UNIVERSITY OF MINNESOTA TWIN CITIES

A tutorial on:
Iterative methods for Sparse Matrix Problems

Yousef Saad

University of Minnesota
Computer Science and Engineering

CRM Montreal - April 30, 2008

Outline

Part 1

- Sparse matrices and sparsity
- Basic iterative techniques
- Projection methods
- Krylov subspace methods

Part 2

- Preconditioned iterations
- Preconditioning techniques

Part 3

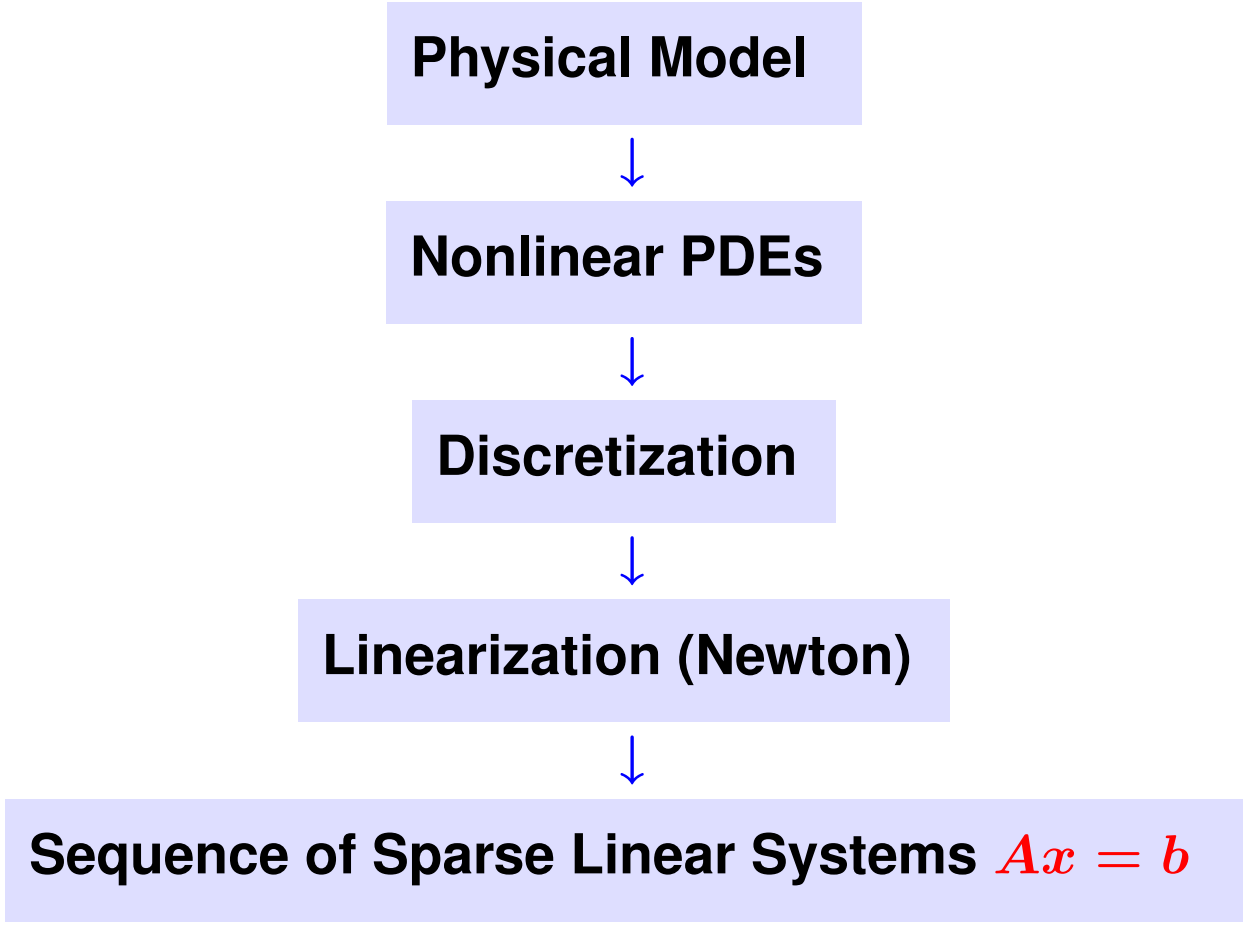
- Parallel implementations
- Multigrid methods

Part 4

- Eigenvalue problems
- Applications

INTRODUCTION TO SPARSE MATRICES

Typical Problem:



