4.3.6. Rectangular Storage Scheme by Pressing from the Right

$$\left(\begin{array}{ccccc|l} 1 & 0 & 2 & 0 & 0 & \\ 3 & 4 & 0 & 5 & 0 & \\ 0 & 6 & 7 & 0 & 8 & \leftarrow \text{pressing from right} \\ 0 & 0 & 9 & 10 & 0 & \\ 0 & 0 & 0 & 11 & 12 & \end{array}\right)$$

gives

$$\text{COEF} = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 0 \\ 11 & 12 & 0 \end{pmatrix} \qquad \text{JCOEF} = \begin{pmatrix} 1 & 3 & * \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & * \\ 4 & 5 & * \end{pmatrix}$$

Storage: n, $n \cdot nl$ integer and float ($nl :=$ nnz of longest row)

Rectangular Storage Scheme by Pressing from the Right (cont.)

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
   for j = 1 : nl
      c_i = c_i + COEF(i,j) * b(JCOEF(i,j));
   end
end
```

This format was used in ELLPACK (package of subroutines for elliptic PDEs).

4.3.7. Jagged Diagonal Form

Prestep: Sort rows after their length. Long rows first.

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix} \Rightarrow PA = \begin{pmatrix} \boxed{3} & \boxed{4} & 0 & \boxed{5} & 0 \\ 0 & \boxed{6} & \boxed{7} & 0 & \boxed{8} \\ \boxed{1} & 0 & \boxed{2} & 0 & 0 \\ 0 & 0 & \boxed{9} & \boxed{10} & 0 \\ 0 & 0 & 0 & \boxed{11} & \boxed{12} \end{pmatrix} \begin{matrix} \} \text{ Length 3} \\ \\ \\ \} \text{ Length 2} \\ \\ \end{matrix}$$

- Values of PA: DJ = (3 6 1 9 11 | 4 7 2 10 12 | 5 8|)
- Column indices: JDIAG = (1 2 1 3 4 | 2 3 3 4 5 | 4 5|)
- Pointer to beginning of jth diagonal: IDIAG = (1 6 11 13)

Jagged Diagonal Form (cont.)

NDIAG := number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Jagged Diagonal Form (cont.)

NDIAG := number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for j = 1 : NDIAG
   for i = 1 : IDIAG(j+1) - IDIAG(j)
      k = IDIAG(j) + i - 1;
      c_i = c_i + DJ(k) * b(JDIAG(k));
   end
end
```

Jagged Diagonal Form (cont.)

NDIAG := number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for j = 1 : NDIAG
   for i = 1 : IDIAG(j+1) - IDIAG(j)
      k = IDIAG(j) + i - 1;
      c_i = c_i + DJ(k) * b(JDIAG(k));
   end
end
```

- Always start with row 1.
- More operations on neighboring data.
- Less indirect addressing.
- Pre-permutation changes only rows. Can be done implicitly.

Survey on Sparse Storage Formats

Coordinate form and CSR traditional way to specify sparse matrix in MATLAB.

	global int	idx int	value floats
Coordinate	2	$2 \cdot \text{nnz}$	nnz
CSR	2	$n + \text{nnz} + 1$	nnz
CSR with Extract. Diag.	2	$\text{nnz} + 1$	$\text{nnz} + 1$
Diagonalwise	2	nd	$n \cdot nd$
Rectang. with Pressing	1	$n \cdot nl$	$n \cdot nl$
Jagged Diagonal	2	$\text{nnz} + nd$	nnz

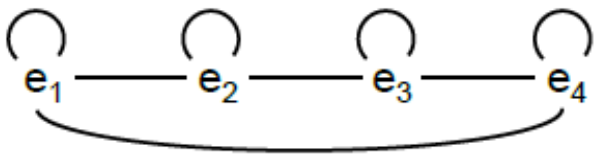
4.4. Sparse Matrices and Graphs

4.4.1. Graph $G(A)$ for symmetric positive definite (SPD)

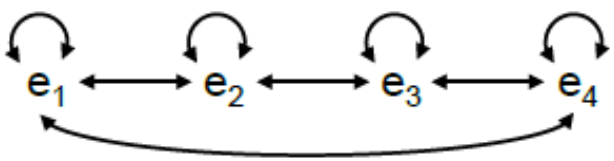
$A = A^T > 0$

$n \times n$-matrix: vertices $e_1, \ldots, e_n$ with edges (e_i, e_k) for $a_{ik} \neq 0$, undirected Graph

$$A = \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix} \rightarrow G(A) :$$



G(A) as directed graph:



Adjacency Matrix for $G(A)$ or A

- Can be obtained directly by replacing in A each nonzero by 1.

$$\mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Adjacency Matrix for $G(A)$ or A

- Can be obtained directly by replacing in A each nonzero by 1.

$$\mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

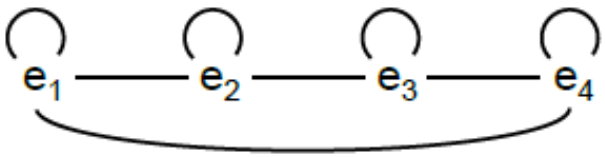
- Symmetric permutations of A in the form PAP^T change the ordering of the rows and columns of A simultaneously.
- Therefore, the graph of PAP^T can be obtained by the graph of A by renumbering the vertices.

Matrix A with graph G(A) (cont.)

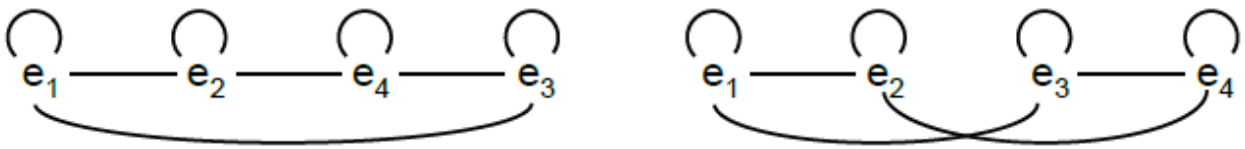
Symmetric permutation PAP^T with graph $G(PAP^T)$.

Example: P permutation that changes $3 \leftrightarrow 4$:

$G(A)$



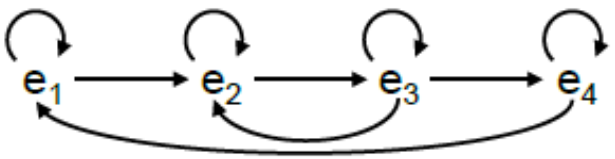
$G(PAP^T)$



4.4.2. Matrix A nonsymmetric, $G(A)$ directed

$$A = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

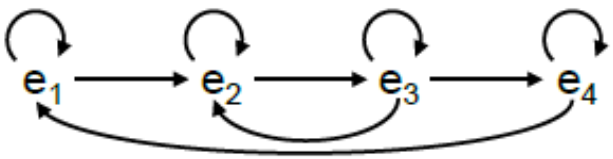
G(A):



4.4.3. Matrix A nonsymmetric, $G(A)$ directed

$$A = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

G(A):



Questions:
How can we characterize “good“ sparsity patterns?

”good”: Gaussian elimination can be reduced to smaller subproblems or produces no (or small) fill-in, or is easy to parallelize.

