SPARSE MATRIX COMPUTATIONS
Lecture 2

Ali Pınar

February 11, 1999

Outline

- Review of row(column) compressed storage.
- Basic operations.
  - Permuting a matrix.
  - Sparse matrix vector multiplication.
- Permuting matrices to special forms.
  - Reducing the bandwidth of a matrix.
  - Permuting to block triangular form.

Row (Column) Compressed Storage

- Store the matrix as a collection of compressed rows (columns).
- Requires three arrays
  - A_{fl} stores the nonzero floating point numbers in the matrix row by row.
    A_{fl}=[nonzeros of the first row, nonzeros of the second row, ...]
  - A_{col} is an integer array which stores the column number of entries in A_{float},
    A_{col}=[column number of $A_{fl}[0]$, column number of $A_{fl}[1]$, ...]
  - A_{ptr} is an integer array which stores the indices where the nonzeros of a row start in arrays A_{fl} and A_{col}.

  A_{ptr}=[0,ptr to nonzeros of the 2nd row, ptr to nonzeros of the 3rd row]

Row Compressed Storage Example

$$A = \begin{pmatrix} 1. & 0 & 0 & -1. & 0 \\ 2. & 0 & -2. & 0 & 3. \\ 0 & -3. & 0 & 0 & 0 \\ 0 & 4. & 0 & -4. & 0 \\ 5. & 0 & -5. & 0 & 6. \end{pmatrix}$$

$$A_{float} = [\ 1.,\ -1.,\ 2.,\ -2.,\ 3.,\ -3.,\ 4.,\ -4.,\ 5.,\ -5.,\ 6\ ]$$

$$A_{col} = [\ 1,\ 4,\ 1,\ 3,\ 5,\ 2,\ 2,\ 4,\ 1,\ 3,\ 5\ ]$$

$$A_{ptr} = [1, 3, 6, 7, 9, 12]$$

Basic Operations Using Row Compressed Storage

- Access a row (multiply ith row with a constant, c).
  - Row Compressed Storage

$$\begin{array}{l} \textbf{for } j = A_{ptr}[i] \textbf{ to } A_{ptr}[i+1]-1 \\ \qquad A_{fl}[j] = A_{fl}[j] * c; \end{array}$$

  - Column Compressed Storage

    Requires traversing all nonzeros.

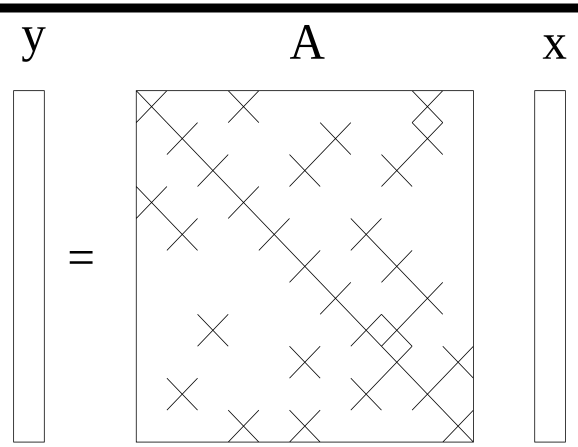
Basic Operations Using Row Compressed Storage

- Permuting rows (columns)
  Let $map[i]$ denote the new position of row/column i in the permuted matrix.
  - Permuting columns

    $$\begin{array}{l} \textbf{for } i = 0 \textbf{ to } Nz - 1 \\ \qquad A_{col}[i] = map[A_{col}[i]]; \end{array}$$

  - Permuting rows
    * Requires extra storage.
    * Reorganize the whole structure.