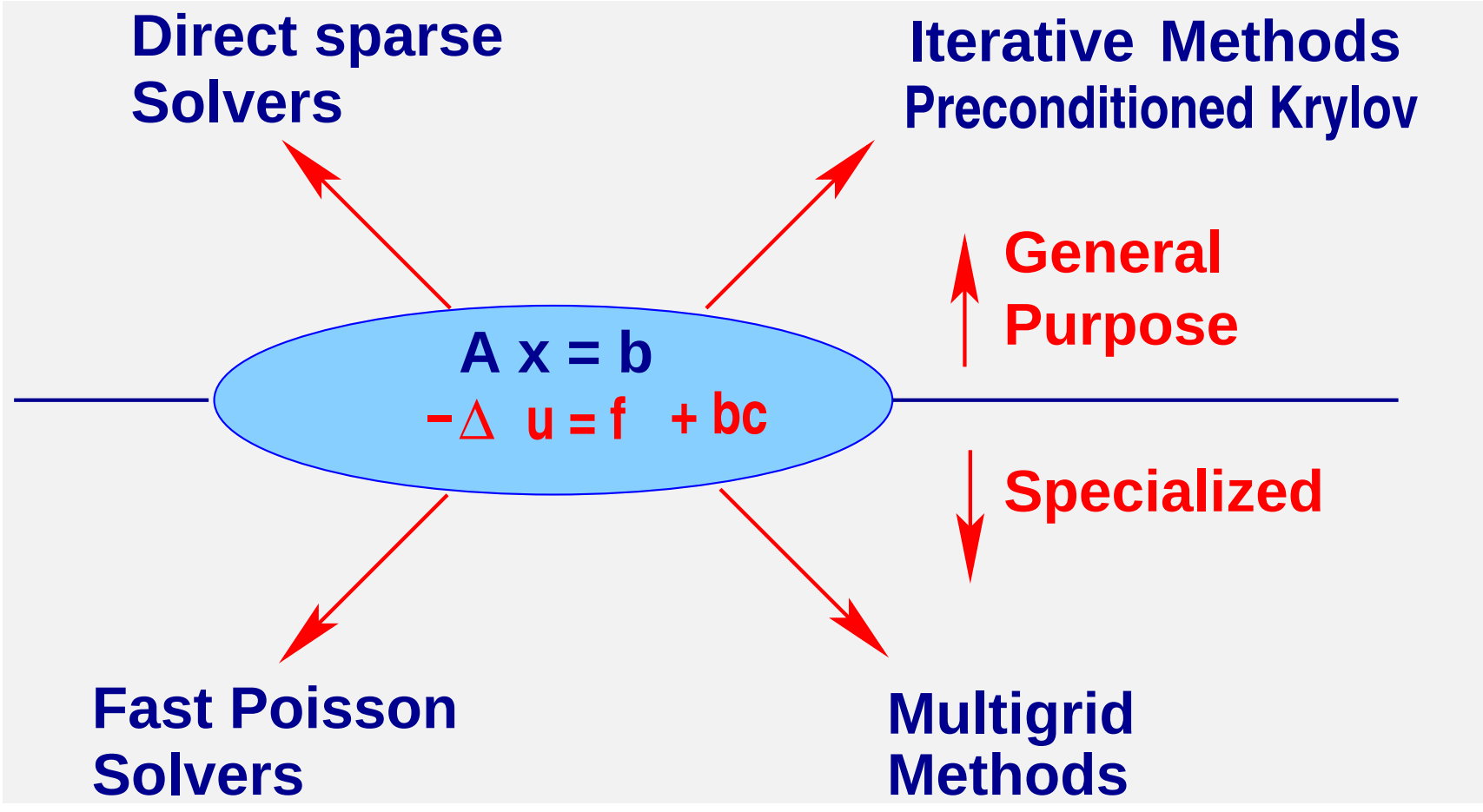
Introduction: Linear System Solvers

- **Problem considered: Linear systems**

$$Ax = b$$

- **Can view the problem from somewhat different angles:**

  - **Discretized problem coming from a PDE**
  - **An algebraic system of equations [ignore origin]**
  - **Sometimes a system of equations where A is not explicitly available**


Direct sparse
Solvers
Iterative Methods
Preconditioned Krylov
General
Purpose
A x = b
−Δ u = f + bc
Specialized
Fast Poisson
Solvers
Multigrid
Methods

Introduction: Linear System Solvers

- **Much of recent work on solvers has focussed on:**

(1) Parallel implementation – scalable performance

(2) Improving Robustness, developing more general preconditioners

A few observations

- **Problems are getting harder for Sparse Direct methods**

(more 3-D models, much bigger problems,..)

- **Problems are also getting difficult for iterative methods Cause:**

more complex models - away from Poisson

- **Researchers in iterative methods are borrowing techniques from direct methods: → preconditioners**

- **The inverse is also happening: Direct methods are being adapted for use as preconditioners**

What are sparse matrices?

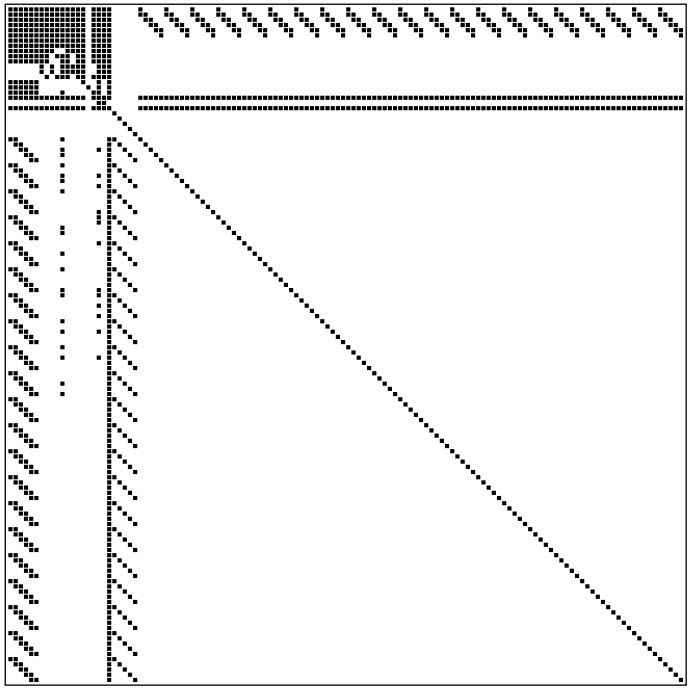
Common definition: "..matrices that allow special techniques to take advantage of the large number of zero elements and the structure."

A few applications of sparse matrices: Structural Engineering, Reservoir simulation, Electrical Networks, optimization problems, ...

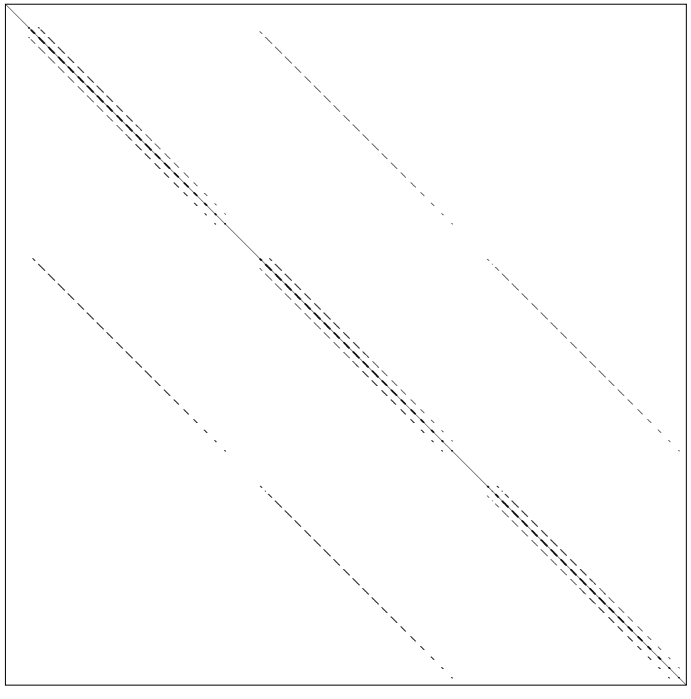
Goals: Much less storage and work than dense computations.

Observation: A^{-1} is usually dense, but L and U in the LU factorization may be reasonably sparse (if a good technique is used).

