Parallel Numerics, WT 2012/2013

4 Sparse Matrices

Contents

4.1. General Properties of Sparse Matrices

- Full $n \times n$-matrix: storage $\mathcal{O}(n^2)$, solution $\mathcal{O}(n^3) \rightarrow$ too costly for most applications, esp. for fine discretization (large n)
- Formulate given problem in clever way that leads to a linear system that is sparse: $\mathcal{O}(n)$, solution $\mathcal{O}(n)$?
  - (that is structured: storage $\mathcal{O}(n)$, solution $O(n\log(n))$, e.g., FFT)
  - (that is dense, but reduced from, e.g., 3D to 2D)
  - (based on sparse grids)
  - (based on tensor approximations)
- Examples:
  - tridiagonal matrix
  - banded matrix
  - block band matrix

Sparse Matrix Example

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

Additionally we need to store:

- the size of the matrix $n = 5$
- the number of nonzero entries $\text{nnz} = 12$

4.1.1. Storage in Coordinate Form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

To store:

- n
- nnz
- $2 \cdot$ nnz integer for row and column indices in JR and JC
- nnz float in AA

No sorting included. Redundant information.

Storage in Coordinate Form (cont.)

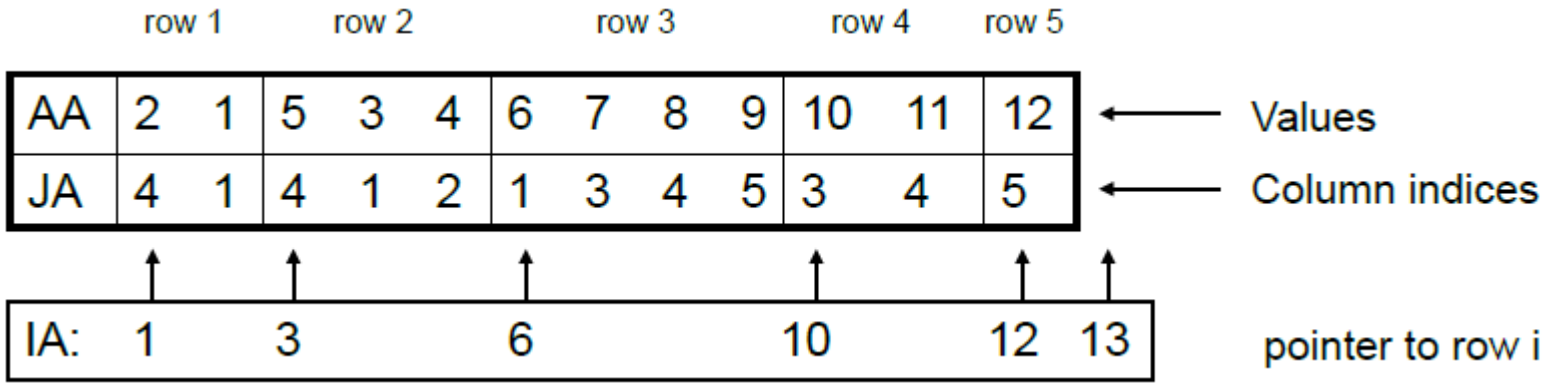
values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

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

$c = 0;$
`for` $j = 1 \ : \ \text{nnz}(A)$
$\quad c_{JR(j)} = c_{JR(j)} + \underbrace{AA(j)}_{A_{JR(j),JC(j)}} * b_{JC(j)};$
`end`

- Disadvantage: Indirect addressing (indexing) in vector c and $b \rightarrow$ jumps in memory
- Advantage: No difference between columns and rows (A and A^T), simple.

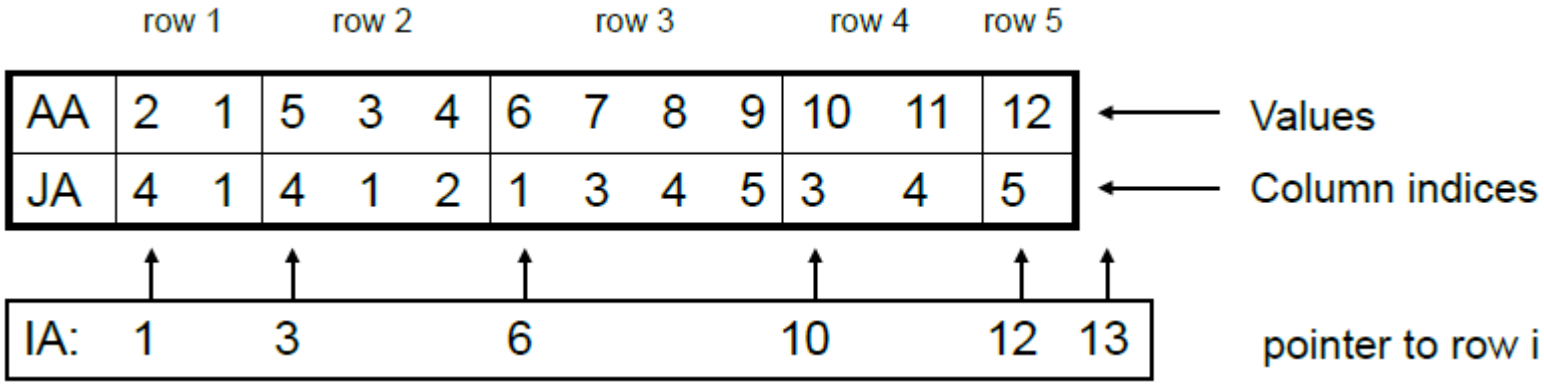
4.1.2. Compressed Sparse Row Format: CSR



Storage:

- n and nnz
- $n + \text{nnz} + 1$ integer
- nnz float

Compressed Sparse Row: CSR (cont.)