Sparse Matrix Vector Multiplication (SpMxV)



- The algorithm

  for i=0 **to** N

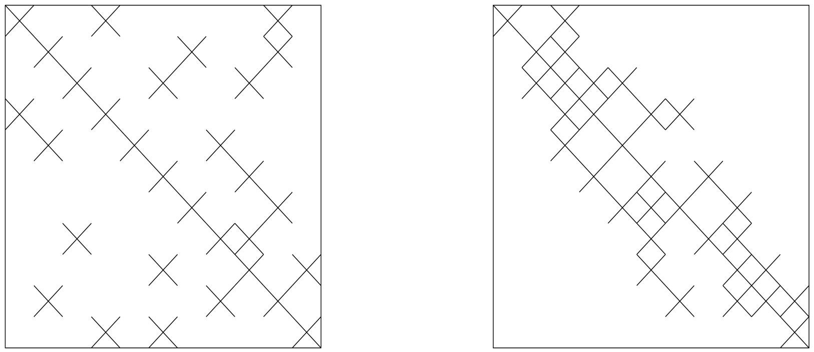
  y[i]=0;

  for j=$A_{ptr}[i]$ **to** $A_{ptr}[i+1]-1$

  $y[i] = y[i] + A_{fl}[j] * x[A_{col}[j]];$

- Problems
  - Cache utilization.
  - Overhead due to extra load operations.

Cache Utilization for SpMxV



(+) Spatial Locality on A.

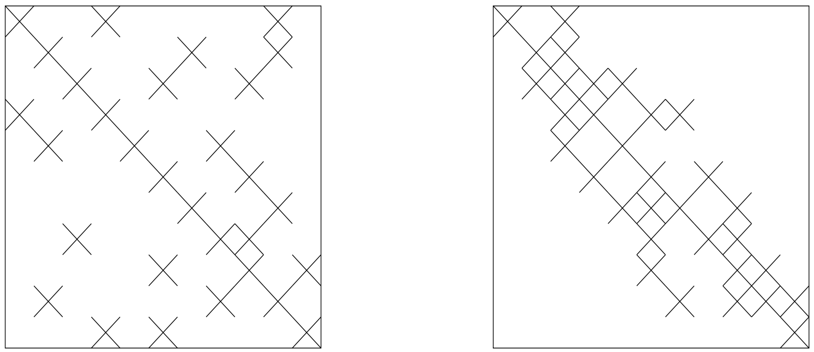
(-) Random access pattern for vector(s).

- Permute the matrix to increase locality.

  – Better spatial locality (the chance that $x_{i+\epsilon}$ will be accessed after x_i).

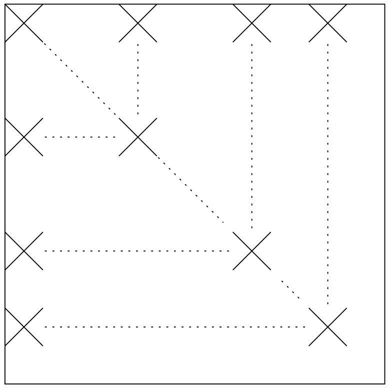
  – Better temporal locality (the chance that x_i will be accessed again soon).

Reordering for small bandwidth



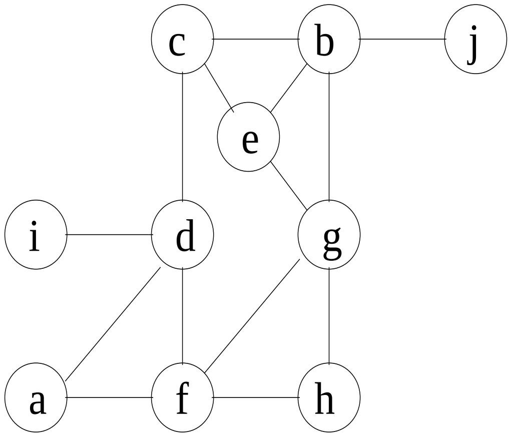
- Bandwidth is a measure of how close the nonzeros are to the diagonal.
- A matrix has bandwidth m, if $a_{ij} = 0$ when $|i - j| > m$.
- Almost all techniques are variants of Cuthill-McKee ordering.
- Most popular technique is Reverse Cuthill-McKee ordering.

