

## 4.3.7. Jagged Diagonal Form

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

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

- 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;
    ci = ci + DJ(k) * b(JDIAG(k));
  end
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;
    ci = ci + DJ(k) * b(JDIAG(k));
  end
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

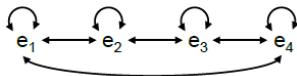
### 4.4.1. Graph $G(A)$ for symmetric positive definite (SPD)

$$A = A^T > 0$$

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

$$A = \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix} \rightarrow G(A) : \begin{array}{cccc} \textcircled{e_1} & \textcircled{e_2} & \textcircled{e_3} & \textcircled{e_4} \\ | & | & | & | \\ \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & & & \text{---} \end{array}$$

$G(A)$  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.

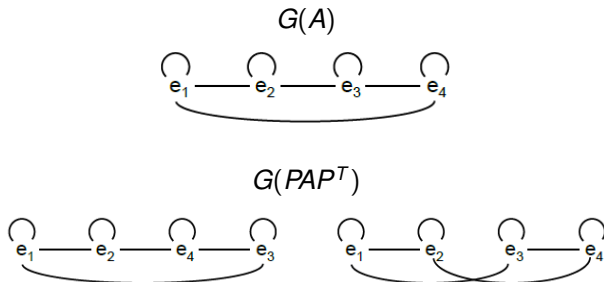




## Matrix $A$ with graph $G(A)$ (cont.)

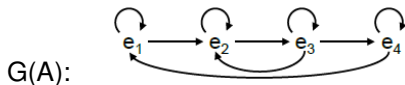
Symmetric permutation  $PAP^T$  with graph  $G(PAP^T)$ .

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



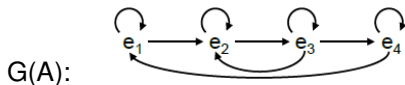
## 4.4.2. Matrix $A$ nonsymmetric, $G(A)$ directed

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



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

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



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  $A!$

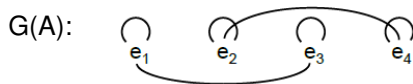
$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$



# Block Diagonal Pattern

New matrix  $A!$

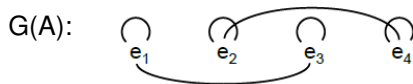
$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow \mathcal{A}(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$



# Block Diagonal Pattern

New matrix  $A!$

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



$2 \leftrightarrow 3$ :

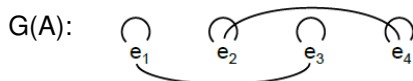
$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$



# Block Diagonal Pattern

New matrix  $A!$

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



$2 \leftrightarrow 3$ :

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

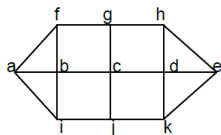
By this permutation,  $A$  can be transformed into block diagonal form  $\rightarrow$  easy to solve, fully parallel!

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



## 4.5. Reordering

Consider sparse matrix  $A$  with graph  $G(A)$ :



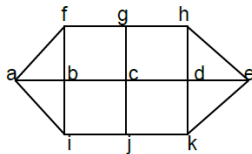
Consider following matrix (in tabular form) with bandwidth 9:

	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

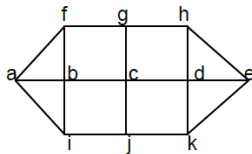
Graph  $G(A)$ :



Optimal ordering that leads to small bandwidth?

## 4.5.2. Cuthill McKee

Graph  $G(A)$ :



Optimal ordering that leads to small bandwidth?

