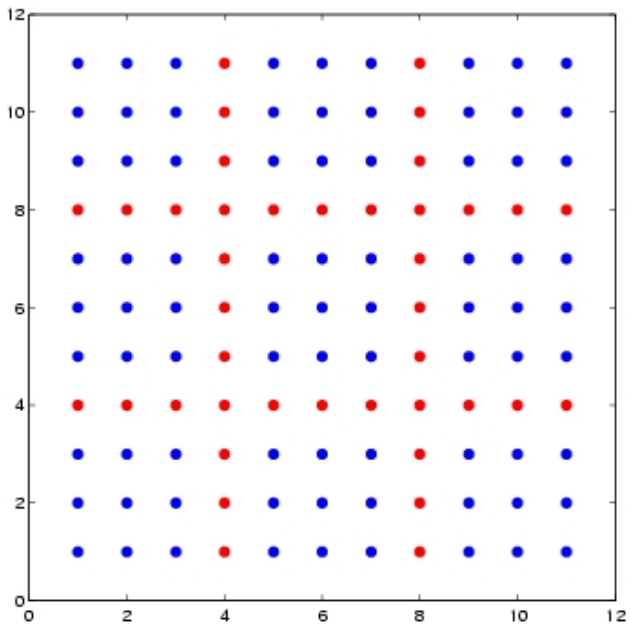


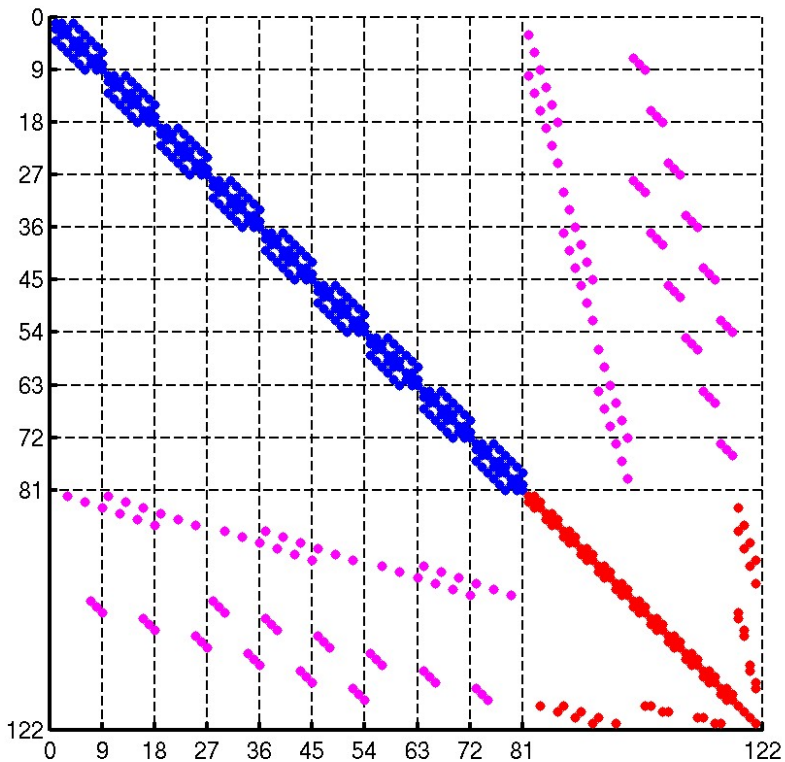


Only fill-in *within* the blocks of the 'block-bordered' matrix!

Domain Decomposition 3/3: Multisection

Generalization of Nested Dissection:





Numbering: first all sub-domains and then the boundary points

Combination with Maximum-Transversal Permutation

- So far: symmetric permutations to avoid fill-in
- Observation: A and PAP^T have the *same* entries on their diagonals (only in different positions)
- Strategy (for unsymmetric A):
  1. Unsymmetric permutation $A' = AQ$ to obtain large diagonal entries (Koster & Duff 1999)
  2. Symmetric permutation $A'' = PA'P^T$ for small fill-in

Idea: the larger the diagonal ('transversal'), the less one (probably) has to resort to pivoting

Remember: no pivoting if A is strictly diagonally dominant!

Algorithm: Matching of columns with rows (Koster & Duff)

Matching algorithm for nonsymmetric matrices I

- Let $A \in \mathbb{R}^{n \times n}$ be a general matrix.
- The nonzero elements of A define a graph with edges $\mathcal{E} = \{(i,j) : a_{ij} \neq 0\}$ of ordered pairs of row and column indices.
- A subset $\mathcal{M} \subset \mathcal{E}$ is called a matching, or transversal, if every row index i and every column index j appears at most once in $\mathcal{M}$.
- A matching is called perfect if its cardinality is n.
- For a nonsingular matrix, at least one perfect matching exists and can be found with known algorithms.