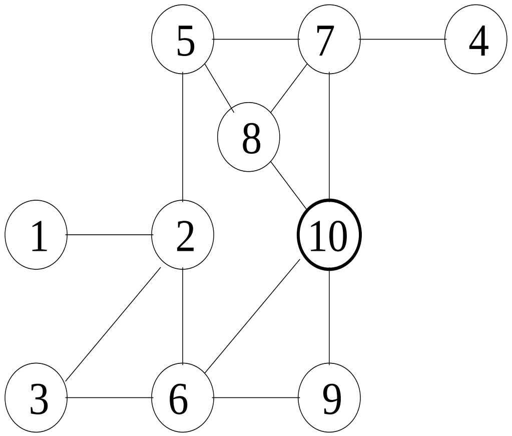
Reverse Cuthill-McKee Ordering

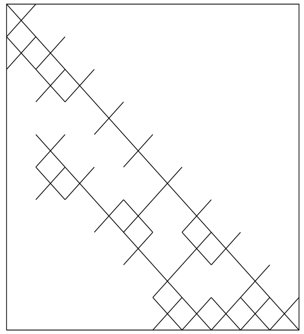


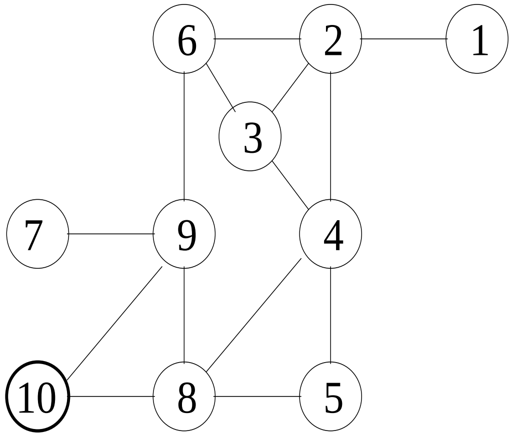
- Determine a starting vertex, v_1.
- **for** $i = 1$ **to** N, find all unordered neighbors of the node x_i and number them in increasing order of degree.
- Reverse the ordering.

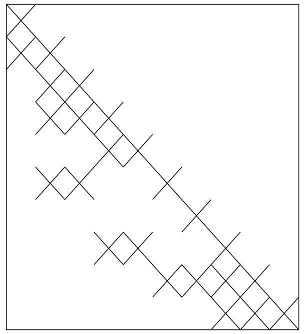
Reverse Cuthill-McKee Ordering Example











Block Triangular Form

$$PAQ = \begin{pmatrix} B_{11} & & & & \\ B_{21} & B_{22} & & & \\ B_{31} & B_{32} & B_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ B_{N1} & B_{N2} & B_{N3} & \dots & B_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ b_3 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{pmatrix}$$

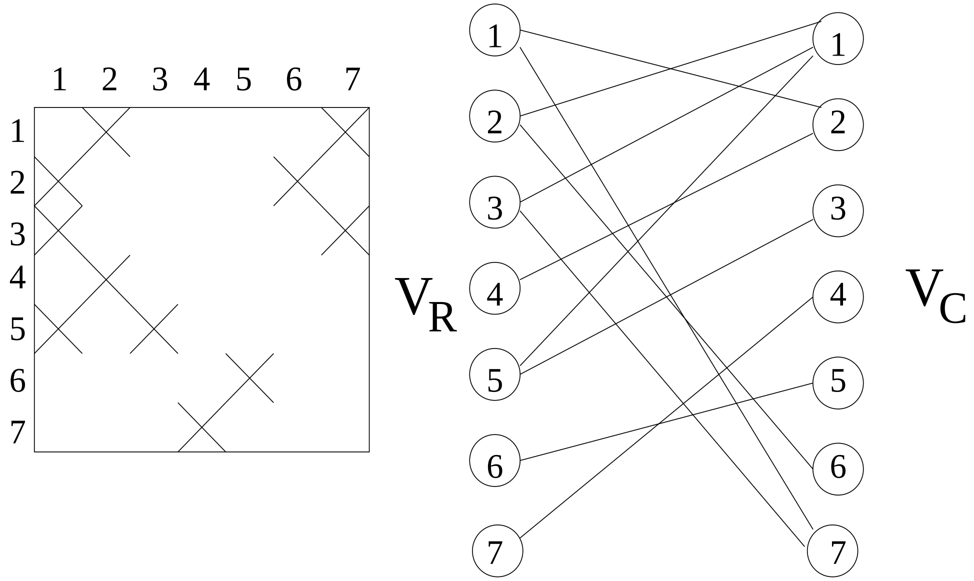
- Why Block Triangular Form (BTF)?
  - Only the blocks on the diagonal are factorized.
- Applicability
  - Not applicable to matrices arising from discretization of the PDEs.
  - Linear programming, chemical process design matrices are often reducible.