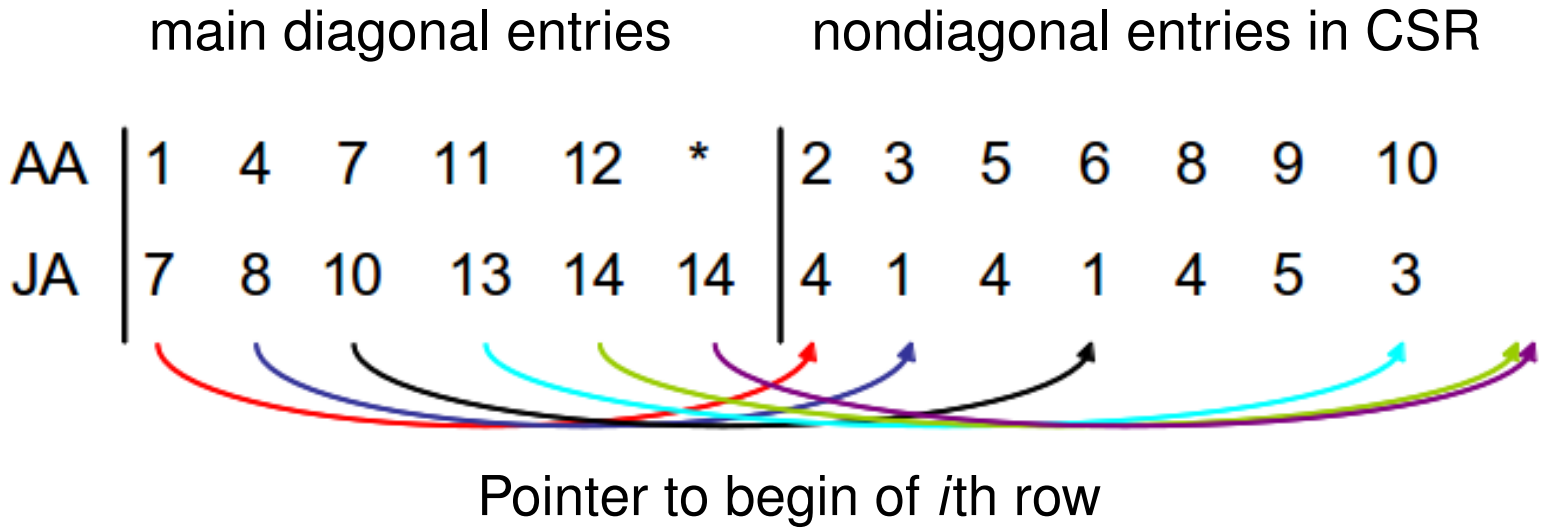


Pseudocode for computing $c = A \cdot b$:

```
c = 0;
for i = 1 : n
   for j = IA(i) : IA(i+1) - 1
      c_i = c_i + AA(j) * b_JA(j);
   end
end
```

- Indirect addressing only in b.
- Columnwise $\rightarrow$ compressed sparse column format.