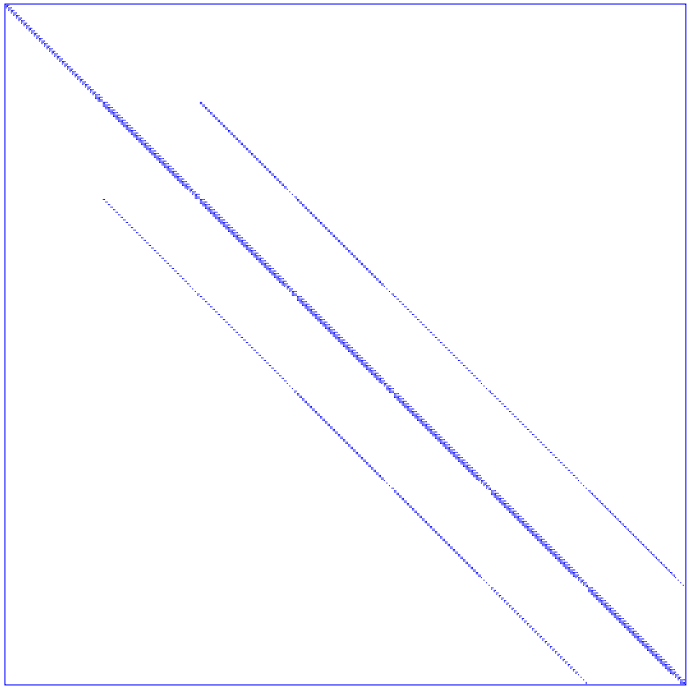
Nonzero patterns of a few sparse matrices



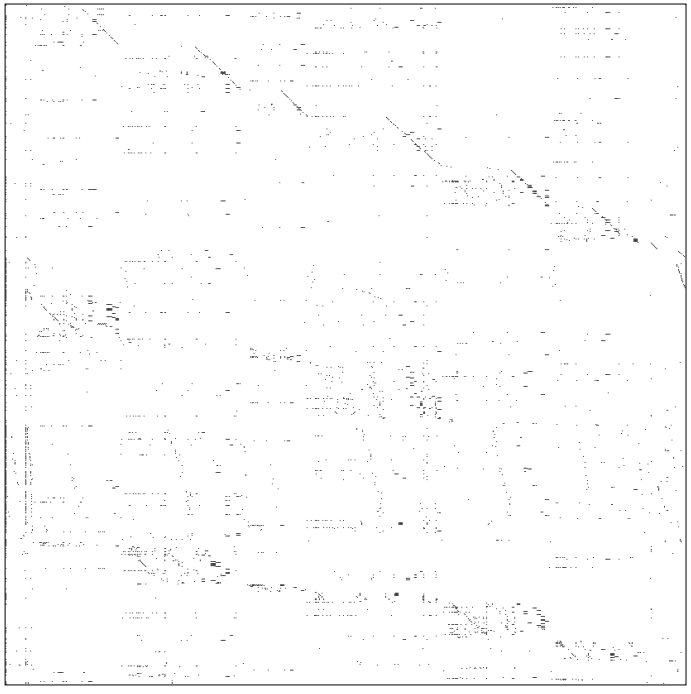
ARC130: Unsymmetric matrix from laser problem. a.r.curtis, oct 1974



SHERMAN5: fully implicit black oil simulator 16 by 23 by 3 grid, 3 unk



PORES3: Unsymmetric MATRIX FROM PORES



BP_1000: UNSYMMETRIC BASIS FROM LP PROBLEM BP

➤ **Two types of matrices:** structured (e.g. Sherman5) and unstructured (e.g. BP_1000)

➤ **Main goal of Sparse Matrix Techniques:** To perform standard matrix computations economically i.e., without storing the zeros of the matrix.

➤ **Example:** To add two square dense matrices of size n requires $O(n^2)$ operations. To add two sparse matrices A and B requires $O(nnz(A) + nnz(B))$ where $nnz(X) =$ number of nonzero elements of a matrix X.

➤ For typical Finite Element /Finite difference matrices, number of nonzero elements is $O(n)$.

Graph Representations of Sparse Matrices

➤ **Graph theory is a fundamental tool in sparse matrix techniques.**

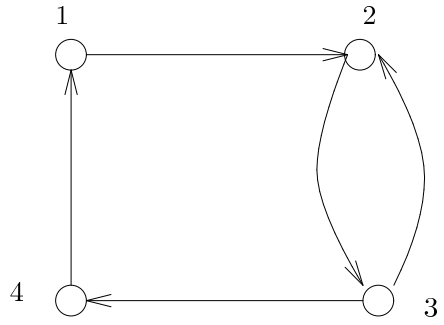
Graph $G = (V, E)$ of an $n \times n$ matrix A defined by

Vertices $V = \{1, 2, ..., N\}$.

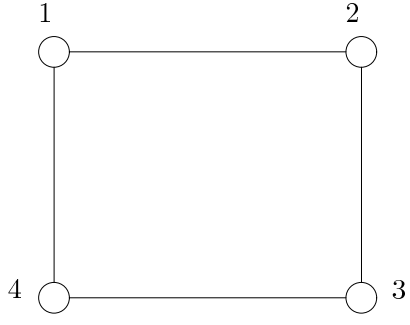
Edges $E = \{(i, j)|a_{ij} \neq 0\}$.

➤ **Graph is undirected if matrix has symmetric structure: $a_{ij} \neq 0$ iff $a_{ji} \neq 0$.**

x	x		
	x	x	
	x	x	x
x			x



x	x		x
x	x	x	
	x	x	x
x		x	x



Example: Adjacency graph of:

$$A = \begin{pmatrix} \star & \star & & & \star & \\ \star & \star & \star & & & \star \\ & \star & \star & & & \\ & & & \star & \star & \\ \star & & & \star & \star & \star \\ & \star & & & \star & \star \end{pmatrix}.$$

Example: For any matrix A, what is the graph of A^2? [interpret in terms of paths in the graph of A]

Direct versus iterative methods

Background. Two types of methods:

- **Direct methods : based on sparse Gaussian eimination, sparse Cholesky,..**

- **Iterative methods: compute a sequence of iterates which converge to the solution - preconditioned Krylov methods..**

Remark: **These two classes of methods have always been in competition.**

- **40 years ago solving a system with $n = 10,000$ was a challenge**

- **Now you can solve this in < 1 sec. on a laptop.**

➤ Sparse direct methods made huge gains in efficiency. As a result they are very competitive for 2-D problems.

➤ 3-D problems lead to more challenging systems [inherent to the underlying graph]

➤ Problems with many unknowns per grid point similar to 3-D problems

Remarks:

- No robust 'black-box' iterative solvers.
- Robustness often conflicts with efficiency
- However, situation improved in last $\approx$ decade
- Line between direct and iterative solvers blurring

Direct Sparse Matrix Techniques

Principle of sparse matrix techniques: **Store only the nonzero elements of A. Try to minimize computations and (perhaps more importantly) storage.**

➤ **Difficulty in Gaussian elimination: Fill-in**

Trivial Example:

$$A = \begin{pmatrix} + & + & + & + & + & + \\ + & + & & & & \\ + & & + & & & \\ + & & & + & & \\ + & & & & + & \\ + & & & & & + \end{pmatrix}$$

➤ **Reorder equations and unknowns in order** $N, N-1, ..., 1$

➤ A **stays sparse during Gaussian eliminatin – i.e., no fill-in.**

$$A = \begin{pmatrix} + & & & & & + \\ & + & & & & + \\ & & + & & & + \\ & & & + & & + \\ & & & & + & + \\ + & + & + & + & + & + \end{pmatrix}$$