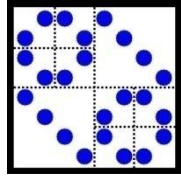
Block pattern, banded pattern, envelope

Block Diagonal Pattern

New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Banded Pattern



$$A = \begin{pmatrix} a_{11} & \cdots & a_{1p} & & & & \\ \vdots & \ddots & & \ddots & & & \\ a_{q1} & & \ddots & & \ddots & & \\ & \ddots & & \ddots & & a_{n-p+1,n} & \\ & & \ddots & & \ddots & \vdots & \\ & & & a_{n,n-q+1} & \cdots & a_{nn} & \end{pmatrix}$$

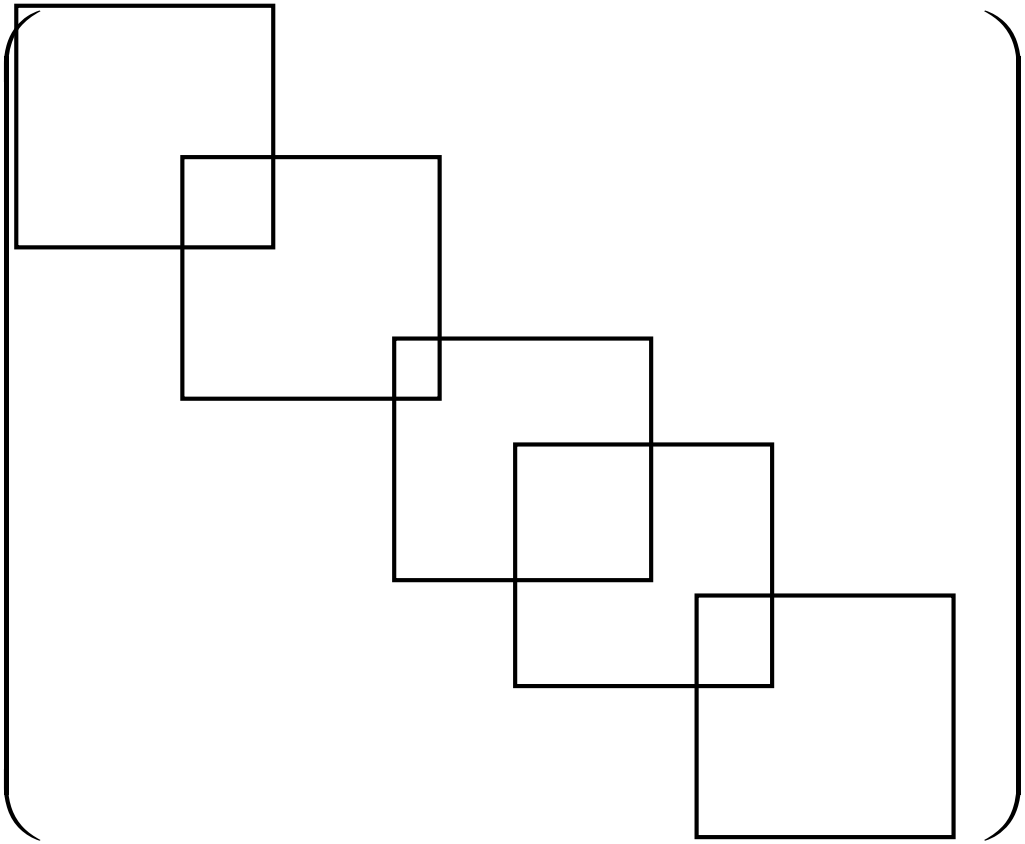
Gauss Elimination without pivoting preserves the sparsity pattern

$$L = \begin{pmatrix} l_{11} & & & & & & \\ \vdots & \ddots & & & & & \\ l_{q1} & & \ddots & & & & \\ & \ddots & & \ddots & & & \\ & & \ddots & & \ddots & & \\ & & & l_{n,n-q+1} & \cdots & l_{nn} & \end{pmatrix} \qquad U = \begin{pmatrix} u_{11} & \cdots & u_{1p} & & & & \\ & \ddots & & \ddots & & & \\ & & \ddots & & \ddots & & \\ & & & \ddots & & u_{n-p+1,n} & \\ & & & & \ddots & \vdots & \\ & & & & & u_{nn} & \end{pmatrix}$$

With pivoting the bandwidth in U grows, but remains <= p+q.

Overlapping Block Diagonal

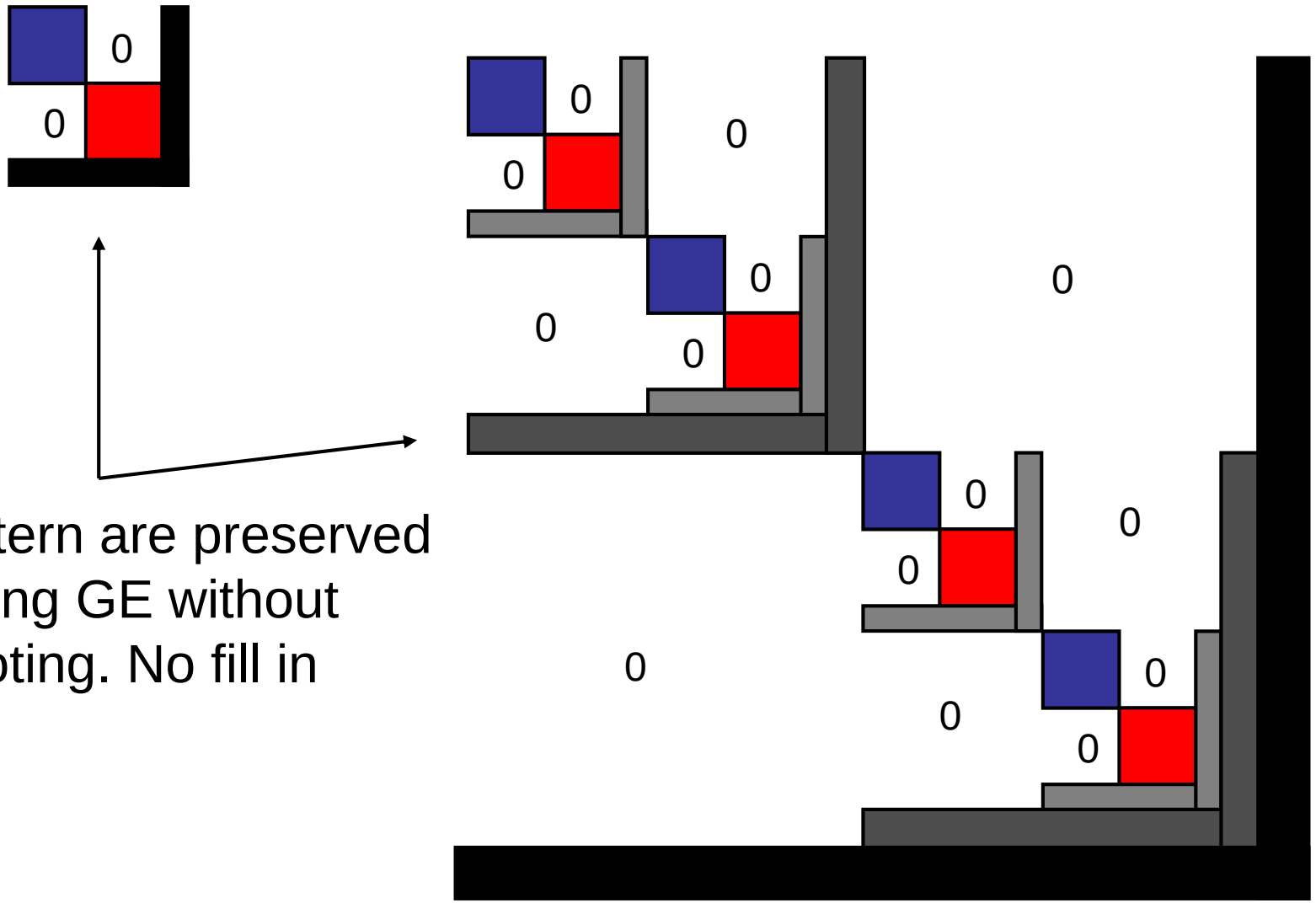
$A =$



Pattern is preserved by Gaussian Elimination (in case of restricted pivoting).

Dissection Form

Nested (recursive) dissection:



Pattern are preserved during GE without pivoting. No fill in

Schur Complement Reduction

Write matrix B in terms of smaller submatrices:

$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \cdot \begin{pmatrix} B_1^{-1} & D \\ 0 & S^{-1} \end{pmatrix} = \begin{pmatrix} I & B_1 D + B_2 S^{-1} \\ B_3 B_1^{-1} & B_3 D + B_4 S^{-1} \end{pmatrix} \overset{!}{=} \begin{pmatrix} I & 0 \\ * & I \end{pmatrix}$$

To satisfy this equation we have to set:

$$B_1 D + B_2 S^{-1} = 0 \Rightarrow D = -B_1^{-1} B_2 S^{-1}$$

$$B_3 D + B_4 S^{-1} = I \Rightarrow I = -B_3 B_1^{-1} B_2 S^{-1} + B_4 S^{-1}$$

$$\Rightarrow \underbrace{S = B_4 - B_3 B_1^{-1} B_2}_{\text{S Schur Complement}} \quad and \quad D = -B_1^{-1} B_2 S^{-1}$$

$$B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B_3 B_1^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} B_1 & B_2 \\ 0 & S \end{pmatrix}$$

Therefore, solving linear system in B is reduced to solving two smaller linear systems, one in B_1 and the other in the Schur complement S.

B sparse → B_1 also sparse, but S usually dense!

Example: Schur complement and dissection form:

$$A = \left(\begin{array}{cc|c} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ \hline G_1 & G_2 & A_3 \end{array}\right)$$

Schur complement:

$$S = A_3 - \begin{pmatrix} G_1 & G_2 \end{pmatrix} \cdot \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} =$$

$$= A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2$$

Direct derivation of Schur complement:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \Rightarrow \begin{array}{lllll} A_1 x_1 + & & F_1 x_3 & = & b_1 \\ & A_2 x_2 + & F_2 x_3 & = & b_2 \\ G_1 x_1 + & G_2 x_2 + & A_3 x_3 & = & b_3 \end{array}$$

$$x_1 = A_1^{-1} b_1 - A_1^{-1} F_1 x_3$$

$$\Rightarrow$$

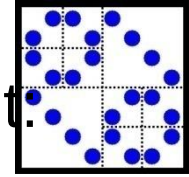
$$x_2 = A_2^{-1} b_2 - A_2^{-1} F_2 x_3$$

$$\Rightarrow (G_1 A_1^{-1} b_1 - G_1 A_1^{-1} F_1 x_3) + (G_2 A_2^{-1} b_2 - G_2 A_2^{-1} F_2 x_3) + A_3 x_3 = b_3$$

$$\Rightarrow (A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2) x_3 = b_3 - G_1 A_1^{-1} b_1 - G_2 A_2^{-1} b_2$$

$$\Rightarrow S x_3 = \tilde{b}_3$$

Algorithm for solving Ax=b based on Schur complement:



1. Compute S by using inv(A_1) and inv(A_2)

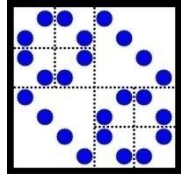
2. Solve $Sx_3 = \tilde{b}_3$

3. Compute x_1 and x_2 by using inv(A_1) and inv(A_2)

The explicit computation of S can be avoided by solving the linear System in S iteratively, e.g. Jacobi, pcg, ….