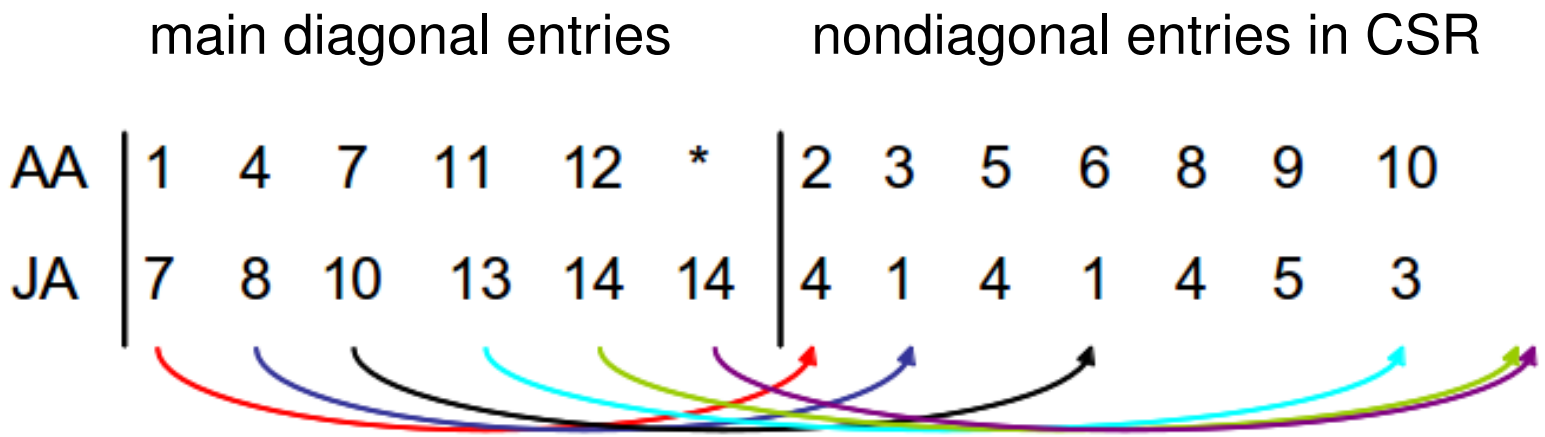
4.1.3. CSR with Extracted Main Diagonal



Storage:

- n and nnz
- nnz $+1$ integer
- nnz $+1$ float

CSR with Extracted Main Diagonal (cont.)



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

```
c = 0;
for i = 1 : n
   c_i = AA_i * b_i;
   for j = JA(i) : JA(i+1) - 1
      c_i = c_i + AA_j * b_JA(j);
   end
end
```

4.1.4. Diagonalwise Storage

$$\begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix}$$

New matrix A!
Different matrix to slides before!

Diagonal numbers: $-1 \;\; 0 \;\; 2$

Values in:

$$\text{DIAG} = \begin{pmatrix} * & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & * \\ 11 & 12 & * \end{pmatrix}, \quad \text{IOFF} = (-1 \;\; 0 \;\; 2)$$

Storage: n, $nd :=$ #diagonals, nd integers for IOFF and $n \cdot nd$ float

4.1.5. Rectangular Storage Scheme by Pressing from the Right

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

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).

4.1.6. Jagged Diagonal Form

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

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

Length 3

Length 2

- 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

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
Coordiante	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

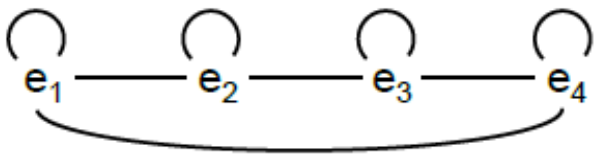
4.2. Sparse Matrices and Graphs

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

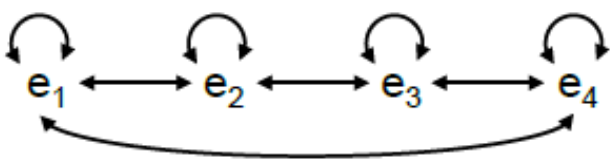
$A = A^T > 0$

$n \times n$-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.

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

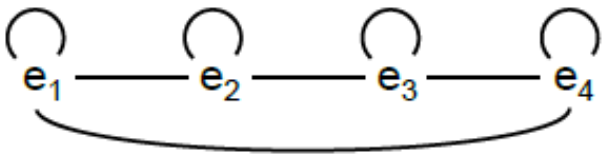
- Symmetric permutations of A in the form PAP^T change the ordering of the rows and columns of A simultaneously.
- Therefore, the graph of PAP^T can be obtained by the graph of A by renumbering the vertices.

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

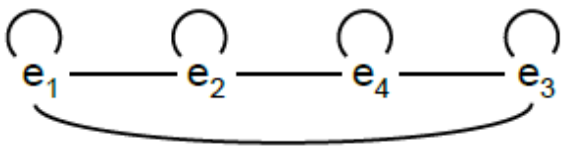
Symmetric permutation PAP^T with graph $G(PAP^T)$.

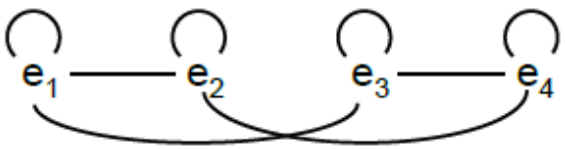
Example: P permutation that changes $3 \leftrightarrow 4$:

$G(A)$



$G(PAP^T)$

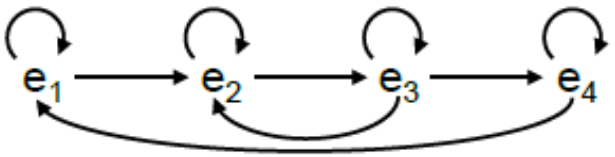




4.2.2. Matrix A nonsymmetric, $G(A)$ directed

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

G(A):



How can we characterize “good“ sparsity patterns?

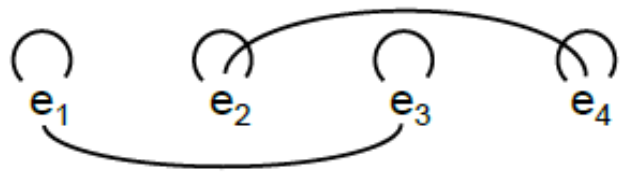
”good”: Gaussian elimination can be reduced to smaller subproblems or produces no (or small) fill-in.