➤ **Finding the best ordering to minimize fill-in is NP-complete.**

➤ **A number of heuristics developed. Among the best known:**

- **Minimum degree ordering (Tinney Scheme 2)**
- **Nested Dissection Ordering.**
- **Approximate Minimal Degree ...**

Reorderings and graphs

- Let $\pi = \{i_1, \cdots, i_n\}$ a permutation
- $A_{\pi,*} = \left\{a_{\pi(i),j}\right\}_{i,j=1,\ldots,n}$ = matrix A with its i-th row replaced by row number $\pi(i)$.
- $A_{*,\pi}$ = matrix A with its j-th column replaced by column $\pi(j)$.
- Define $P_\pi = I_{\pi,*}$ = "Permutation matrix" – Then:

(1) Each row (column) of P_π consists of zeros and exactly one "1"

(2) $A_{\pi,*} = P_\pi A$

(3) $P_\pi P_\pi^T = I$

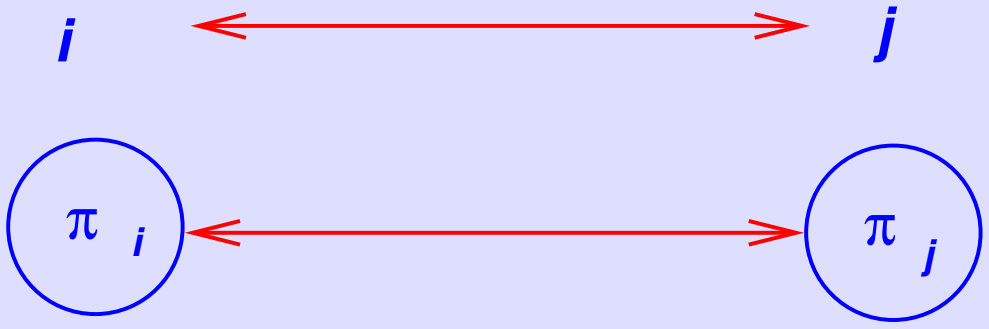
(4) $A_{*,\pi} = A P_\pi^T$

Consider now: $A' = A_{\pi,\pi} = P_\pi A P_\pi^T$

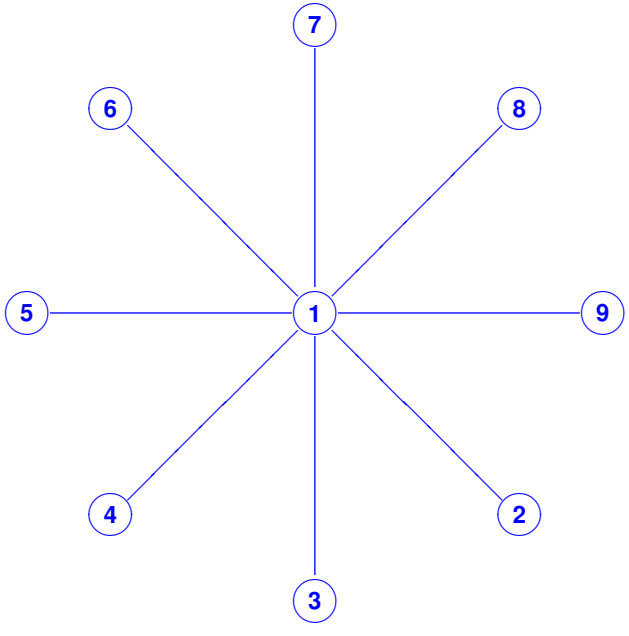
➤ **Entry (i,j) in matrix A' is exactly entry in position $(\pi(i), \pi(j))$ in A, i.e., $(a'_{ij} = a_{\pi(i),\pi(j)})$**

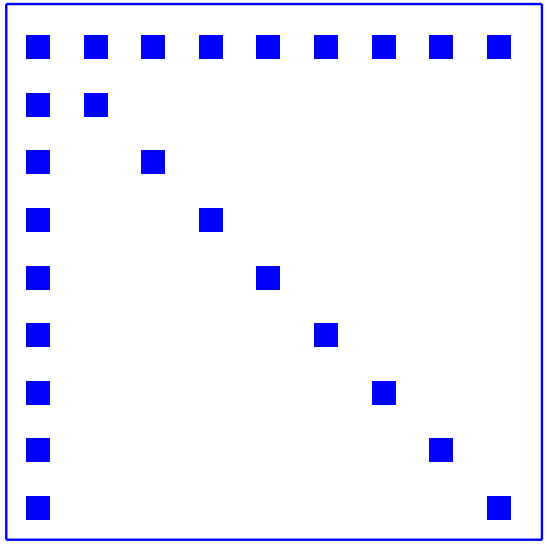
$$(i,j) \in E_{A'} \iff (\pi(i), \pi(j)) \in E_A$$

General picture :

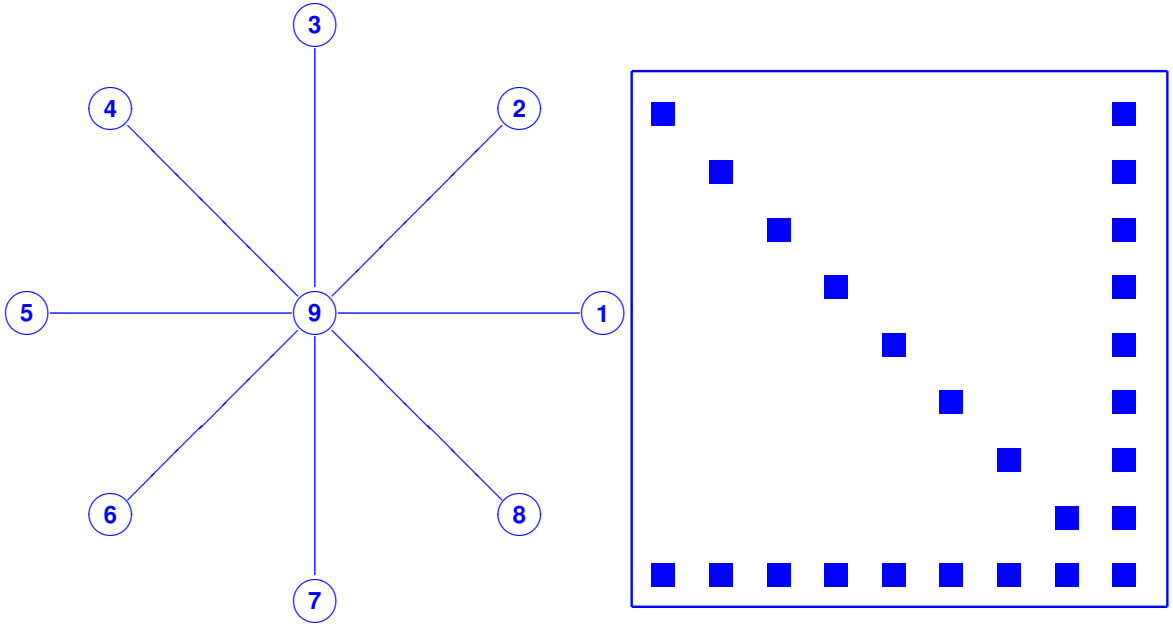


Example A 9 $\times$ 9 'arrow' matrix and its adjacency graph.