Then we need only part of S and in every iteration step we have to compute S * intermediate vector.

To achieve fast convergence, a preconditioner (approximation) for S has to be used!

Iterative methods and preconditioning will be subject of later chapters.



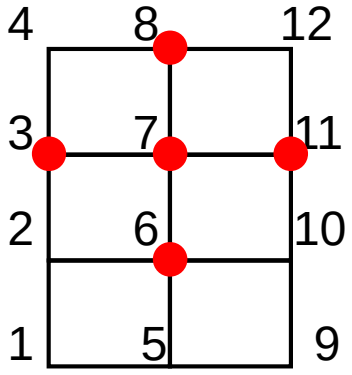
4.4 GE in Graph

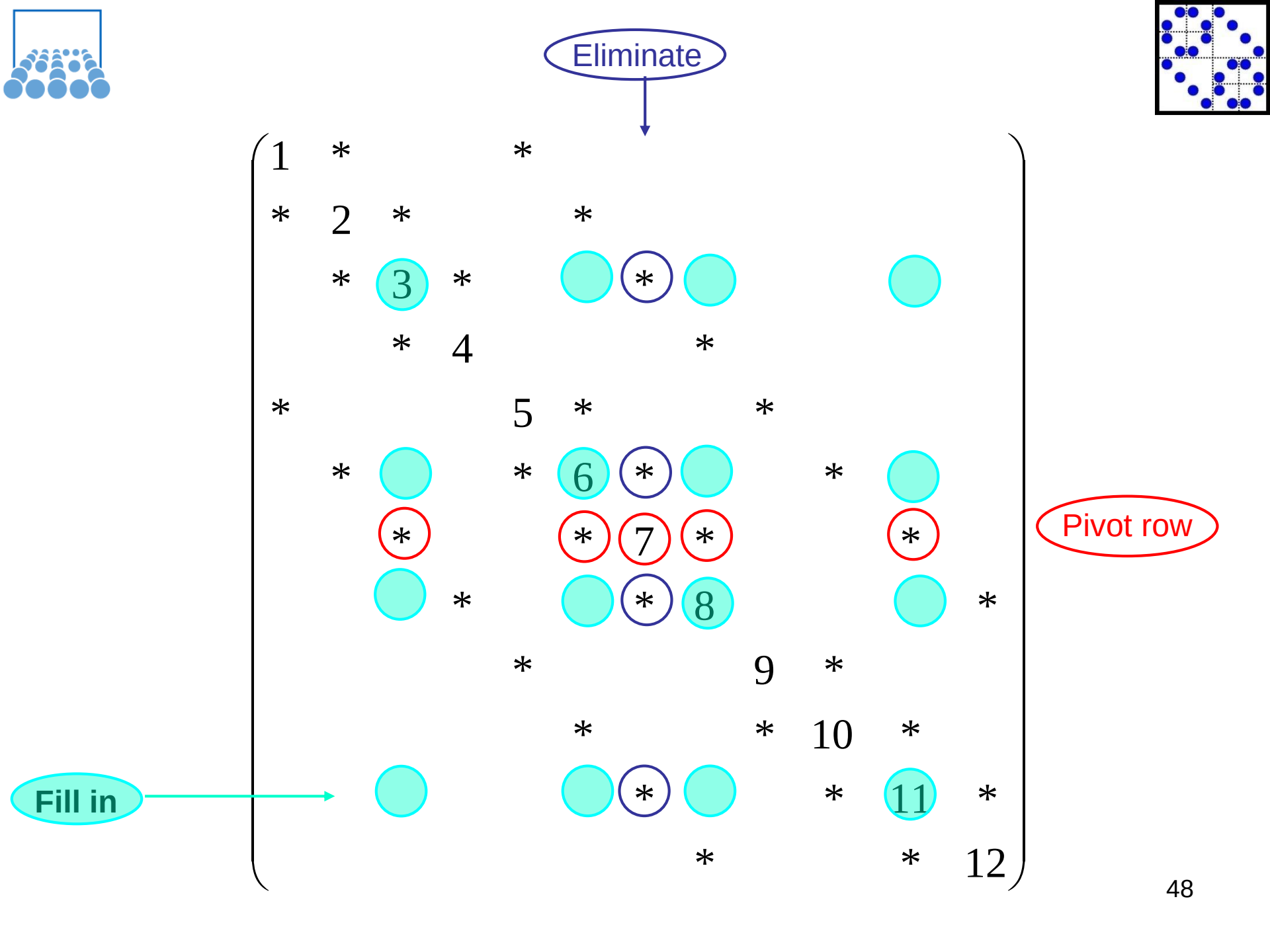
Assume $A = A^T > 0$ spd, symmetric positive definite
(no pivoting necessary)

Consider the graphs related to the Gaussian Elimination step by step.

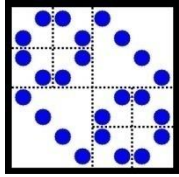
Example: Matrix A with graph G(A)

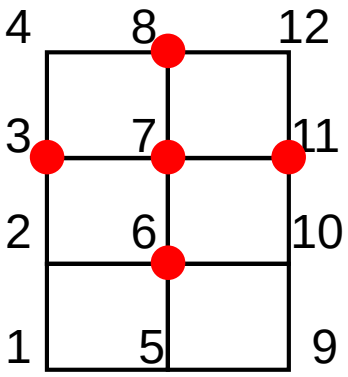
Choose 7 as pivot entry:

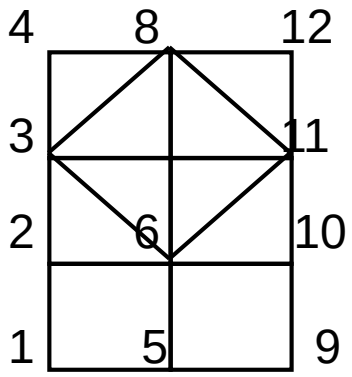



Eliminate
Pivot row
Fill in

Graph after first elimination step:



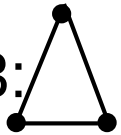
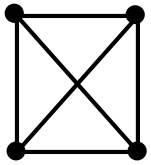
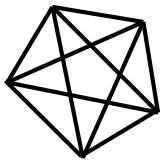




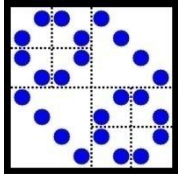
Remove vertex 7 and

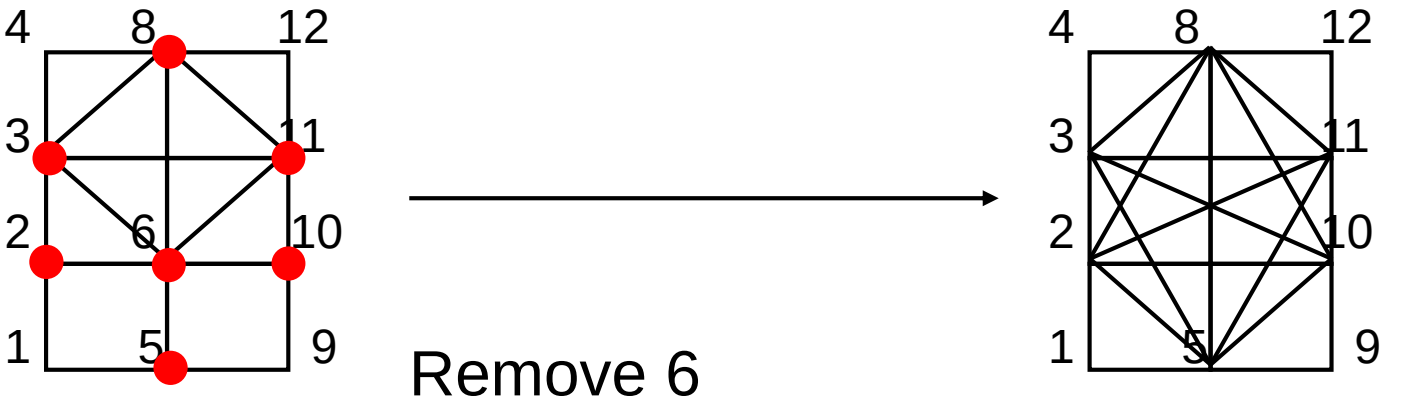
connect all former neighbors of 7 by edges.

New subgraph with vertices 3,6,8, and 11 (former neighbors of 7) is a fully connected graph = clique = dense submatrix

Clique: 1: 2:  3:  4:  5: 

Next elimination step with pivot 6:



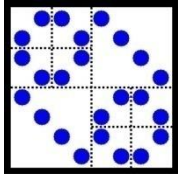


Remove 6

and connect all former neighbors of 6

New clique: 2,3,5,8,10,11

For graph algorithms: Use subroutine package METIS.



Result:

Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs
(in terms of dense subgraphs (matrices) = clique)

Modification of GE:

1. Apply algebraic prestep for GE, determining the graphs related to the elimination matrices A_k in GE.
2. Based on these graphs we can decide
   - whether GE will lead to nearly dense matrices
     (don‘t use GE in this case!)
   - what additional entries will appear during GE
     (prepare the storage)

Algebraic prestep is cheap, can be implemented using cliques.