Block Diagonal Pattern

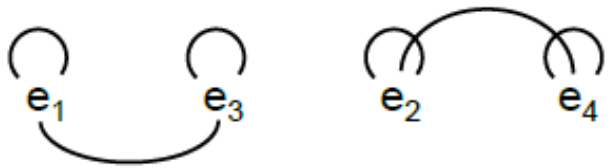
New matrix A!

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

G(A):



$2 \leftrightarrow 3$:



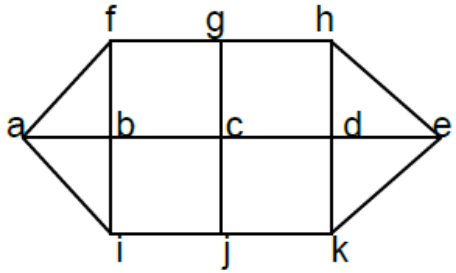
$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

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

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

4.3. Reordering

Consider sparse matrix A with graph G(A):

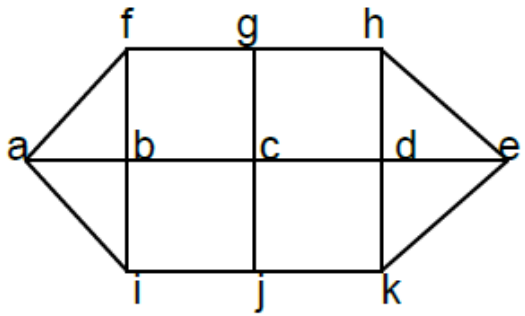


Consider following matrix (in tabular form) with bandwidth 9:

	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

4.3.1. 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
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!

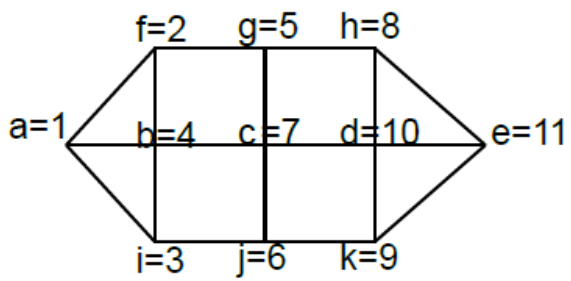
$S_1 := \{a\}$

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

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

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

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



Cuthill McKee (cont. 2)

- Resulting matrix with small bandwidth 4:

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

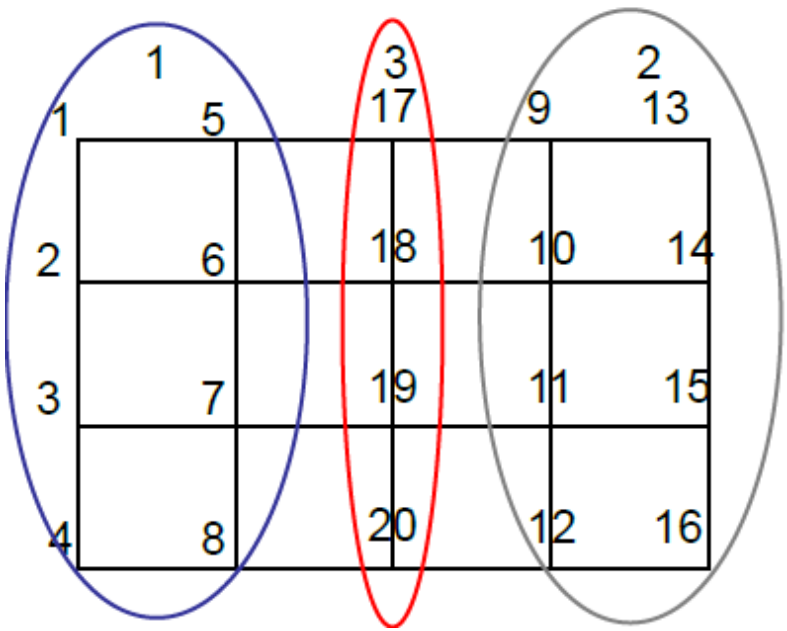
- Often „Reverse Cuthill Mckee“ gives slightly better results: reversing the ordering of Cuthill McKee: $11 \leftrightarrow 1$, $10 \leftrightarrow 2, \ldots$ Choose start as extreme vertex (diameter)!
- This method is far from being optimal and strongly heuristical.