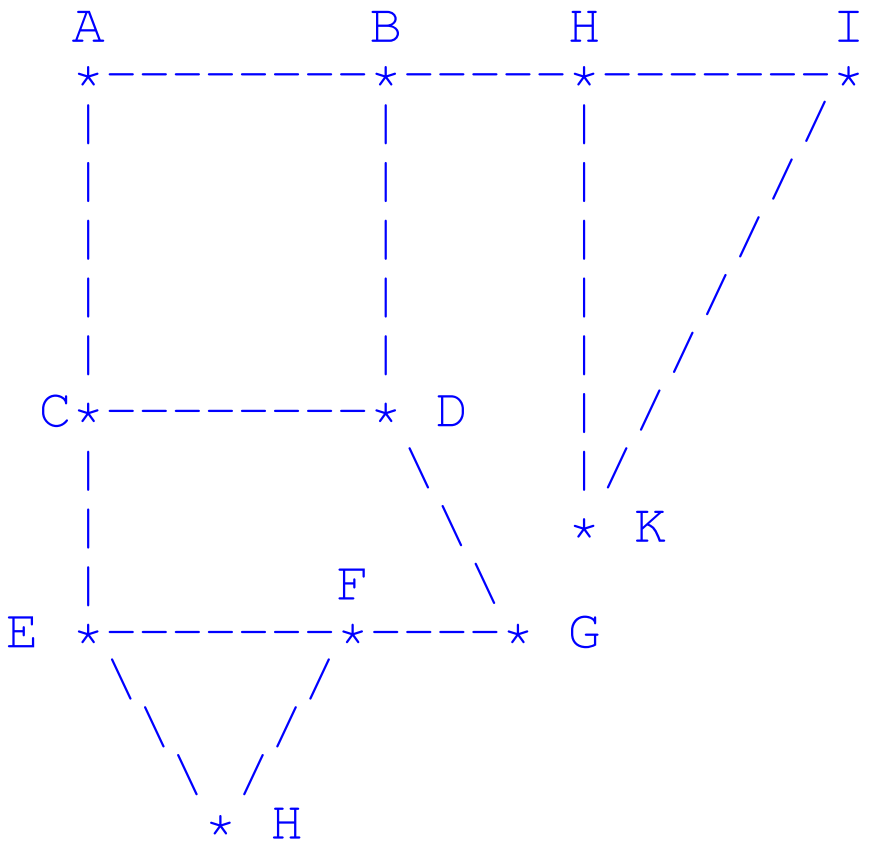
Graph and matrix after permuting the nodes in reverse order.



Cuthill-McKee & reverse Cuthill-McKee

- A class of reordering techniques proceeds by levels in the graph.
- Related to Breadth First Search (BFS) traversal in graph theory.
- Idea of BFS is to visit the nodes by 'levels'. Level 0 = level of starting node.
- Start with a node, visit its neighbors, then the (unmarked) neighbors of its neighbors, etc...

Example:



BFS from node A:
Level 0: A
Level 1: B, C;
Level 2: E, D, H;
Level 3: I, K, E, F, G, H.

Implementation using levels

Algorithm $BFS(G, v)$ – by level sets –

- Initialize $S = \{v\}$, $seen = 1$; Mark v;
- While $seen < n$ Do
  - $S_{new} = \emptyset$;
  - For each node v in S do
    - $*$ For each unmarked w in adj(v) do
      - $\cdot$ Add w to S_{new};
      - $\cdot$ Mark w;
      - $\cdot$ $seen++$;
  - $S := S_{new}$

A few properties of Breadth-First-Search

- If G is a connected undirected graph then each vertex will be visited once each edge will be inspected at least once

- Therefore, for a connected undirected graph,

  The cost of BFS is $O(|V| + |E|)$

- Distance = level number; - For each node v we have:

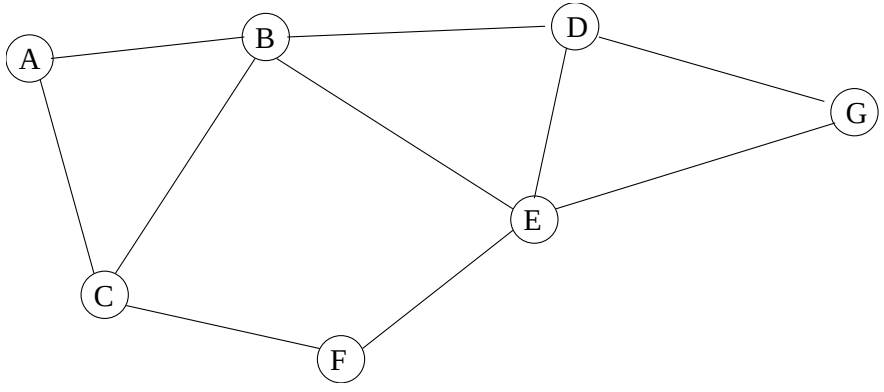
$$min\_dist(s, v) = level\_number(v) = depth_T(v)$$

- Several reordering algorithms are based on variants of Breadth-First-Search

Cuthill McKee ordering

Algorithm proceeds by levels. Same as BFS except: in each level, nodes are ordered by increasing degree

Example



Level	Nodes	Deg.	Order
0	A	2	A
1	B, C	4, 3	C, B
2	D, E, F	3, 4, 2	F, D, E
3	G	2	G

