Reorderings and graphs

- Let $\pi = \{i_1, \cdots, i_n\}$ a permutation
- $A_{\pi,*} = \{a_{\pi(i),j}\}_{i,j=1,...,n}$ = 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 $P_\pi$ 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:

- Element in position $(i, j)$ in matrix $A'$ is exactly element in position $(\pi(i), \pi(j))$ in $A$. ($a'_{ij} = a_{\pi(i),\pi(j)}$)

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

General Picture:

- General Picture:

Example: A 9 × 9 ‘arrow’ matrix and its adjacency graph.
The Cuthill-McKee and its reverse orderings

A class of reordering techniques which proceed 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:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>E</td>
<td>G</td>
<td>J</td>
<td>K</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm $BFS(G,v)$ – Queue implementation

- Initialize: $Queue := \{v\}$; Mark $v$; $ptr = 1$
- While $ptr < \text{length}(Queue)$ do
  - $head = Queue(ptr)$;
  - ForEach Unmarked $w \in \text{Adj}(head)$:
    - Mark $w$;
    - Add $w$ to $Queue$: $Queue = \{Queue, w\}$
  - $ptr ++$;

Final traversal order:

Level 0: $A, B, C, D, I, E, G, J, K, F, H$

Level 1: $A, B, C, D, I, E, G, J, K, F, H$


Levels represent distances from the root

Algorithm can be implemented by crossing levels 1,2, ...

More common: Queue implementation
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:
  \[
  \text{min\_dist}(s, v) = \text{level\_number}(v) = \text{depth}_T(v)
  \]
- Several reordering algorithms are based on variants of Breadth-First-Search.

Cuthill McKee ordering

Same as BFS except: Adj(head) always sorted by increasing degree.

Rule: when adding nodes to the queue list them in ↑ deg.

Example:

<table>
<thead>
<tr>
<th>Original matrix</th>
<th>CM ordering</th>
</tr>
</thead>
</table>
| \[ \begin{array}{ccc}
A & B(4) & C(3) \\
A, C & B & F(2) \\
A, C, B & F, D(3), E(4) \\
A, C, B, F & D, E \\
A, C, B, F, D & E, G(2) \\
A, C, B, F, D, E & G \\
A, C, B, F, D, E, G & \end{array} \] |
| \[ \begin{array}{ccc}
0 & 10 & 20 \\
20 & 30 & 40 \\
30 & 40 & 50 \\
40 & 50 & 60 \\
50 & 60 & 70 \\
60 & 70 & 0 \\
70 & 0 & 10 \\
10 & 0 & 20 \\
20 & 10 & 0 \\
30 & 20 & 0 \\
40 & 30 & 0 \\
50 & 40 & 0 \\
60 & 50 & 0 \\
70 & 60 & 0 \\
\end{array} \] |
Idea: Take the reverse ordering
Reverse Cuthill M Kee ordering (RCM).

\[ \beta_i(A) = \max_{j \leq i; a_{ij} \neq 0} |i - j| \]

Definition: Envelope of \( A \) is the set of all pairs \((i, j)\) such that \( 0 < i - j \leq \beta_i(A) \). The quantity \(|\text{Env}(A)|\) is called profile of \( A \).

Main result: The envelope is preserved by GE (no-pivoting)

Theorem: Let \( A = LL^T \) the Cholesky factorization of \( A \). Then
\[ \text{Env}(A) = \text{Env}(L + L^T) \]

An envelope / profile/ Skyline method is a method which treats any entry \( a_{ij} \), with \((i, j) \in \text{Env}(A)\) as nonzero.

Matlab test: do the following
1. Generate \( A = \text{Lap2D}(64, 64) \)
2. Compute \( R = \text{chol}(A) \)
3. \( \text{show} \ \text{nnz}(R) \)
4. Compute RCM permutation (symrcm)
5. Compute \( B = A(p, p) \)
6. \( \text{spy}(B) \)
7. \( \text{compute} \ \text{R1} = \text{chol}(B) \)
8. Show \( \text{nnz}(R) \)
9. \( \text{spy}(R1) \)
**Orderings for parallelism: Multicoloring**

- General technique that can be exploited in many different ways to introduce parallelism – generally of order $N$.
- Constitutes one of the most successful techniques for introducing vector computations for iterative methods.
- Want: assign colors so that no two adjacent nodes have the same color.

**Simple example:** Red-Black ordering.

**Corresponding matrix**

- Observe: L-U solves with lower and upper parts of $A$ will require only diagonal scalings + matrix-vector products with matrices of size $N/2$.

**How to generalize Red-Black ordering?**

**Answer:** Multicoloring & independent sets

- A greedy multicoloring technique:
  - Initially assign color number zero (uncolored) to every node.
  - Choose an order in which to traverse the nodes.
  - Scan all nodes in the chosen order and at every node $i$ do

  $$\text{Color}(i) = \min\{k \neq 0 | k \neq \text{Color}(j), \forall j \in \text{Adj}(i)\}$$

  Adj(i) = set of nearest neighbors of $i = \{k | a_{ik} \neq 0\}$. 

**Text:** sec. 3.3 – coloring
### Independent Sets

An independent set (IS) is a set of nodes that are not coupled by an equation. The set is maximal if all other nodes in the graph are coupled to a node of IS. If the unknowns of the IS are labeled first, then the matrix will have the form:

\[
\begin{bmatrix}
B & F \\
E & C
\end{bmatrix}
\]

in which $B$ is a diagonal matrix, and $E$, $F$, and $C$ are sparse.