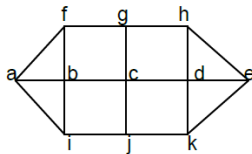
Level sets, starting with:

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



### 4.5.3. Cuthill McKee

Graph  $G(A)$ :



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

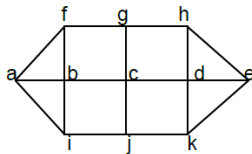
$$S_1 := \{a\}$$

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



## 4.5.4. Cuthill McKee

Graph  $G(A)$ :



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

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

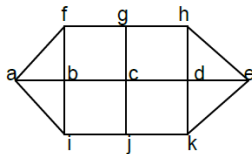
$$S_2 := \{f, b, i\}, \text{ all vertices directly connected with } S_1$$

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



## 4.5.5. Cuthill McKee

Graph  $G(A)$ :



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

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

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

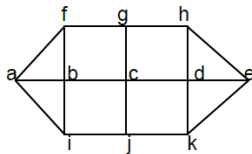
$$S_3 := \{g, c, j\}, \text{ all vertices directly connected with } S_2$$

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



## 4.5.6. Cuthill McKee

Graph  $G(A)$ :



Optimal ordering that leads to small bandwidth?

Level sets, starting with:

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

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

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

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

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



# 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!

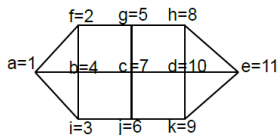
$$S_1 := \{a\}$$

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

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

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

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





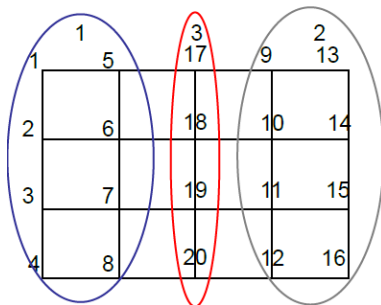


## 4.5.7. Dissection Reordering

Consider matrix  $A$  with graph  $G(A)$ :

Numbering of unknowns in two groups separated by third group.

group:





## 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 complement in the Chapter Domain Decomposition methods.



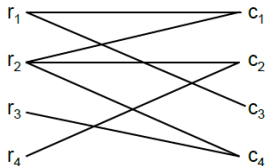
## 4.5.8. Perfect Matching Reordering

- 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  $V_r = \{r_1, \dots, r_n\}$  and  $V_c = \{c_1, \dots, c_n\}$ .
- Vertices  $r_i$  and  $c_j$  are connected by edge  $\leftrightarrow a_{i,j} \neq 0$ :

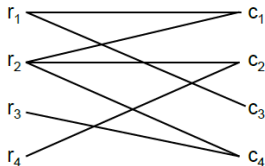
$$A = \begin{pmatrix} * & 0 & * & 0 \\ * & * & 0 & 0 \\ 0 & 0 & 0 & * \\ 0 & * & 0 & 0 \end{pmatrix}$$



## 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  $V_r = \{r_1, \dots, r_n\}$  and  $V_c = \{c_1, \dots, c_n\}$ .
- Vertices  $r_i$  and  $c_j$  are connected by edge  $\leftrightarrow a_{i,j} \neq 0$ :

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

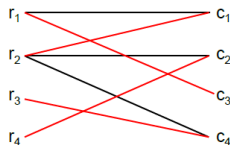


For each column we want to choose a large entry  $\rightarrow$  diagonal.

## 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  $V_c$ .

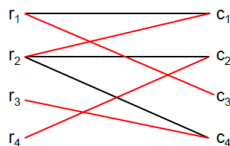
Matching in our example:



## 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:

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

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

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



## Perfect Matching (cont.)

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

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

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

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



## 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.

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





## 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.

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

- 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 $A$ (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

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



## Perfect Matching for Symmetric $A$ (cont.)

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

Original row permutation gives

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



## Perfect Matching for Symmetric $A$ (cont.)

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

Original row permutation gives

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

Reordering of columns in view of cyclic representation gives

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

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











## 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  
→ minimum degree reordering



## Special Case $A = A^T$

- For elimination in the  $k$ th 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  $k$ th 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  $m$ th 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  $k$ th 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  $m$ th row = #vertices directly connected with vertex  $m$   
Hence, pivot vertex is vertex with minimum degree in  $G(A_k)$
- Heuristics: few entries in  $m$ th 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  $r_m := \text{nnz}$  in row  $m$ ;  $c_p := \text{nnz}$  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  $r_m := \text{nnz}$  in row  $m$ ;  $c_p := \text{nnz}$  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  $r_m := \text{nnz}$  in row  $m$ ;  $c_p := \text{nnz}$  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 too small, e.g.,

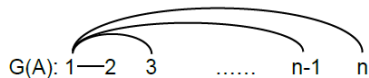
$$|a_{i,j}| \geq 0.1 \cdot \max_{r,s} |a_{r,s}|$$



# Comparison

Example:

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



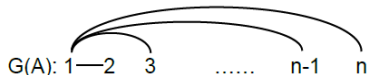
First elimination step leads to dense matrix!