Block Diagonal Pattern

New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

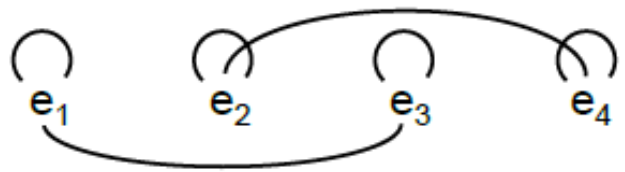
G(A):

Block Diagonal Pattern

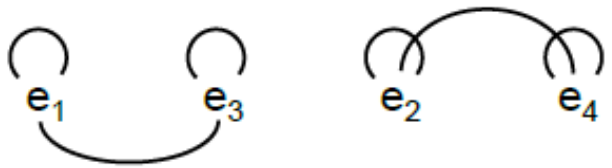
New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

G(A):



$2 \leftrightarrow 3$:



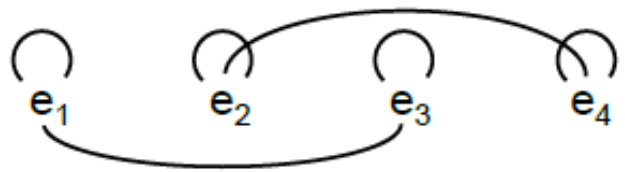
$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

Block Diagonal Pattern

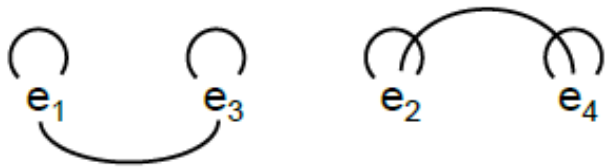
New matrix A!

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

G(A):



$2 \leftrightarrow 3$:



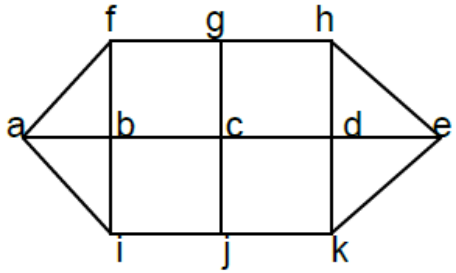
$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

By this permutation, A can be transformed into block diagonal form $\rightarrow$ easy to solve, fully parallel!

$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}^{-1} = \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix}$$

4.5. Reordering

Consider sparse matrix A with graph G(A):

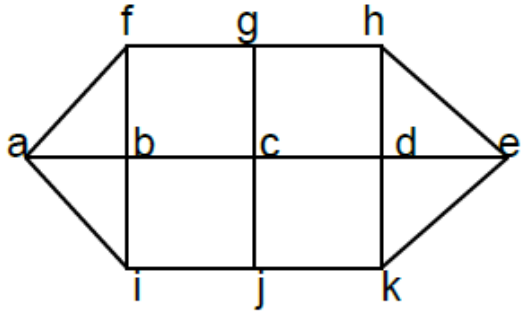


Consider following matrix (in tabular form) with bandwidth 9:

	a	b	c	d	e	f	g	h	i	j	k
a	a	∗				∗			∗		
b	∗	b	∗			∗			∗		
c		∗	c	∗			∗			∗	
d			∗	d	∗			∗			∗
e				∗	e			∗			∗
f	∗	∗				f	∗				
g			∗			∗	g	∗			
h				∗	∗		∗	h			
i	∗	∗							i	∗	
j			∗						∗	j	∗
k				∗	∗					∗	k

4.5.1. Cuthill McKee

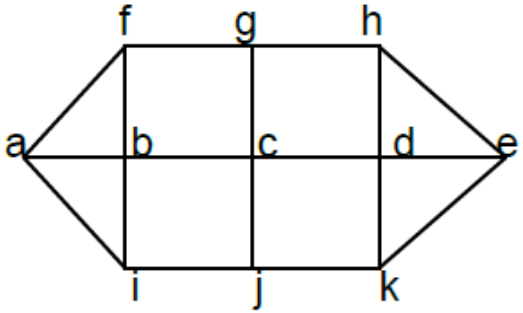
Graph G(A):



Optimal ordering that leads to small bandwidth?

4.5.2. Cuthill McKee

Graph G(A):



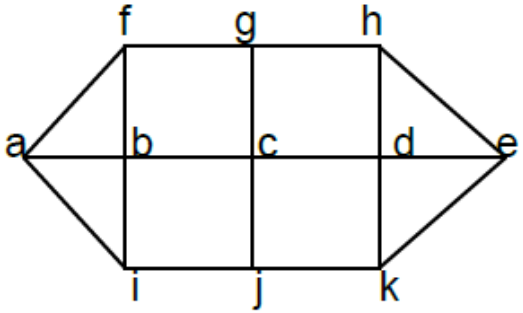
Optimal ordering that leads to small bandwidth?

Level sets, starting with:

$$S_1 := \{a\}$$

4.5.3. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

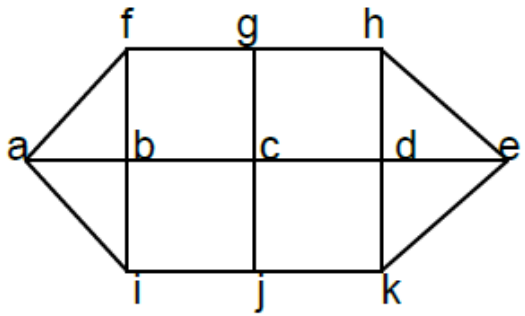
Level sets, starting with:

$S_1 := \{a\}$

$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

4.5.4. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

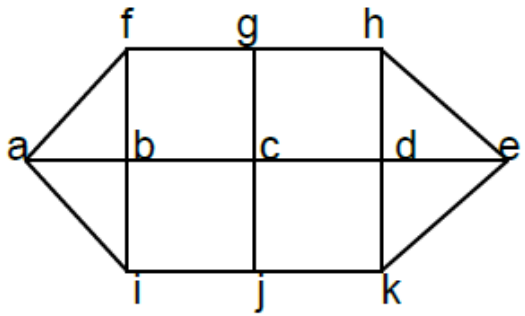
$S_1 := \{a\}$

$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

$S_3 := \{g, c, j\}$, all vertices directly connected with S_2

4.5.5. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

$S_1 := \{a\}$

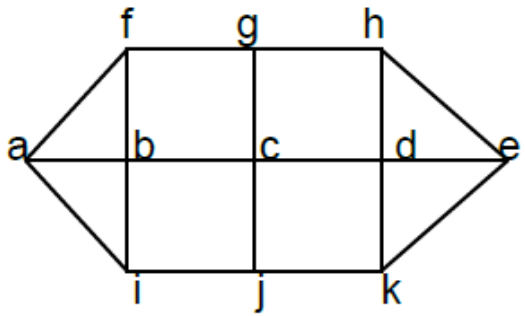
$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

$S_3 := \{g, c, j\}$, all vertices directly connected with S_2

$S_4 := \{h, d, k\}$, all vertices directly connected with S_3

4.5.6. Cuthill McKee

Graph G(A):



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

$S_1 := \{a\}$

$S_2 := \{f, b, i\}$, all vertices directly connected with S_1

$S_3 := \{g, c, j\}$, all vertices directly connected with S_2

$S_4 := \{h, d, k\}$, all vertices directly connected with S_3

$S_5 := \{e\}$

Cuthill McKee (cont.)

1. First ordering by level sets

Cuthill McKee (cont.)

1. First ordering by level sets
2. Inside level sets order the vertices such that first group of indices in S_i are neighbors of first vertex in S_{i-1}

Cuthill McKee (cont.)

1. First ordering by level sets
2. Inside level sets order the vertices such that first group of indices in S_i are neighbors of first vertex in S_{i-1}
3. If there is choice left: number indices with small degree first!

Cuthill McKee (cont.)

1. First ordering by level sets
2. Inside level sets order the vertices such that first group of indices in S_i are neighbors of first vertex in S_{i-1}
3. If there is choice left: number indices with small degree first!

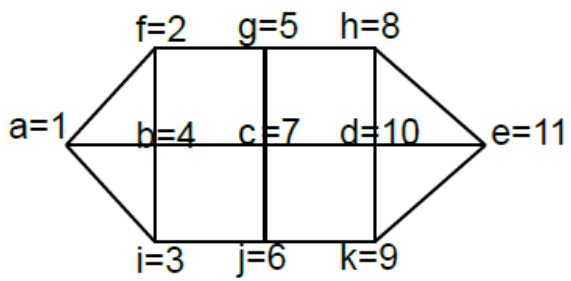
$S_1 := \{a\}$

$S_2 := \{f = 2, i = 3, b = 4\}$, (could also be different!)