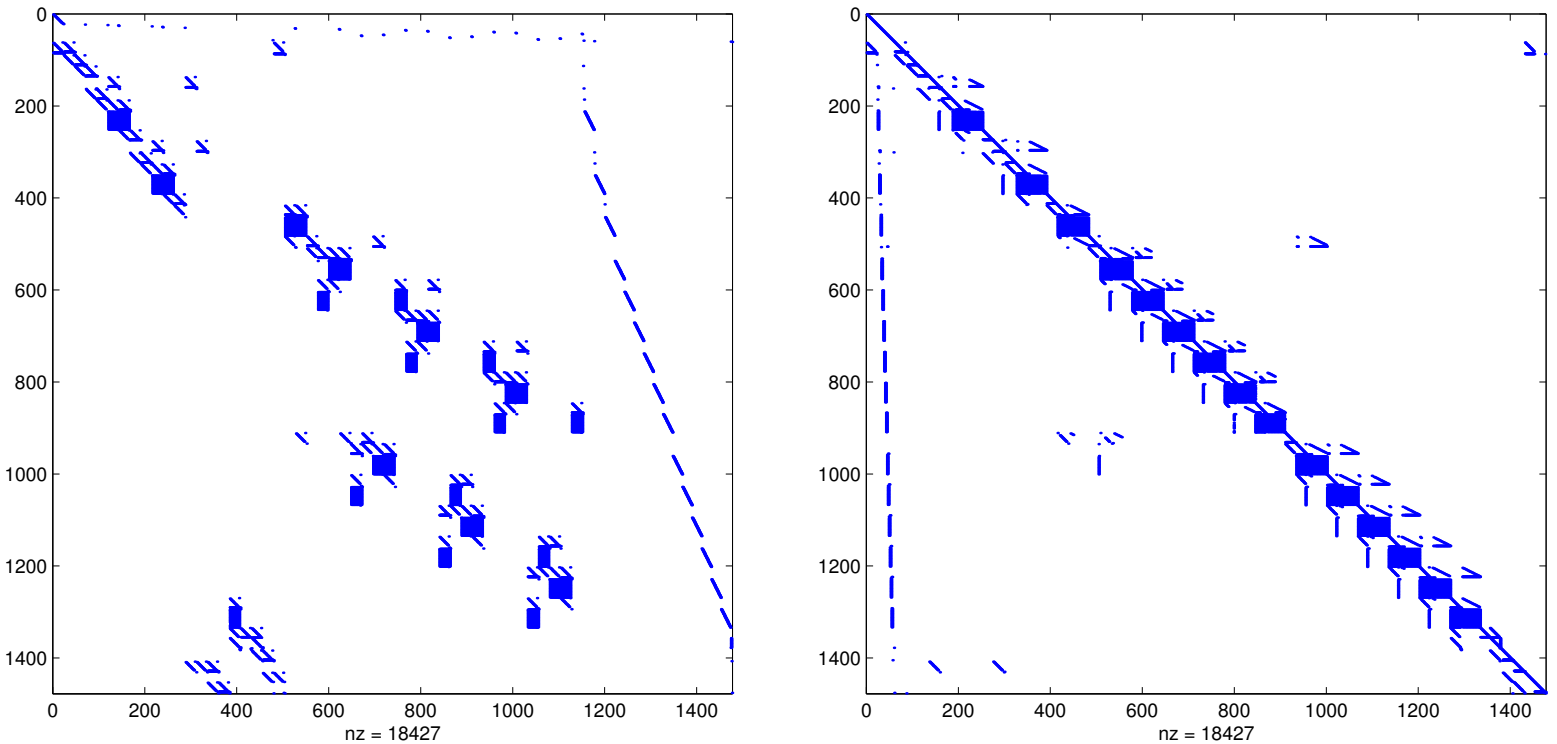
Matching algorithm for nonsymmetric matrices II

- With a perfect matching $\mathcal{M}$, it is possible to define a permutation matrix $P_{\mathcal{M}} = (p_{ij})$ with

$$p_{ij} = \begin{cases} 1 & (j,i) \in \mathcal{M}, \\ 0 & \textit{otherwise}. \end{cases}$$

  By consequence, $P_{\mathcal{M}}A$ has nonzero elements on its diagonal.
- This takes only the nonzero structure of the matrix into account.
- Other approaches maximize the diagonal values in some sense.
- One could try to find a matching such that the product of the diagonal values of $P_{\mathcal{M}}A$ is maximal.
- Problem in combinatorial optimization.