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



# Comparison (cont.)

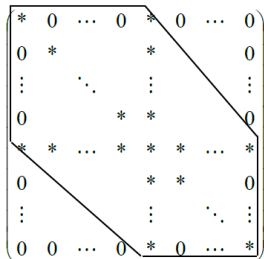
Cuthill McKee:



With starting vertex 1:

keeps numbers unchanged  $\rightarrow$  no improvement

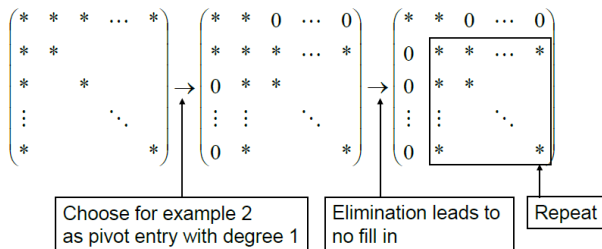
Optimal bandwidth is not satisfactory: bandwidth  $\frac{n}{2}$





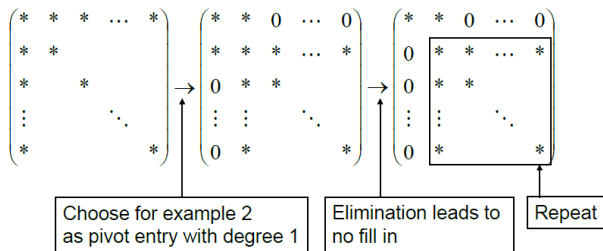
## Comparison (cont. 2)

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



## Comparison (cont. 2)

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



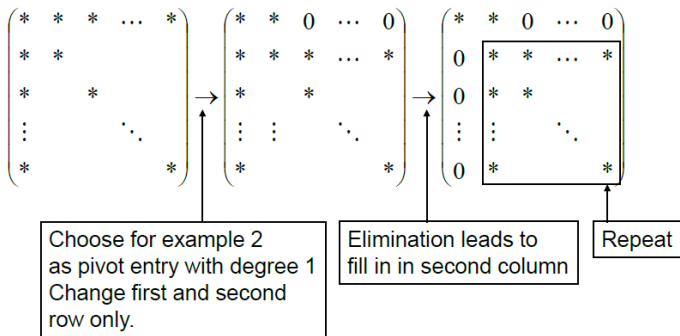
Optimal reordering in one step:  $1 \leftrightarrow n$       Optimal costs:  $\mathcal{O}(n)$ .

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



## 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)



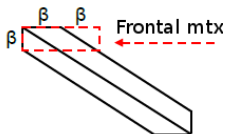
## 4.7.4. Gaussian Elimination in Graph

- 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.



## 4.7.5. A Parallel Direct Solver

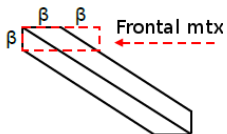
Frontal methods for band matrices (from PDE)



- Frontal dense matrix of size  $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.

## 4.7.6. A Parallel Direct Solver

Frontal methods for band matrices (from PDE)

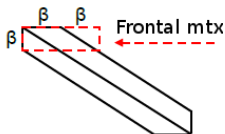


- Frontal dense matrix of size  $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.
- Then move frontal matrix one entry down-right and do next elimination.
- Repeat until done.



## 4.7.7. 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:



## 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  $2 \times 2$  block):

$k = 1$ :

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

$k = 2$ :

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

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