$S_3 := \{g = 5, j = 6, c = 7\}$, (as we start with the neighbors of $f = 2$, then i, and then b)

$S_4 := \{h = 8, k = 9, d = 10\}$, (as we start with neighbors of g)

$S_5 := \{e = 11\}$



Cuthill McKee (cont. 2)

- Resulting matrix with small bandwidth 4:

$$\begin{pmatrix}
1 & * & * & * & & & & & & & \\
* & 2 & 0 & * & * & & & & & & \\
* & 0 & 3 & * & 0 & * & & & & & \\
* & * & * & 4 & 0 & 0 & * & & & & \\
 & * & 0 & 0 & 5 & 0 & * & * & & & \\
 & & * & 0 & 0 & 6 & * & 0 & * & & \\
 & & & * & * & * & 7 & 0 & 0 & * & \\
 & & & & * & 0 & 0 & 8 & 0 & * & * \\
 & & & & & * & 0 & 0 & 9 & * & * \\
 & & & & & & * & * & * & 10 & * \\
 & & & & & & & * & * & * & 11
\end{pmatrix}$$

Cuthill McKee (cont. 2)

- Resulting matrix with small bandwidth 4:

$$\begin{pmatrix}
1 & * & * & * & & & & & & & \\
* & 2 & 0 & * & * & & & & & & \\
* & 0 & 3 & * & 0 & * & & & & & \\
* & * & * & 4 & 0 & 0 & * & & & & \\
 & * & 0 & 0 & 5 & 0 & * & * & & & \\
 & & * & 0 & 0 & 6 & * & 0 & * & & \\
 & & & * & * & * & 7 & 0 & 0 & * & \\
 & & & & * & 0 & 0 & 8 & 0 & * & * \\
 & & & & & * & 0 & 0 & 9 & * & * \\
 & & & & & & * & * & * & 10 & * \\
 & & & & & & & * & * & * & 11
\end{pmatrix}$$

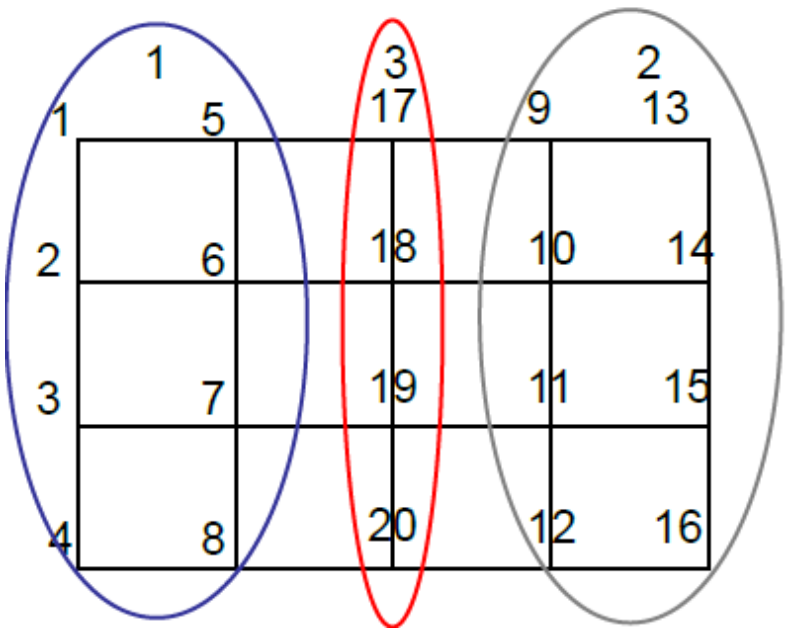
- Often „Reverse Cuthill Mckee“ gives slightly better results: reversing the ordering of Cuthill McKee: $11 \leftrightarrow 1$, $10 \leftrightarrow 2, \ldots$ Choose start as extreme vertex (diameter)!
- This method is far from being optimal and strongly heuristical.

4.5.7. Dissection Reordering

Consider matrix A with graph G(A):

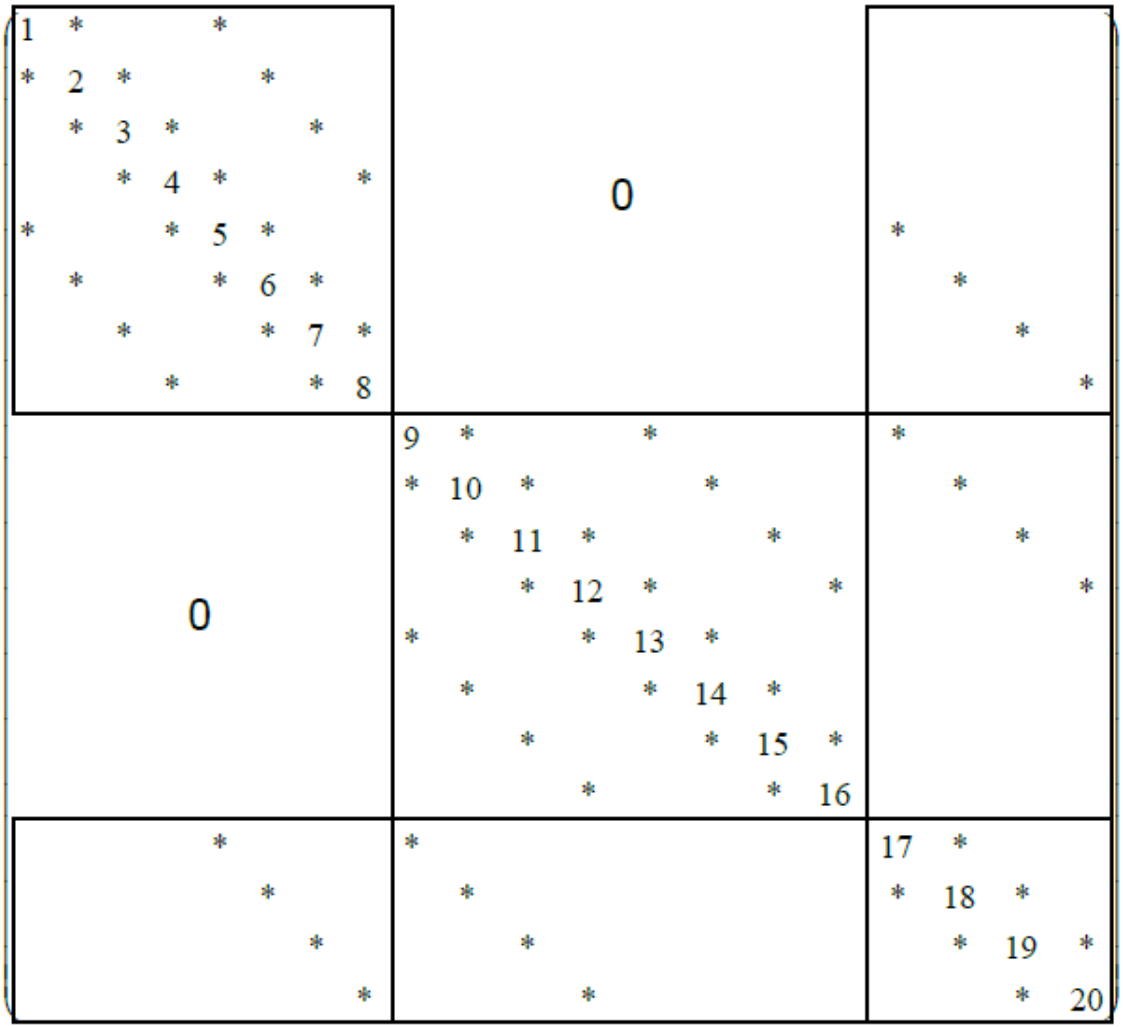
Numbering of unknows in two groups separated by third group.

group:



Dissection Reordering (cont.)

This numbering leads to sparsity pattern:



Dissection Reordering (cont. 2)

- Leads automatically to dissection form in the matrix:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

Dissection Reordering (cont. 2)

- Leads automatically to dissection form in the matrix:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

- Can be solved, e.g. based on Schur complement.

Dissection Reordering (cont. 2)

- Leads automatically to dissection form in the matrix:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

- Can be solved, e.g. based on Schur complement.
- General idea:
  - Cut $G(A)$ by separator between unconnected subgraphs.
  - Separator is numbered last.
  - Repeat recursively $\rightarrow$ Nested dissection form
- Looking for partitioning of the graph with minimum connections!

More on Schur compement in the Chapter Domain Decomposition methods.

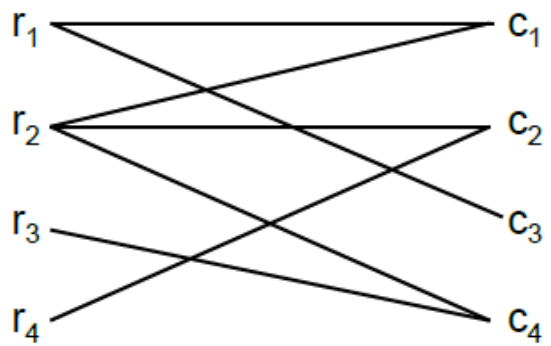
4.5.8. Perfect Matching Reordering

- Find row permutation of A such that elements with largest absolute values are located at diagonal.
- Advantage: No pivoting necessary (also in the indefinite case).

4.5.9. Perfect Matching Reordering

- Find row permutation of A such that elements with largest absolute values are located at diagonal.
- Advantage: No pivoting necessary (also in the indefinite case).
- Describe the sparsity pattern of A by bipartite graph $G = (V_r, V_c, E)$ with $V_r = \{r_1, \ldots, r_n\}$ and $V_c = \{c_1, \ldots, c_n\}$.