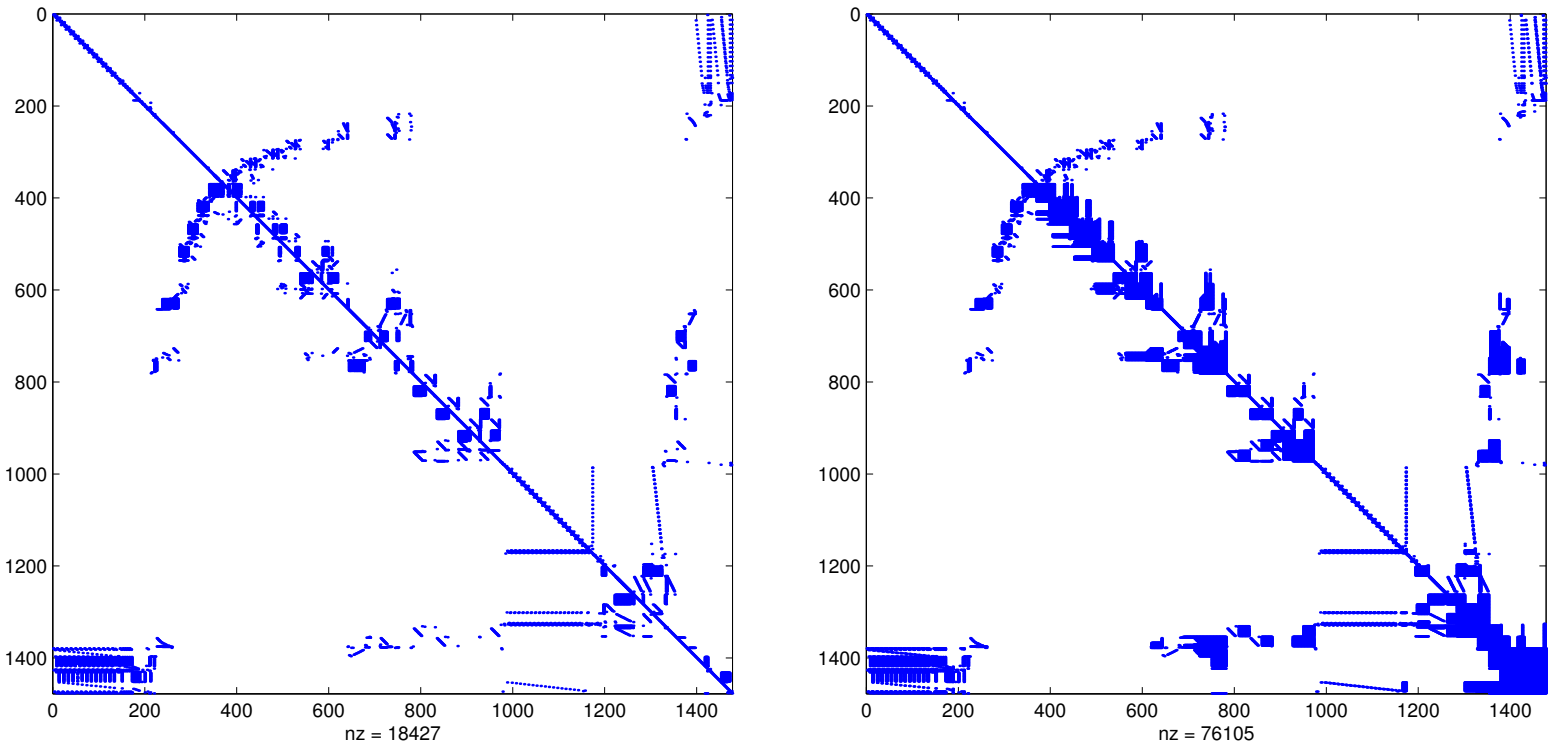
Practical effect of Maximum-Transversal Permutation



- Left: Original A
- Right: Column-permuted version $A' = AQ$

(Data from: Amestoy & L'Excellent: *Direct methods for sparse linear algebra*, CEA-EDF-INRIA School on High Performance Scientific Computing, 2006)

Symmetric reordering (AMD) and factorization



- Left: symmetric permutation $A'' = PA'P^T$
- Right: factorization $A'' = LU$

Available Software

- Sequential:
  - UMFPACK (Davis, U. Florida): indep. & in Matlab
  - SuperLU (Li, Berkeley)
- Parallel:
  - SuperLU_MT (Li, Berkeley): shared-memory, Multi-Threaded
  - SuperLU_DIST (Li, Berkeley): distributed memory
  - MUMPS (Amestoy, L'Excellent, France): distributed memory

References

- T. Davis: *Summary of available software for sparse direct methods*.
- N. Gould, J. Scott, Y. Hu: *A Numerical Evaluation of Sparse Direct Solvers for [..] Symmetric Linear Systems of Equations*, 2007.

Use in Scientific Environments

Direct 'Sparse' Linear Solvers are important part of/integratable with numerical frameworks and libraries:

- Matlab: integrated or via *mex*-interface
- Mathematica: integrated
- Trilinos: interfaced via Amesos-Package
- Portable, Extensible Toolkit for Scientific Computation (PETSc): interfaces within library of linear solvers

Note: *Incomplete* Factorizations of sparse matrices can also make good *preconditioners* for iterative linear solvers.

Exercise-5:

http://www.inf.ethz.ch/personal/tulink/FEM13/Exercise5.pdf