Permuting Block Triangular Form

$$PAQ = \begin{pmatrix} B_{11} & & & & \\ B_{21} & B_{22} & & & \\ B_{31} & B_{32} & B_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ B_{N1} & B_{N2} & B_{N3} & \dots & B_{NN} \end{pmatrix}$$

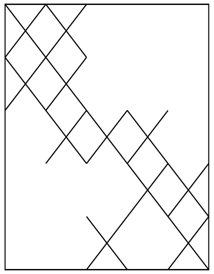
- Permutation is essentially unique.
- Phase 1: *Permute entries onto the diagonal.*
  - Bipartite graph model for sparse matrices.
  - Maximum Matching on a bipartite graph.
- Phase 2: *Use symmetric permutations to find the block form itself.*
  - Directed graph model for sparse matrices.
  - Strongly connected components.

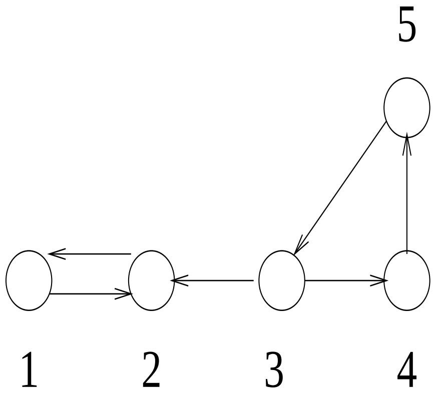
Bipartite Graph Model for Sparse Matrices



- Each row is represented by a vertex in $\mathcal{V}_R$ and each column is represented by a vertex in $\mathcal{V}_C$.
- $(\mathcal{V}_R, \mathcal{V}_C)$ is a bipartition of the graph.
- $v_i \in \mathcal{V}_R$ is adjacent to $v_j \in \mathcal{V}_C$ iff a_{ij} is non-zero in the matrix.

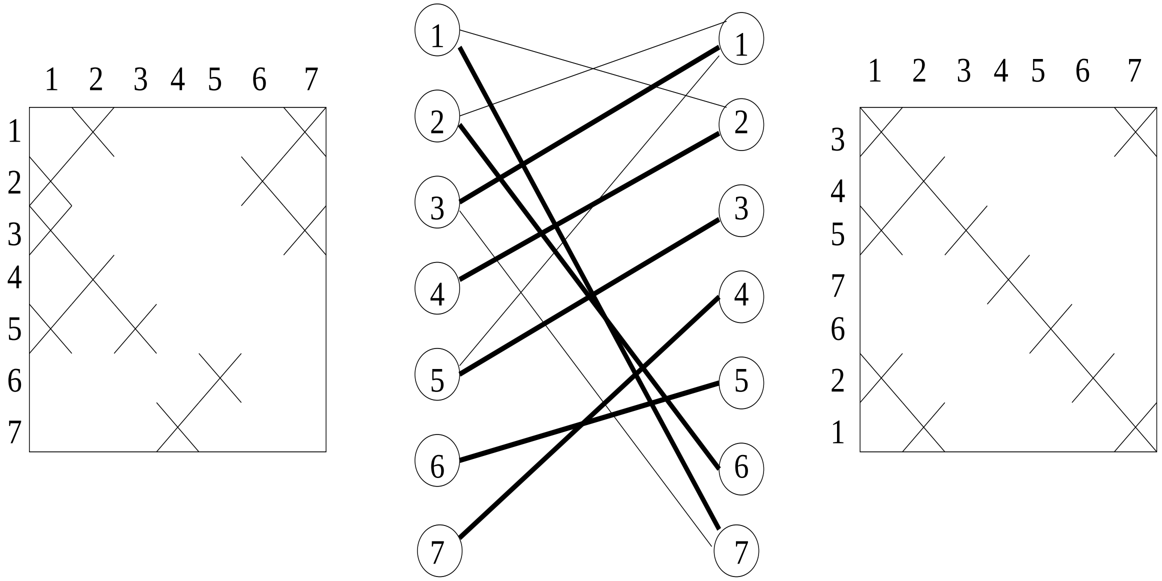
Directed Graph Model for Sparse Matrices