4.3.2. Dissection Reordering

Consider matrix A with graph G(A):

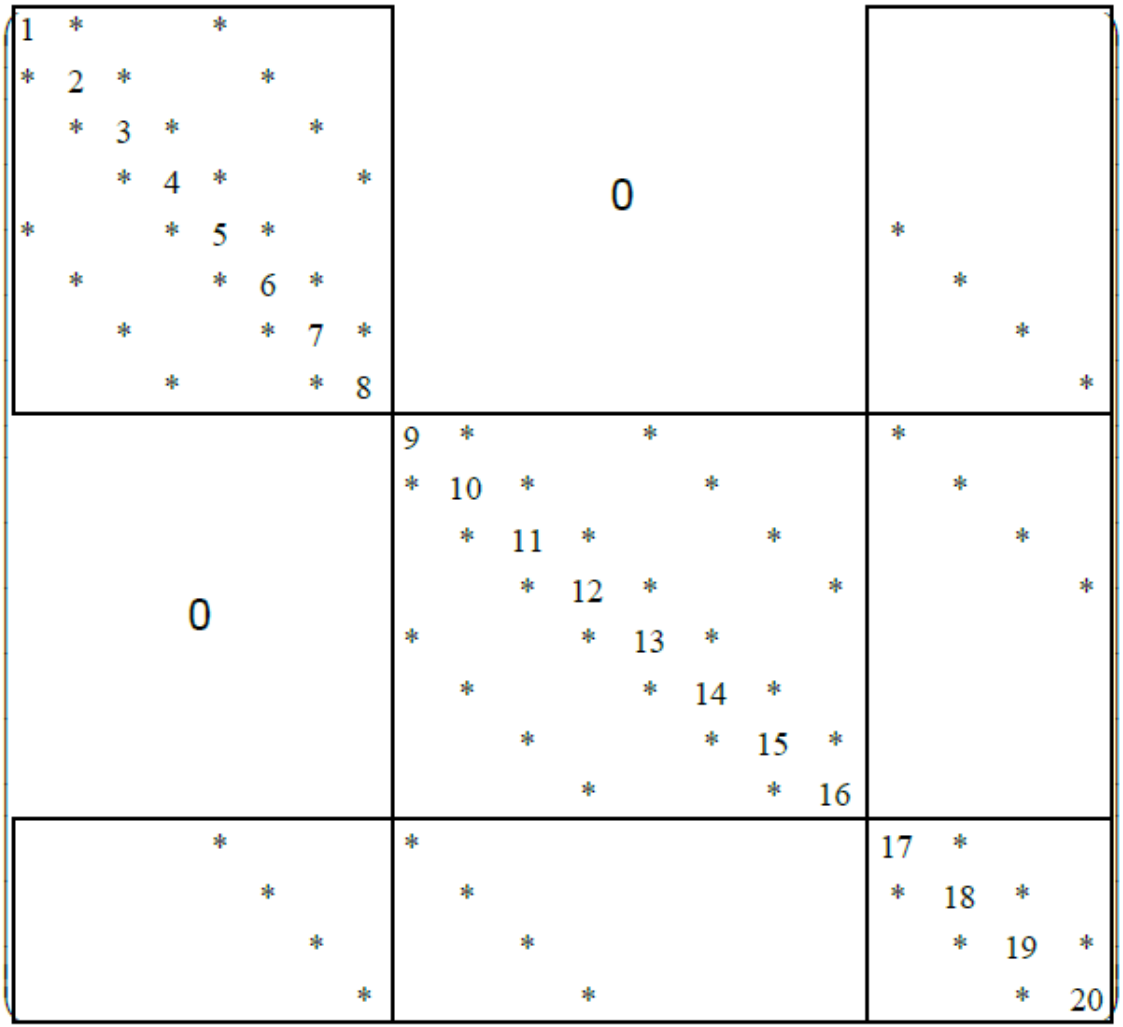
Numbering of unknows in two groups separated by third group.

group:



Dissection Reordering (cont.)

This numbering leads to sparsity pattern:



Dissection Reordering (cont. 2)

- Leads automatically to dissection form:

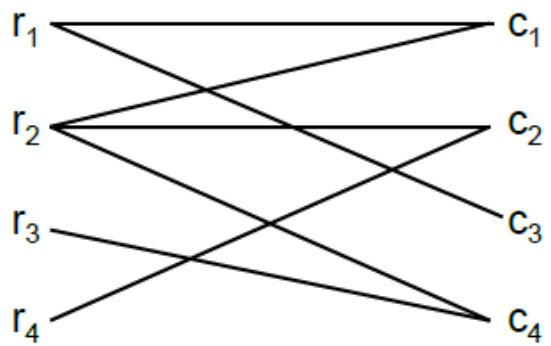
$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

- 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!

4.3.3. 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 $\leftrightarrow$ $a_{i,j} \neq 0$:

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

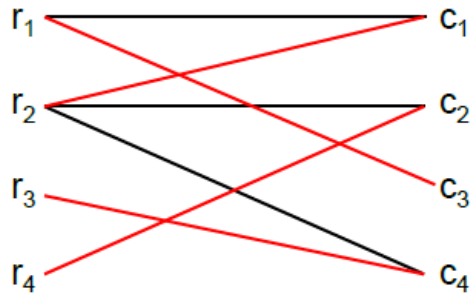


For each column we want to choose a large entry $\rightarrow$ diagonal.