- Vertices r_i and c_j are connected by edge $\leftrightarrow$ $a_{i,j} \neq 0$:

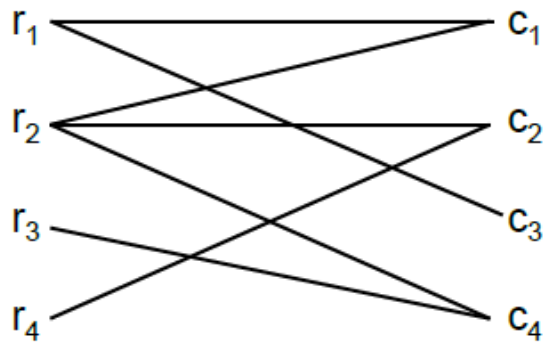
$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & * \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$



4.5.10. Perfect Matching Reordering

- Find row permutation of A such that elements with largest absolute values are located at diagonal.
- Advantage: No pivoting necessary (also in the indefinite case).
- Describe the sparsity pattern of A by bipartite graph $G = (V_r, V_c, E)$ with $V_r = \{r_1, \ldots, r_n\}$ and $V_c = \{c_1, \ldots, c_n\}$.
- Vertices r_i and c_j are connected by edge $\leftrightarrow$ $a_{i,j} \neq 0$:

$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & * \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$

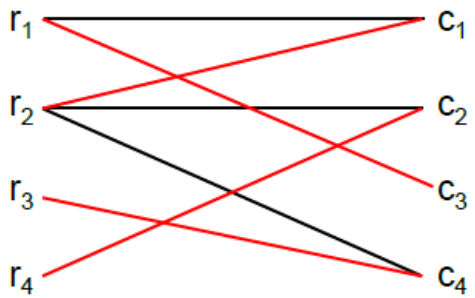


For each column we want to choose a large entry $\rightarrow$ diagonal.

Perfect Matching (cont.)

- Matching is subset of edges such that each row vertex is connected exactly to one column vertex and vice versa. Bijective mapping between V_r and V_c.

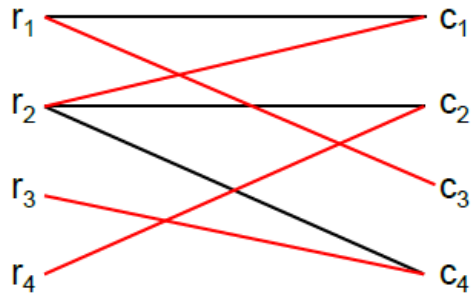
Matching in our example:



Perfect Matching (cont.)

- Matching is subset of edges such that each row vertex is connected exactly to one column vertex and vice versa. Bijective mapping between V_r and V_c.

Matching in our example:



- This matching induces row permutation to permute related entries $a_{1,3}$, $a_{2,1}$, $a_{3,4}$, and $a_{4,2}$ on the diagonal positions.

Permutation:
$1 \rightarrow 3, 2 \rightarrow 1, 3 \rightarrow 4, 4 \rightarrow 2$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

$$(1 \quad 3 \quad 4 \quad 2)$$

$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & * \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$

Perfect Matching (cont.)

Permutation: $1 \to 3, 2 \to 1, 3 \to 4, 4 \to 2$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix}$$

$$(1 \quad 3 \quad 4 \quad 2)$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_4 \\ x_1 \\ x_3 \end{pmatrix}$$

Perfect Matching (cont. 2)

- Add additional condition to matching problem: ”Find matching that maximizes a given function”
- Look for a subset of edges $M \leq E$:
  - where each vertex is incident to exactly one edge e in M and
  - where the matched edges maximize a weight function, e.g.

$$w(M) = \sum_{(i,j) \in M} c_{i,j} \qquad \text{with} \qquad c_{i,j} = \begin{cases} \log(|a_{i,j}|) & \text{for } a_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Perfect Matching (cont. 2)

- Add additional condition to matching problem: ”Find matching that maximizes a given function”
- Look for a subset of edges $M \leq E$:
  - where each vertex is incident to exactly one edge e in M and
  - where the matched edges maximize a weight function, e.g.

$$w(M) = \sum_{(i,j) \in M} c_{i,j} \qquad \text{with} \qquad c_{i,j} = \begin{cases} \log(|a_{i,j}|) & \text{for } a_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Solution of this perfect matching problem leads to permutation of A such that product of the new diagonal entries is maximized (therefore, all the diagonal entries should be large)
- Exact solution to costly, use heuristic approximate solutions!

Perfect Matching: Example

$$A = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & & \\ & & & & 1 \\ & & 1 & & \\ & 1 & & & \\ & & & 1 & \end{pmatrix} \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \\ 0 & 1 & -80 & 0 & 0 \\ -1 & 0 & 0 & -120 & 0 \\ 1 & -2 & 0 & 0 & 110 \end{pmatrix}$$

Perfect Matching: Example

$$A = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & & \\ & & & & 1 \\ & & 1 & & \\ & 1 & & & \\ & & & 1 & \end{pmatrix} \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 1 & -2 & 0 & 0 & 110 \\ -1 & 0 & 0 & -120 & 0 \\ 0 & 1 & -80 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 100 & 0 & -1 & 0 & 0 \\ 2 & 90 & -2 & 1 & 1 \\ 0 & 1 & -80 & 0 & 0 \\ -1 & 0 & 0 & -120 & 0 \\ 1 & -2 & 0 & 0 & 110 \end{pmatrix}$$

$$A = \begin{pmatrix} 100 & 90 & -100 & 80 & 70 \\ 1 & -2 & 0 & 0 & 10 \\ -1 & 0 & 0 & -20 & 0 \\ 0 & 1 & -8 & 0 & 0 \\ 2 & 9 & -2 & 1 & 1 \end{pmatrix} \rightarrow \; ?$$

Perfect Matching for Symmetric *A*

- Given: Symmetric matrix *A*
- *s*: small entry, *l*: large entry
- Find symmetric reordering that maximizes the diagonal.

$$A = \begin{pmatrix} s & \boxed{l} & & l & & \\ l & & & \boxed{l} & & s \\ & & & & \boxed{l} & \\ \boxed{l} & l & & & s & \\ & & \boxed{l} & s & & \\ & s & & & & \boxed{l} \end{pmatrix}$$

Perfect Matching for Symmetric A (cont.)

- For symmetric A row permutation would destroy symmetry! We need way to move large entries **near** the diagonal and permute rows and columns symmetrically!

Perfect Matching for Symmetric A (cont.)

- For symmetric A row permutation would destroy symmetry! We need way to move large entries **near** the diagonal and permute rows and columns symmetrically!
- Idea: Solve perfect matching unsymmetrically $\rightarrow$ gives permutation.
- Resulting permutation can be written as sequence of cyclic permutations, e.g., we can rewrite the permutation in the following way

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix} \Rightarrow (1 \quad 2 \quad 4)(3 \quad 5)(6)$$

Perfect Matching for Symmetric A (cont.)

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix} \Rightarrow (1 \quad 2 \quad 4)(3 \quad 5)(6)$$

Original row permutation gives

$$P := \begin{pmatrix} & 1 & & & & \\ & & & & 1 & \\ & & & 1 & & \\ 1 & & & & & \\ & & 1 & & & \\ & & & & & 1 \end{pmatrix}$$

Perfect Matching for Symmetric A (cont.)

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix} \Rightarrow (1 \quad 2 \quad 4)(3 \quad 5)(6)$$

Original row permutation gives

$$P := \begin{pmatrix} & 1 & & & & \\ & & & & 1 & \\ & & & 1 & & \\ 1 & & & & & \\ & & 1 & & & \\ & & & & & 1 \end{pmatrix}$$

Reordering of columns in view of cyclic representation gives

Reordering such that the indices inside a cycle are ordered sequentially: $3 \leftrightarrow 4$

$$P_c := \begin{pmatrix} & 1 & & & & \\ & & 1 & & & \\ 1 & & & & & \\ & & & & 1 & \\ & & & 1 & & \\ & & & & & 1 \end{pmatrix}$$

Perfect Matching for Symmetric A: Example

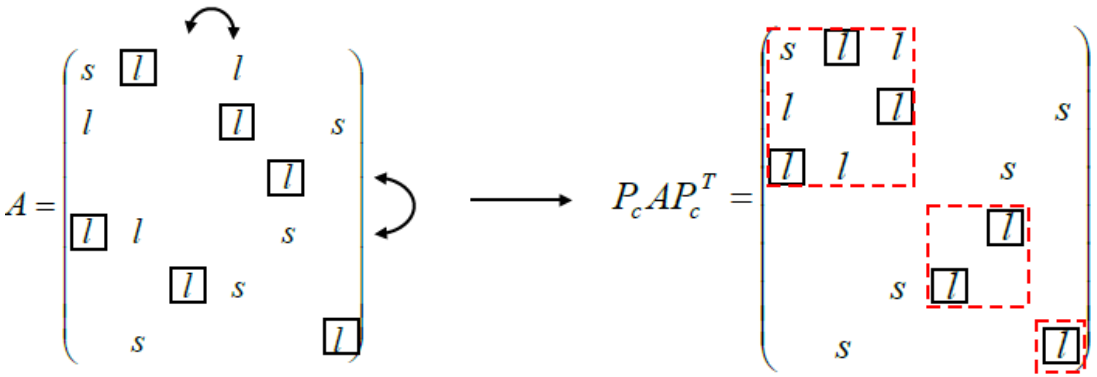
s: small entry, l: large entry, $\boxed{l}$: chosen entry

$$A = \begin{pmatrix} s & \boxed{l} & l & & & \\ l & & \boxed{l} & & s & \\ & & & \boxed{l} & & \\ \boxed{l} & l & & & s & \\ & & \boxed{l} & s & & \\ & s & & & & \boxed{l} \end{pmatrix}$$

- Nonsymmetric perfect matching $P = (1,2,4)(3,5)(6)$
- Apply symmetric permutation that groups together the blocks in P: switch $3 \leftrightarrow 4$.