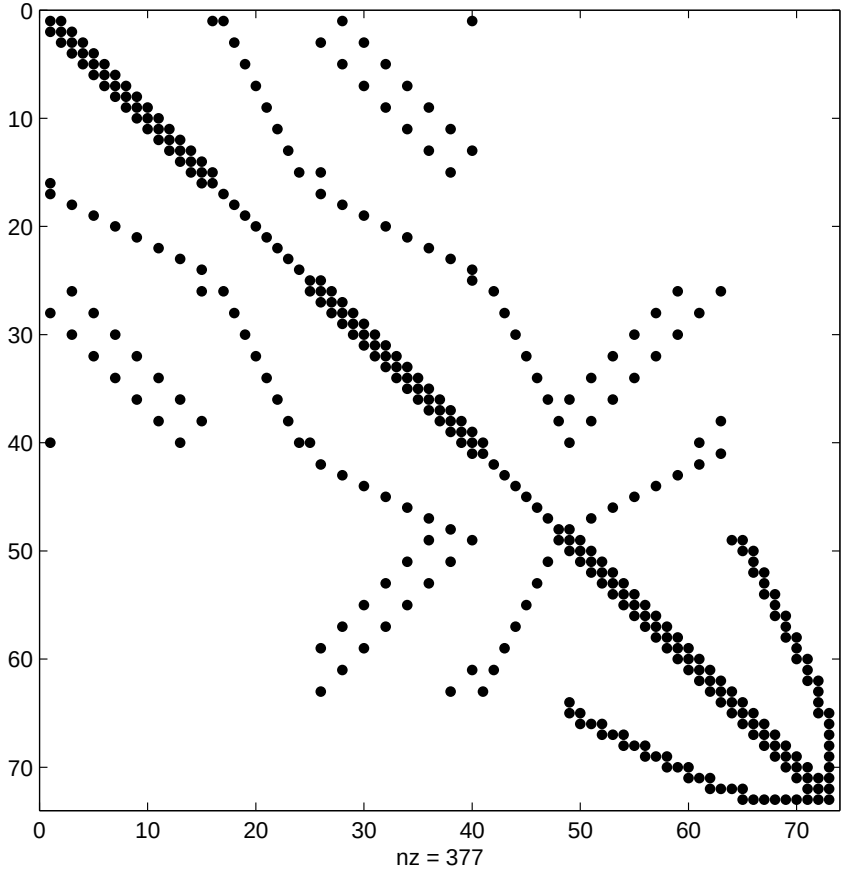
ALGORITHM : 1. *Cuthill Mc Kee ordering*

0. *Find an intial node for the traversal*
1. *Initialize* $S = \{v\}$, $seen = 1$, $\pi(seen) = v$; *Mark* v;
2. *While* $seen < n$ *Do*
3. $S_{new} = \emptyset$;
4. *For each node* v, *going from lowest to highest degree, Do:*
5. $\pi(++seen) = v$;
6. *For each unmarked* w *in adj(*v*) do*
7. *Add* w *to* S_{new};
8. *Mark* w;
9. *EndDo*
10. $S := S_{new}$
11. *EndDo*
12. *EndWhile*

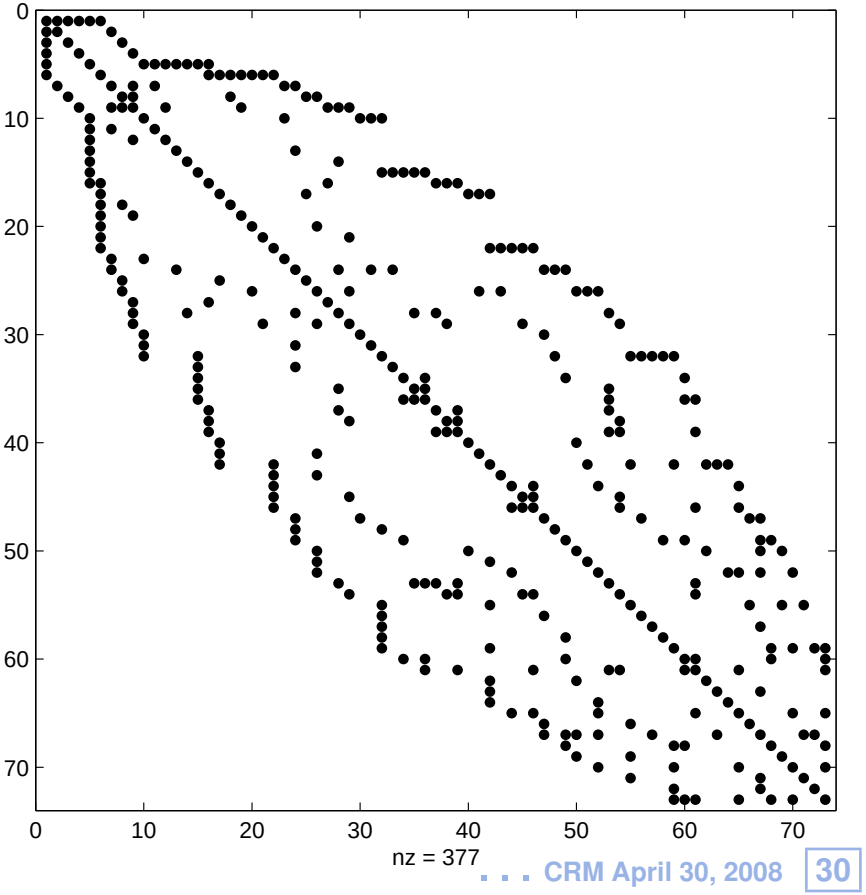
Reverse Cuthill McKee ordering

➤ **The Cuthill - Mc Kee ordering has a tendency to create small arrow matrices (going the wrong way):**