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:



- 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. 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

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

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

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

Perfect Matching for Symmetric A

- 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

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

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

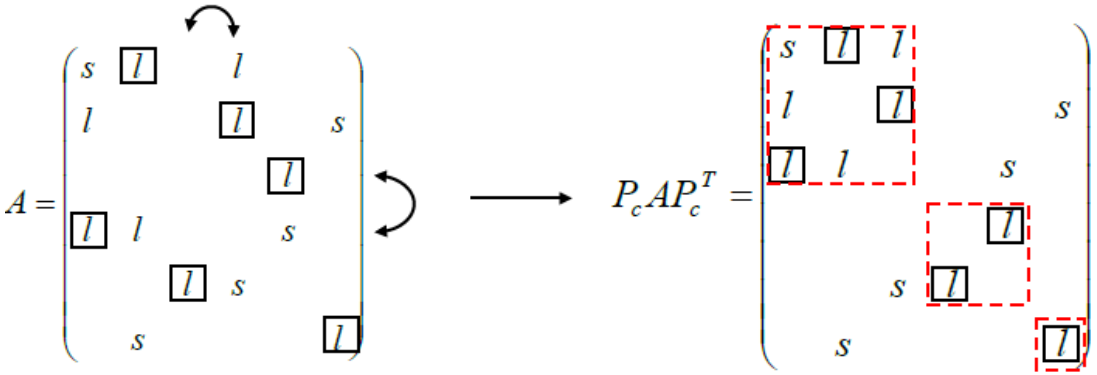
s: small entry, l: large entry, $\boxed{l}$: chosen entry

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

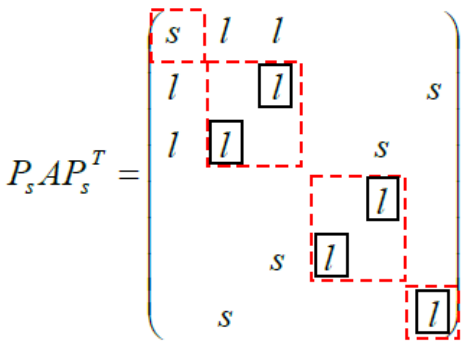
- Nonsymmetric perfect matching $P = (1,2,4)(3,5)(6)$
- To allow symmetric permutation: switch $3 \leftrightarrow 4$.
- Change numbering in A accordingly.
- Then, the nonsymmetric perfect matching is $P = (1,2,3)(4,5)(6)$
  $\rightarrow$ block structure maintained under symmetric application of P!

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).

4.4. Gaussian Elimination for Sparse Matrices

4.4.1. 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

- 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

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

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

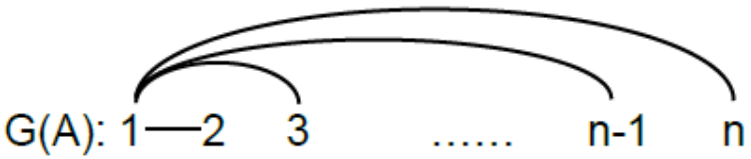
- Heuristics:
  - small c_j leads to few elimination steps
  - small r_i leads to sparse pivot row used in the elimination.
- Special case $r_i = 1$ or $c_j = 1$: no fill in.
- Include numerical pivoting by applying algebraic pivoting only on indices with absolute value that is not to small, e.g.,

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

Comparison

Example:

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

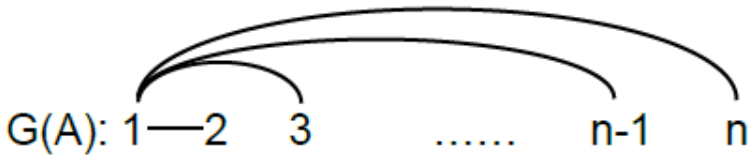


First elimination step leads to dense matrix!

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

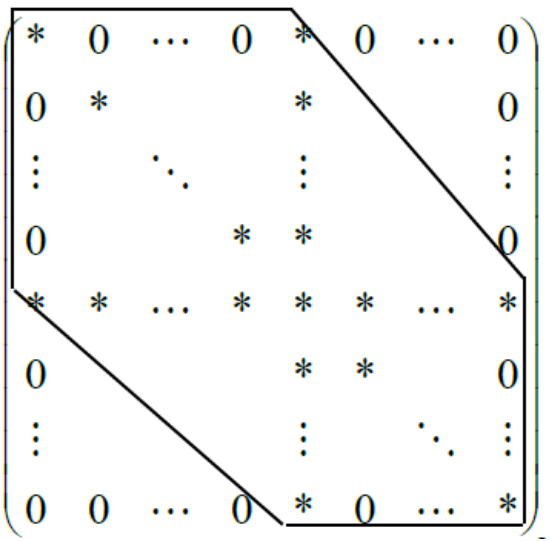
Comparison (cont.)