- Each row/column is represented by a vertex.
- There is an edge from v_i to v_j iff $a_{ij} \neq 0$.

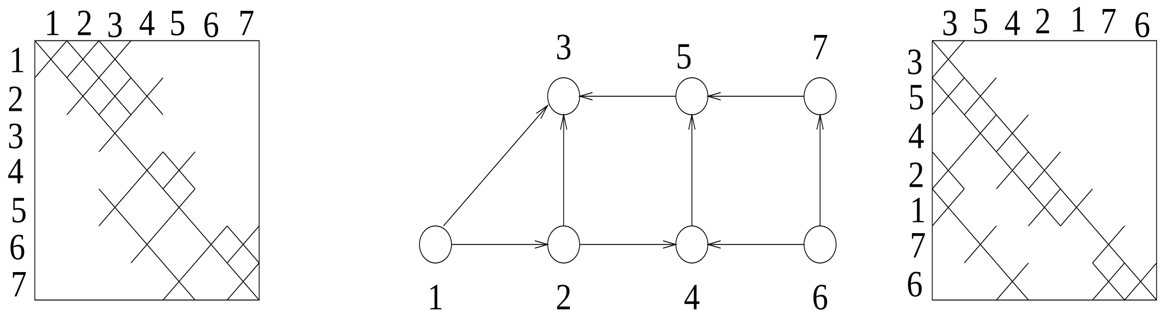
Permuting to Block Triangular Form (Phase 1)



Bipartite Graph Matching:

- Match each vertex with another. Each vertex should be matched exactly once. Can be solved in $O(|V||E|)$.
- Permute rows according to the matched columns.
  If Row i is matched to column j, then it is mapped to j th position in the reordered matrix.

Permuting to Triangular Form (Phase 2)



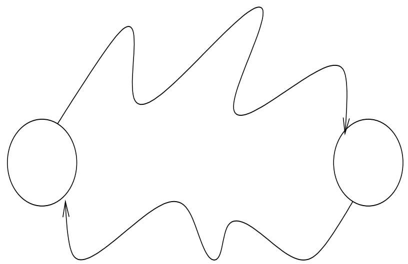
for $i = 1, 2, \ldots N$

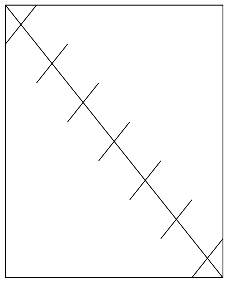
find a vertex, with no edges leaving.

map this vertex to the ith position.

delete all edges coming to it.

Permuting to Block Triangular Form (Phase 2)





In a directed graph, a set of vertices form a *strongly connected component*, if there is a path form u to v for every u, v in the set.

- Identify the strongly connected components, collapse them.
- Find a permutation of strongly connected components, as described above.

Summary

- Basic operations with sparse data structures
- Performance restrictions
  - Poor cache utilization.
  - Extra load operations.
- Improving Performance by permutation.
  - Bandwidth reduction.
  - Block triangular form.