Origimal matrix

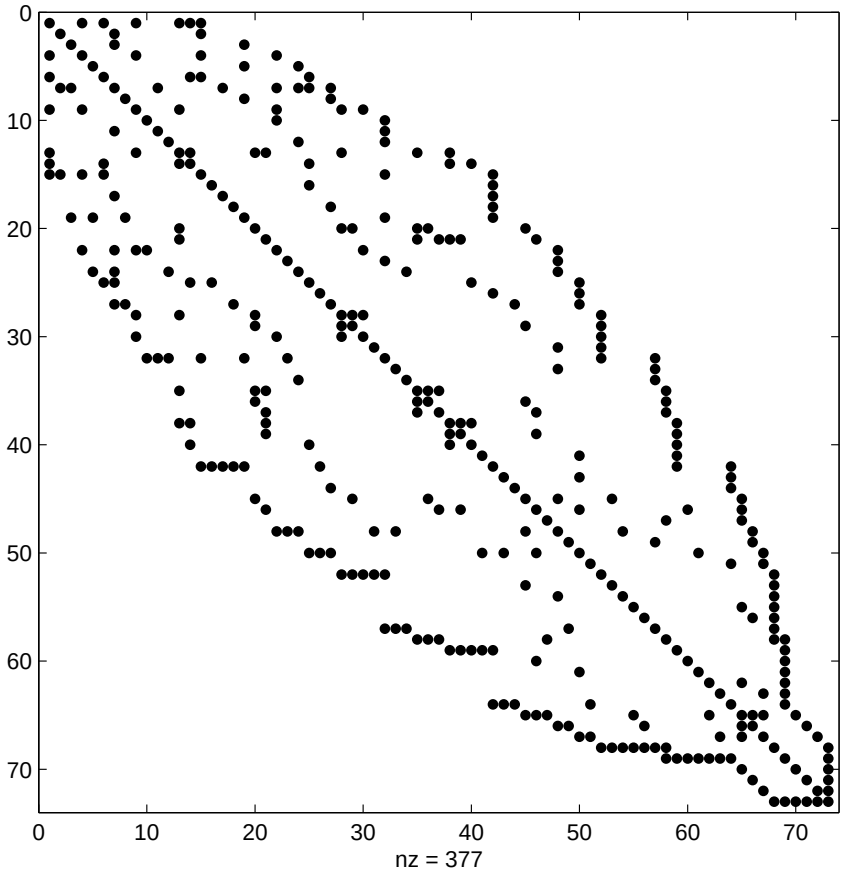


CM ordering



- **Idea: Take the reverse ordering**

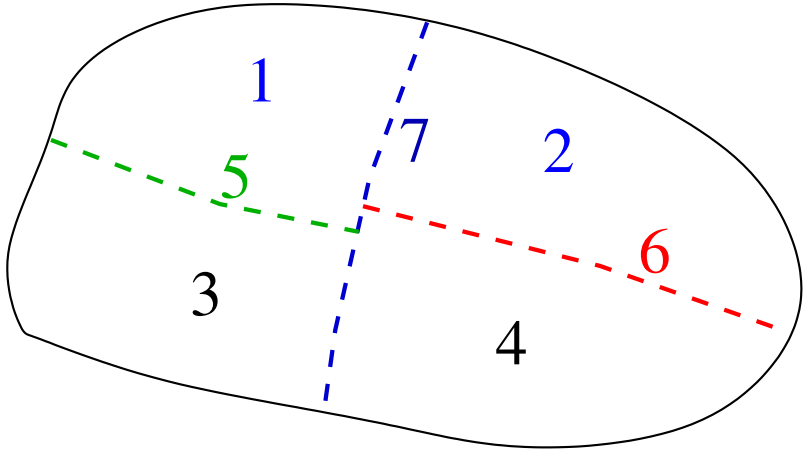
RCM ordering

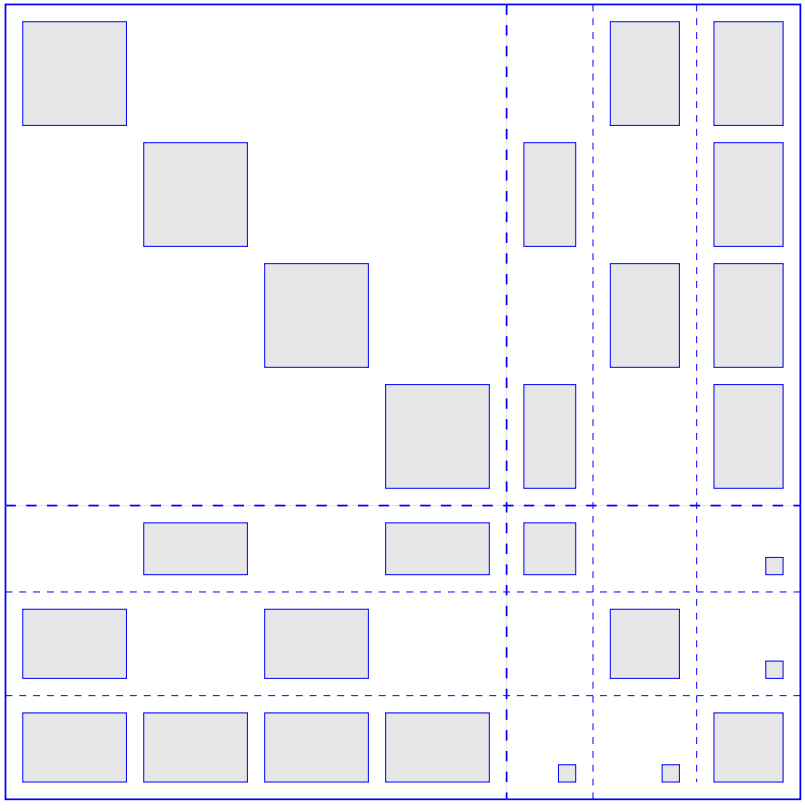


- **Reverse Cuthill M Kee ordering (RCM).**

Nested Dissection ordering

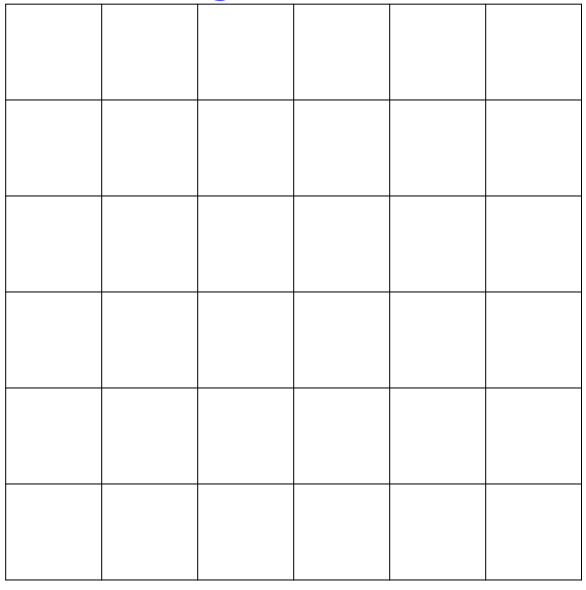
➤ **The idea of divide and conquer – recursively divide graph in two using a separator.**