Cuthill McKee:



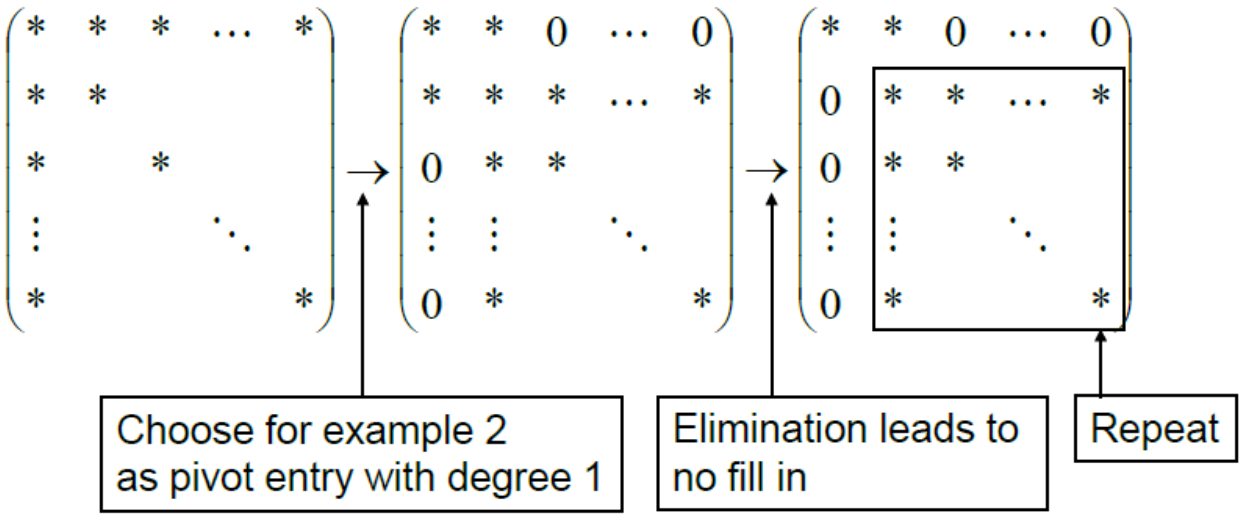
With starting vertex 1:
keeps numbers unchanged $\rightarrow$ no improvement

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



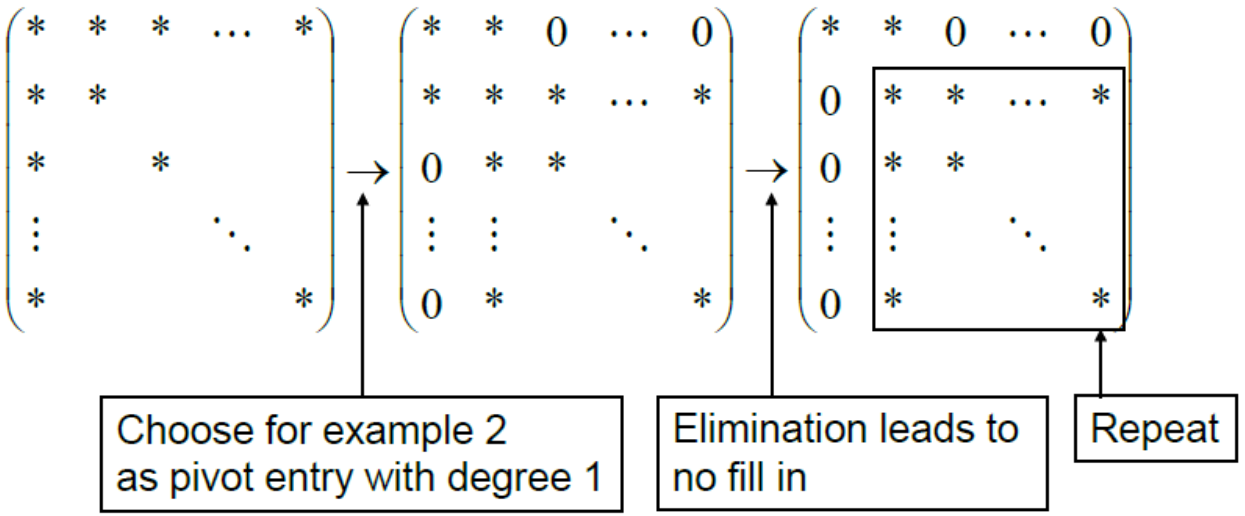
Comparison (cont. 2)

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



Comparison (cont. 2)