- Change numbering in A accordingly.
- Then, the nonsymmetric perfect matching would be $P = (1,2,3)(4,5)(6)$
  $\rightarrow$, hence inside the blocks are the largest entries!

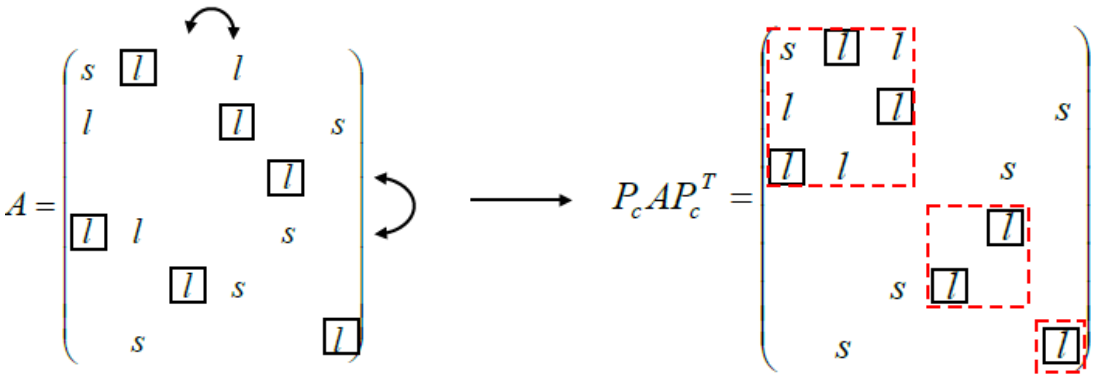
Perfect Matching for Symmetric A: Example (cont.)

P_c from P by changing the order of the columns: $P_c : 3 \leftrightarrow 4$

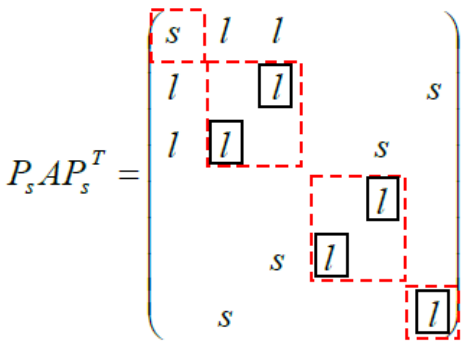


Perfect Matching for Symmetric A: Example (cont.)

P_c from P by changing the order of the columns: $P_c : 3 \leftrightarrow 4$



P_s from P_c by changing the interpretation of the blocks:



PARDISO: fastest parallel direct solver (uses perfect matching).

4.6. Gaussian Elimination for Sparse Matrices

4.6.1. Algebraic Pivoting in GE

- Numerical pivoting: for eliminating elements in column k choose large(st) entry in column/row/block k and permute this element on the diagonal position.
- Disadvantage: may lead to large fill in in the sparsity pattern of A.

4.7. Gaussian Elimination for Sparse Matrices

4.7.1. Algebraic Pivoting in GE

- Numerical pivoting: for eliminating elements in column k choose large(st) entry in column/row/block k and permute this element on the diagonal position.
- Disadvantage: may lead to large fill in in the sparsity pattern of A.
- Idea: Choose pivot element according to minimum fill in! Note that for well-conditioned $A = A^T > 0$ no numerical pivoting is necessary.
- Heuristic: Choose pivot element according to the degree in graph $\rightarrow$ minimum degree reordering

Special Case $A = A^T$

- For elimination in the kth column of A:
  - Define $r_m :=$ number of nonzero entries in row m
  - Choose pivot index i by $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column k

Special Case $A = A^T$

- For elimination in the kth column of A:
  - Define $r_m :=$ number of nonzero entries in row m
  - Choose pivot index i by $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column k

- r_m is #nonzeros in the mth row = #vertices directly connected with vertex m
  Hence, pivot vertex is vertex with minimum degree in $G(A_k)$

Special Case $A = A^T$

- For elimination in the kth column of A:
  - Define $r_m :=$ number of nonzero entries in row m
  - Choose pivot index i by $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column k

- r_m is #nonzeros in the mth row = #vertices directly connected with vertex m
  Hence, pivot vertex is vertex with minimum degree in $G(A_k)$

- Heuristics: few entries in mth row/column $\rightarrow$ few fill in because
  - only few elements to eliminate
  - the pivot row used in the elimination is very sparse

  $\rightarrow$ Multiple minimum degree reordering

Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.

Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.
  - Eliminating any will have no effect on the degree of the others, hence we can eliminate in Gaussian Elimination process in parallel (compare Multifrontal methods)
  - It reduces also the runtime because the outer loop is less than n.
  - It often leads to fewer nonzeros.

Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.
  - Eliminating any will have no effect on the degree of the others, hence we can eliminate in Gaussian Elimination process in parallel (compare Multifrontal methods)
  - It reduces also the runtime because the outer loop is less than n.
  - It often leads to fewer nonzeros.
- Approximative Minimum Degree Reordering (AMD) uses approximations of the degrees.

Generalization to Nonsymmetric Problems: Markowitz

- Define $r_m :=$ nnz in row m; $\quad c_p :=$ nnz in column p
- Choose pivot element with index pair (i, j) such that

$$(r_i - 1)(c_j - 1) = \min_{m,p}(r_m - 1)(c_p - 1)$$

Generalization to Nonsymmetric Problems: Markowitz

- Define $r_m :=$ nnz in row m; $\quad c_p :=$ nnz in column p
- Choose pivot element with index pair (i, j) such that

$$(r_i - 1)(c_j - 1) = \min_{m,p}(r_m - 1)(c_p - 1)$$

- Heuristics:
  - small c_j leads to few elimination steps
  - small r_i leads to sparse pivot row used in the elimination.
- Special case $r_i = 1$ or $c_j = 1$: no fill in.

Generalization to Nonsymmetric Problems: Markowitz

- Define $r_m :=$ nnz in row m; $\quad c_p :=$ nnz in column p
- Choose pivot element with index pair (i, j) such that

$$(r_i - 1)(c_j - 1) = \min_{m,p}(r_m - 1)(c_p - 1)$$

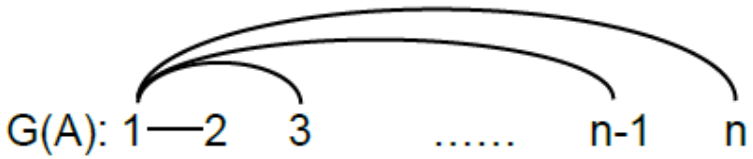
- Heuristics:
  - small c_j leads to few elimination steps
  - small r_i leads to sparse pivot row used in the elimination.
- Special case $r_i = 1$ or $c_j = 1$: no fill in.
- Include numerical pivoting by applying algebraic pivoting only on indices with absolute value that is not to small, e.g.,

$$|a_{i,j}| \geq 0.1 \cdot \max_{r,s} |a_{r,s}|$$

Comparison

Example:

$$A = \begin{pmatrix} * & * & * & \dots & * \\ * & * & & & \\ * & & * & & \\ \vdots & & & \ddots & \\ * & & & & * \end{pmatrix}$$

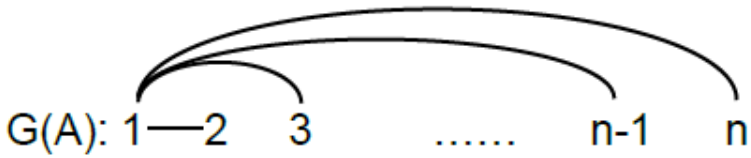


First elimination step leads to dense matrix!

$$\begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & * & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \dots & * \end{pmatrix}$$

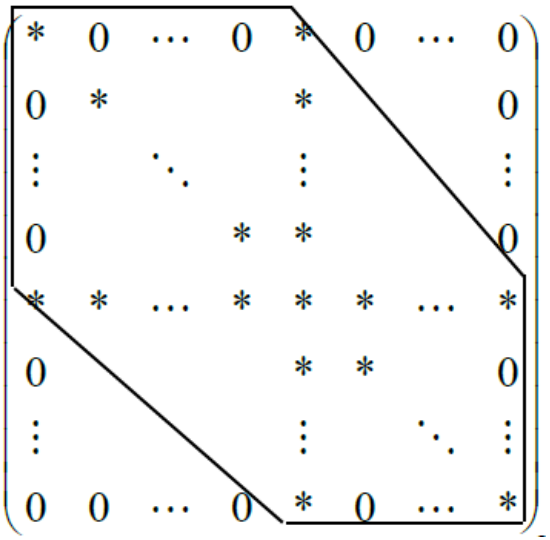
Comparison (cont.)