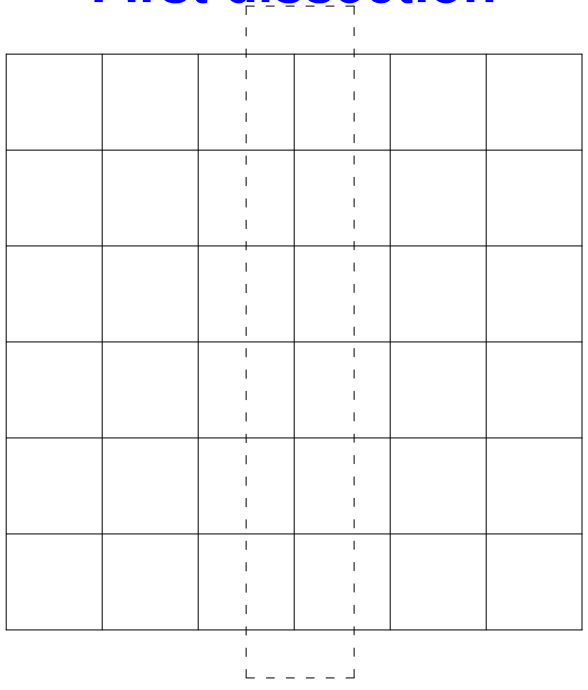


Nested dissection for a small mesh

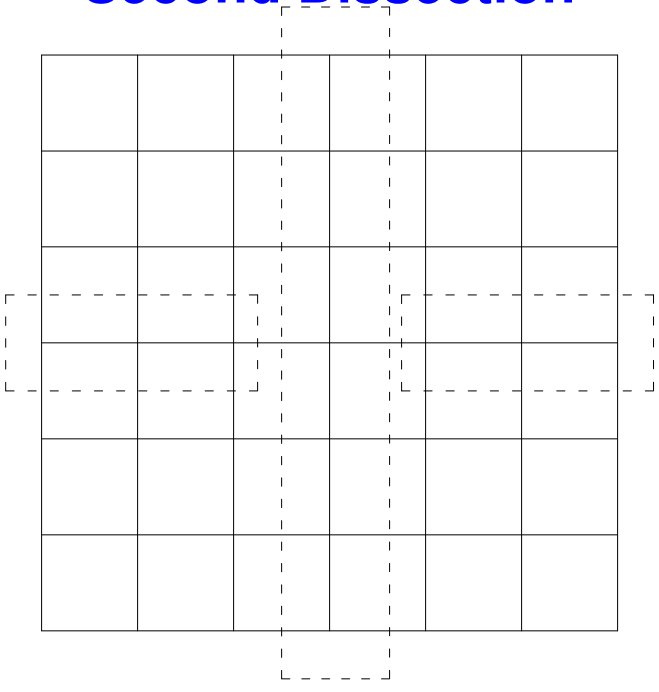
Original Grid



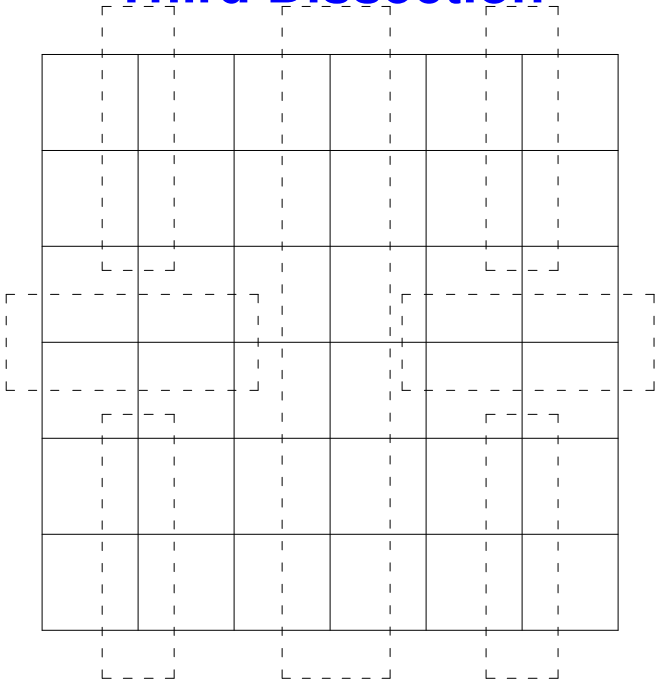
First dissection



Second Dissection



Third Dissection



Nested dissection: cost for a regular mesh

- In 2-D consider an $n \times n$ problem, $N = n^2$
- In 3-D consider an $n \times n \times n$ problem, $N = n^3$

	2-D	3-D
space (fill)	$O(N \log N)$	$O(N^{4/3})$
time (flops)	$O(N^{3/2})$	$O(N^2)$

- Significant difference in complexity between 2-D and 3-D

Ordering techniques for direct methods in practice

- **In practice: Nested dissection (+ variants) is preferred for parallel processing**

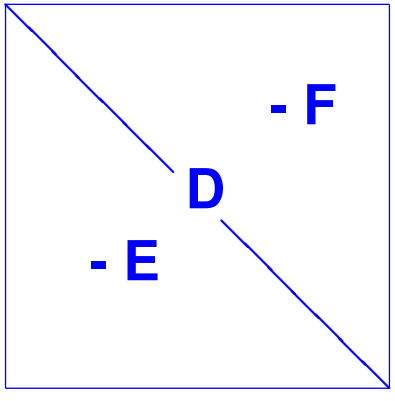
- **Good implementations of Min. Degree algorithm work well in practice. Currently AMD and AMF are best known implementations/variants/**

- **Best practical reordering algorithms usually combine Nested dissection and min. degree algorithms.**

BASIC RELAXATION METHODS

Basic Relaxation Schemes

Relaxation schemes: based on the decomposition $A = D - E - F$



D = diag(A), $-E$ = strict lower part of A and $-F$ its strict upper part.

Gauss-Seidel iteration for solving $Ax = b$:

$$(D - E)x^{(k+1)} = Fx^{(k)} + b$$

$\longrightarrow$ **idea: correct the** j**-th component of the current approximate solution,** $j = 1, 2, ..n$**, to zero the** $j - th$ **component of residual.**

Can also define a backward Gauss-Seidel Iteration:

$$(D - F)x^{(k+1)} = Ex^{(k)} + b$$

and a Symmetric Gauss-Seidel Iteration: forward sweep followed by backward sweep.

Over-relaxation is based on the decomposition:

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega)D)$$

$\longrightarrow$ successive overrelaxation, (SOR):

$$(D - \omega E)x^{(k+1)} = [\omega F + (1 - \omega)D]x^{(k)} + \omega b$$

Iteration matrices

Jacobi, Gauss-Seidel, SOR, & SSOR iterations are of the form

$$x^{(k+1)} = Mx^{(k)} + f$$

- $M_{Jac} = D^{-1}(E+F) = I - D^{-1}A$
- $M_{GS}(A) = (D-E)^{-1}F == I - (D-E)^{-1}A$
- $M_{SOR}(A) = (D-\omega E)^{-1}(\omega F + (1-\omega)D) = I - (\omega^{-1}D - E)^{-1}A$
- $M_{SSOR}(A) = I - (2\omega^{-1}-1)(\omega^{-1}D - F)^{-1}D(\omega^{-1}D - E)^{-1}A$
  $= I - \omega(2\omega - 1)(D - \omega F)^{-1}D(D - \omega E)^{-1}A$

General convergence result

Consider the iteration: $x^{(k+1)} = Gx^{(k)} + f$

(1) Assume that $\rho(A) < 1$. Then $I - G$ is non-singular and G has a fixed point. Iteration converges to a fixed point for any f and $x^{(0)}$.

(2) If iteration converges for any f and $x^{(0)}$ then $\rho(G) < 1$.

Example: Richardson's iteration $x^{(k+1)} = x^{(k)} + \alpha(b - A^{(k)})$

$\diamondsuit$Assume $\Lambda(A) \subset \mathbb{R}$. When does the iteration converge?

- **Jacobi and Gauss-Seidel converge for diagonal dominant A**
- **SOR converges for $0 < \omega < 2$ for SPD matrices**

An observation. Introduction to Preconditioning

➤ **The iteration $x^{(k+1)} = Mx^{(k)} + f$ is attempting to solve $(I - M)x = f$. Since M is of the form $M = I - P^{-1}A$ this system can be rewritten as**

$$P^{-1}Ax = P^{-1}b$$

where for SSOR, we have

$$P_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

referred to as the SSOR 'preconditioning' matrix.

In other words:

Relaxation Scheme $\Longleftrightarrow$ Preconditioned Fixed Point Iteration