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

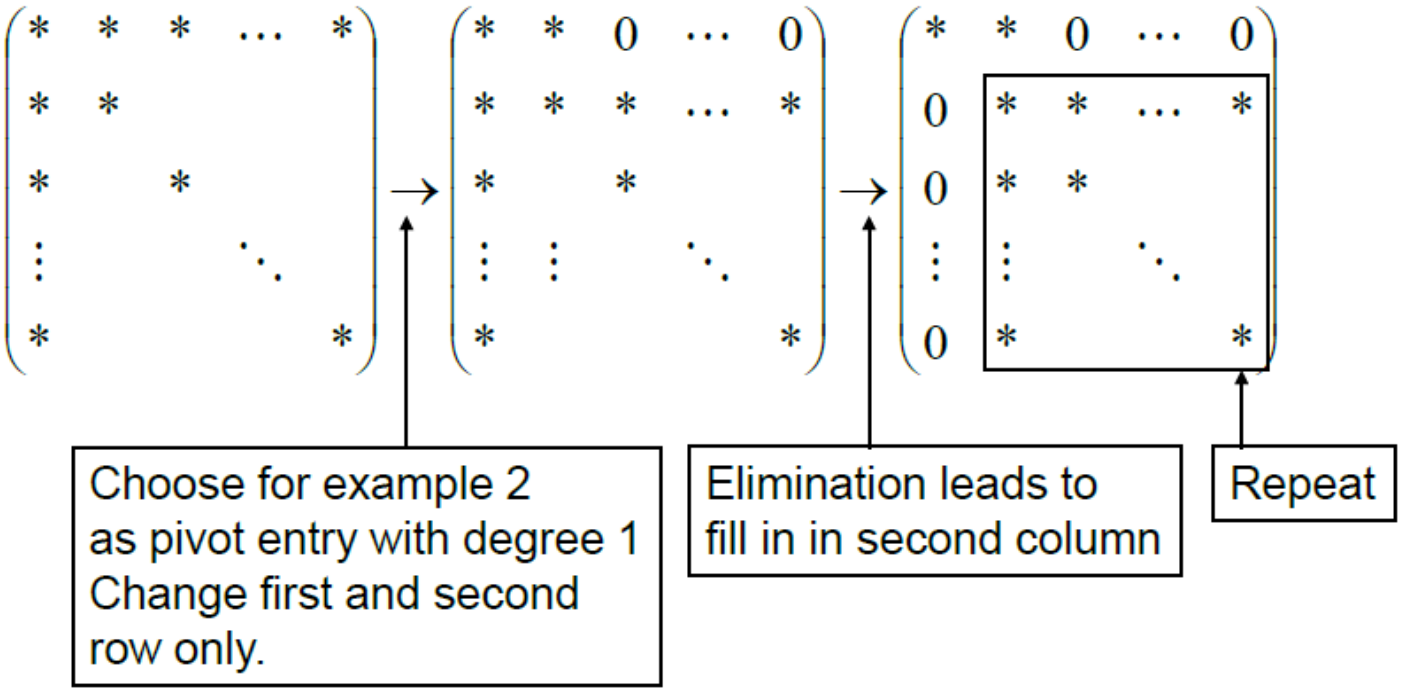


Optimal reordering in one step: $1 \leftrightarrow n$ Optimal costs: $\mathcal{O}(n)$.

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

Example: Nonsymmetric Permutation

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



Test: MATLAB (symmmd, colamd, colmmd, symrcm,syamd)

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

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

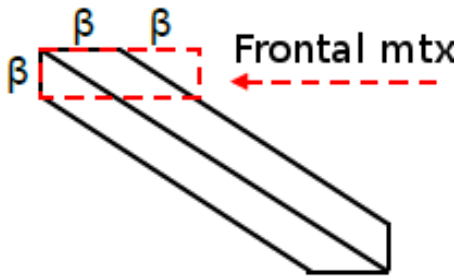
http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/chemwest/west0479.html

4.4.2. 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)
- 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.

4.4.3. A Parallel Direct Solver

Frontal methods for band matrices (from PDE)



- Frontal dense matrix of size $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.
- Then move frontal matrix one entry down-right and do next elimination.
- Repeat until done.
- No parallelism until now.

Multifrontal in Parallel

To make efficient use of parallelism search for “independent“ elimination steps!

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

$A_{1,1}$ as first pivot element is related to a first frontal matrix that contains all information to eliminate the first column:

Dense submatrix for $k = 1$:

$$\begin{array}{ccc} A_{1,1} & A_{1,3} & A_{1,4} \\ A_{3,1} & \frac{A_{3,1} \cdot A_{1,3}}{A_{1,1}} & \frac{A_{3,1} \cdot A_{1,4}}{A_{1,1}} \\ A_{4,1} & \frac{A_{4,1} \cdot A_{1,3}}{A_{1,1}} & \frac{A_{4,1} \cdot A_{1,4}}{A_{1,1}} \end{array}$$

Multifrontal in Parallel (cont.)

Because $A_{1,2} = A_{2,1} = 0$ we can in parallel consider already the frontal matrix related to $k = 2$, the second step:

$$\begin{array}{ccc} A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,2} & \frac{A_{3,2} \cdot A_{2,3}}{A_{2,2}} & \frac{A_{3,2} \cdot A_{2,4}}{A_{2,2}} \\ A_{4,2} & \frac{A_{4,2} \cdot A_{2,3}}{A_{2,2}} & \frac{A_{4,2} \cdot A_{2,4}}{A_{2,2}} \end{array}$$

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

Multifrontal in Parallel (cont. 2)

The computations for $k = 1$ and $k = 2$ are independent and can be done in parallel (in the 2×2 block):

$k = 1$:

$$A_{i,j} \to A_{i,j} - \frac{A_{i,1} \cdot A_{1,j}}{A_{1,1}}$$

$k = 2$:

$$A_{i,j} \to A_{i,j} - \frac{A_{i,2} \cdot A_{2,j}}{A_{2,2}}$$

Number of frontal matrices that can be used in parallel depends on the sparsity pattern.