Cuthill McKee:



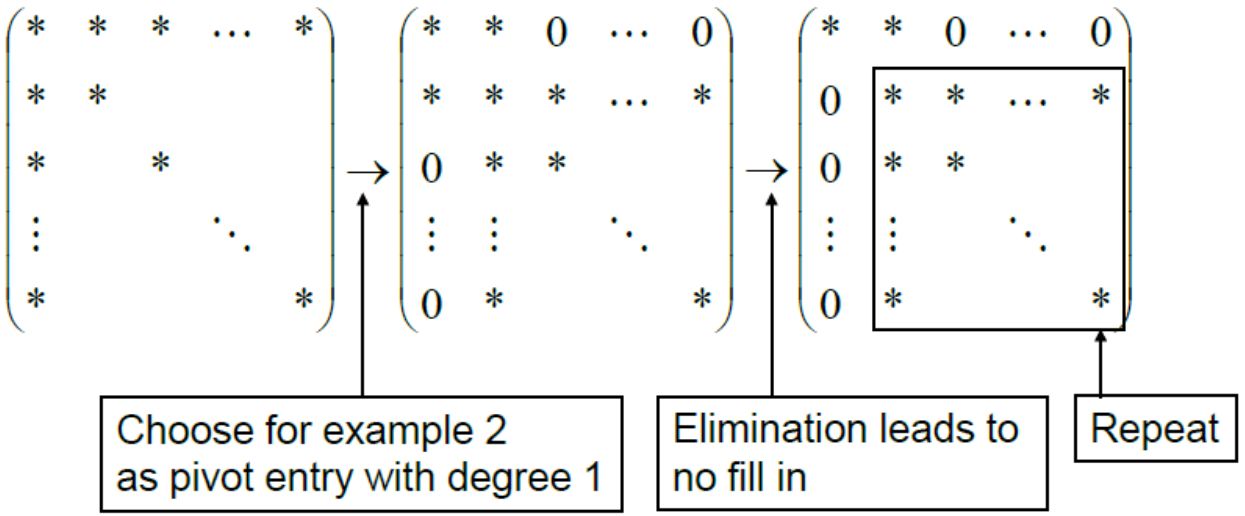
With starting vertex 1:
keeps numbers unchanged $\rightarrow$ no improvement

Optimal bandwidth is not satisfactory: bandwidht $\frac{n}{2}$



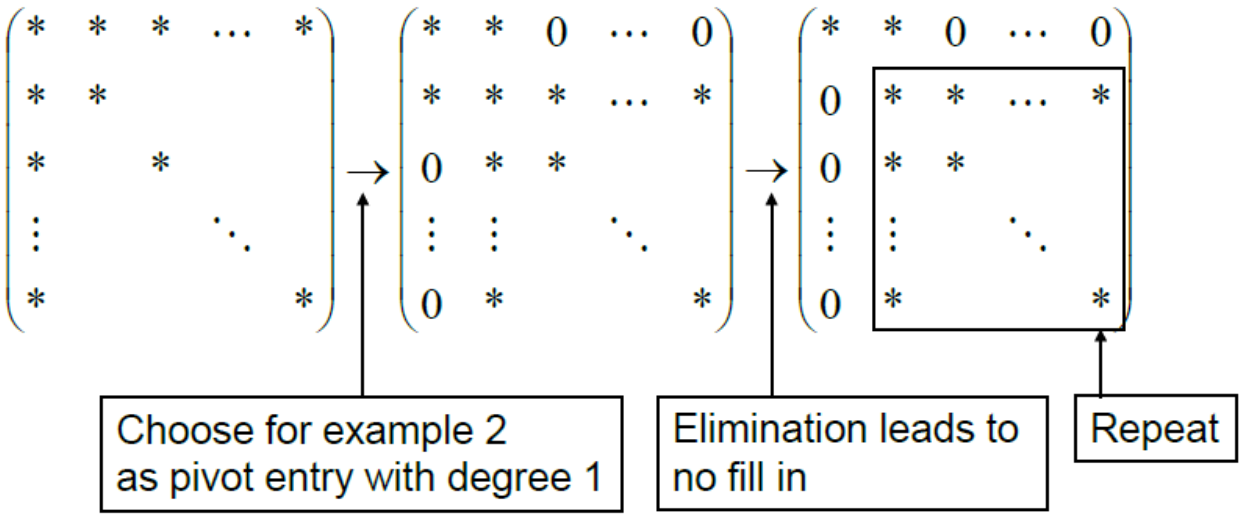
Comparison (cont. 2)

Minimum degree is efficient for this example (PAP^T):



Comparison (cont. 2)

Minimum degree is efficient for this example (PAP^T):

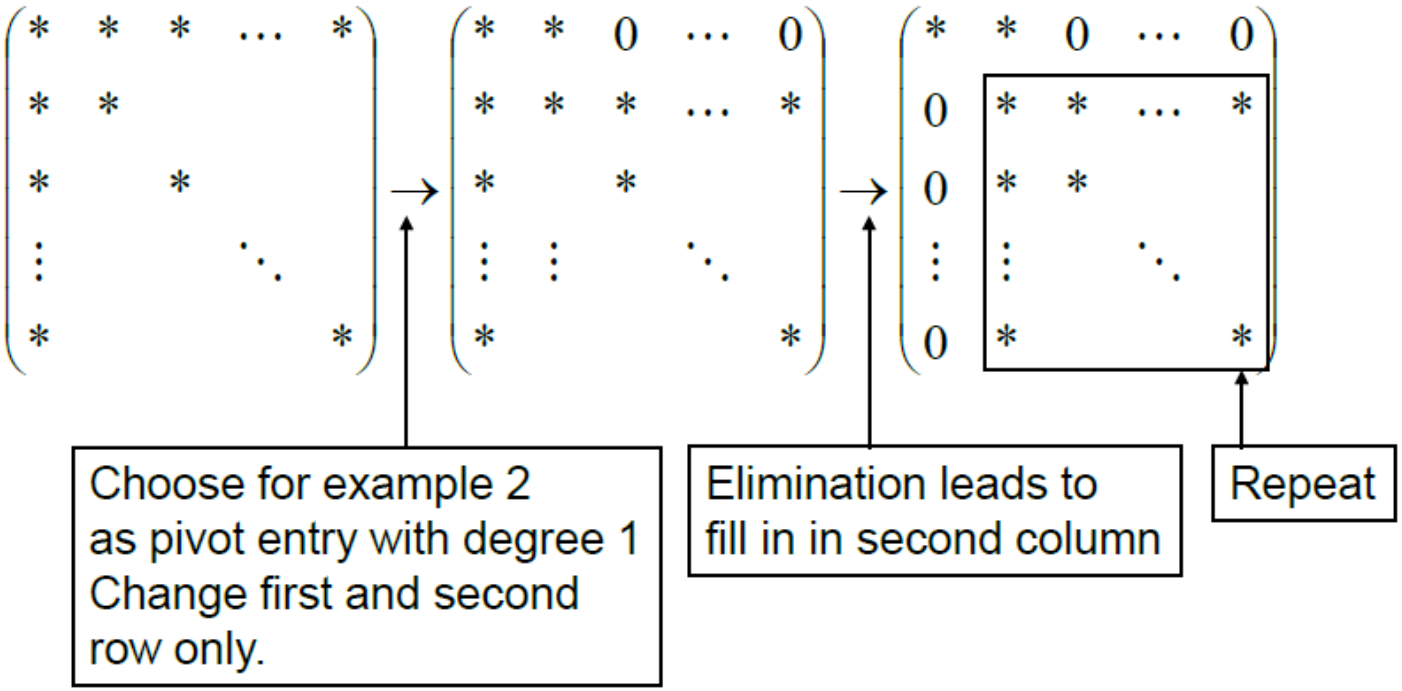


Optimal reordering in one step: $1 \leftrightarrow n$ Optimal costs: $\mathcal{O}(n)$.

$$\begin{pmatrix} * & 0 & 0 & \dots & * \\ 0 & * & & \dots & * \\ 0 & & * & & \vdots \\ \vdots & \vdots & \vdots & \ddots & * \\ * & * & \dots & * & * \end{pmatrix}$$

Example: Nonsymmetric Permutation

Costs: $\mathcal{O}(n^2)$



Test: MATLAB (symamd, colamd, amd, colperm, symrcm)

```
load('west0479.mat');a=west0479; s=a'*a; p=symamd(s); spy(s(p,p));
```

See matrix `'west0479.mat'` on Matrix Market:

http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/chemwest/west0479.html

4.7.2. Gaussian Elimination in Graph

- Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs (in terms of dense subgraphs (matrices) = clique)

4.7.3. Gaussian Elimination in Graph

- Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs (in terms of dense subgraphs (matrices) = clique)
- Modification of GE:
  1. Apply algebraic prestep for GE, determining the graphs related to the elimination matrices A_k in GE.
  2. Based on these graphs we can decide
     - whether GE will lead to nearly dense matrices (do not use GE in this case!)
     - what additional entries will appear during GE (prepare the storage)

4.7.4. Gaussian Elimination in Graph

- Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs (in terms of dense subgraphs (matrices) = clique)
- Modification of GE:
  1. Apply algebraic prestep for GE, determining the graphs related to the elimination matrices A_k in GE.
  2. Based on these graphs we can decide
     - whether GE will lead to nearly dense matrices (do not use GE in this case!)
     - what additional entries will appear during GE (prepare the storage)
- Algebraic prestep is cheap, can be implemented using cliques.