**Greedy algorithm:** Scan all nodes in a certain order and at every node $i$ do:

- if $i$ is not colored color it Red and color all its neighbors Black. Independent set: set of red nodes. Complexity: $O(|E| + |V|)$.

---

Show that the size of the independent set $I$ is such that

$$|I| \geq \frac{n}{1 + d_I}$$

where $d_I$ is the maximum degree of each vertex in $I$ (not counting self cycle).

According to the above inequality what is a good (heuristic) order in which to traverse the vertices in the greedy algorithm?

Are there situations when the greedy algorithm for independent sets yield the same sets as the multicoloring algorithm?
**Orderings used in direct solution methods**

- Two broad types of orderings used:
  - Minimal degree ordering + many variations
  - Nested dissection ordering + many variations
- Minimal degree ordering is easiest to describe:

  At each step of GE, select next node to eliminate, as the node $v$ of smallest degree. After eliminating node $v$, update degrees and repeat.

**Minimal Degree Ordering**

At any step $i$ of Gaussian elimination define for any candidate pivot row $j$

$$\text{Cost}(j) = (\text{nz}_c(j) - 1)(\text{nz}_r(j) - 1)$$

where $\text{nz}_c(j)$ = number of nonzero elements in column $j$ of ‘active’ matrix, $\text{nz}_r(j)$ = number of nonzero elements in row $j$ of ‘active’ matrix.

- Heuristic: fill-in at step $j$ is $\leq \text{cost}(j)$
- Strategy: select pivot with minimal cost.
- Local, greedy algorithm
- Good results in practice.

**Many improvements made over the years**


<table>
<thead>
<tr>
<th>Min. Deg. Algorithm</th>
<th>Storage (words)</th>
<th>Order. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final min. degree</td>
<td>1,181 K</td>
<td>43.90</td>
</tr>
<tr>
<td>Above w/o multiple elimn.</td>
<td>1,375 K</td>
<td>57.38</td>
</tr>
<tr>
<td>Above w/o elimn. absorption</td>
<td>1,375 K</td>
<td>56.00</td>
</tr>
<tr>
<td>Above w/o incompl. deg. update</td>
<td>1,375 K</td>
<td>83.26</td>
</tr>
<tr>
<td>Above w/o indistiguishible nodes</td>
<td>1,308 K</td>
<td>183.26</td>
</tr>
<tr>
<td>Above w/o mass-elimination</td>
<td>1,308 K</td>
<td>2289.44</td>
</tr>
</tbody>
</table>

- Results for a $180 \times 180$ 9-point mesh problem

- Since this article, many important developments took place.
- In particular the idea of “Approximate Min. Degree” and and “Approximate Min. Fill”, see
**Practical Minimal degree algorithms**

**First Idea:** Use quotient graphs
- Avoids elimination graphs which are not economical
- Elimination creates cliques
- Represent each clique by a node termed an *element* (recall FEM methods)
- No need to create fill-edges and elimination graph
- Still expensive: updating the degrees

**Second idea:** Multiple Minimum degree
- Many nodes will have the same degree. Idea: eliminate many of them simultaneously –
- Specifically eliminate independent sets of nodes with same degree.

**Third idea:** Approximate Minimum degree
- Degree updates are expensive –
- Goal: To save time.
- Approach: only compute an approximation (upper bound) to degrees.
- Details are complex and can be found in Tim Davis’ book

Explore *symamd* and *amd* in matlab

**Nested Dissection Reordering (Alan George)**
- Computer science ‘Divide-and-Conquer’ strategy.
- Best illustration: PDE finite difference grid.
- Easily described by using recursivity and by exploiting ‘separators’: ‘separate’ the graph in three parts, two of which have no coupling between them. The 3rd set (‘the separator’) has couplings with vertices from both of the first 2 sets.
- Key idea: dissect the graph; take the subgraphs and dissect them recursively.
- Nodes of separators always labeled last after those of the parents

For regular $n \times n$ meshes, can show: fill-in is of order $n^2 \log n$ and computational cost of factorization is $O(n^3)$

How does this compare with a standard band solver?
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, \( N = n^3 \)

<table>
<thead>
<tr>
<th></th>
<th>2-D</th>
<th>3-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>space (fill)</td>
<td>( O(N \log N) )</td>
<td>( O(N^{4/3}) )</td>
</tr>
<tr>
<td>time (flops)</td>
<td>( O(N^{3/2}) )</td>
<td>( O(N^2) )</td>
</tr>
</tbody>
</table>

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

Nested dissection and separators

- Nested dissection methods depend on finding a good graph separator: \( V = T_1 \cup U T_2 \cup S \) such that the removal of \( S \) leaves \( T_1 \) and \( T_2 \) disconnected.
- Want: \( S \) small and \( T_1 \) and \( T_2 \) of about the same size.
- Simplest version of the graph partitioning problem.

A theoretical result: If \( G \) is a planar graph with \( N \) vertices, then there is a separator \( S \) of size \( \leq \sqrt{N} \) such that \( |T_1| \leq 2N/3 \) and \( |T_2| \leq 2N/3 \).

In other words “Planar graphs have \( O(\sqrt{N}) \) separators”

- Many techniques for finding separators: Spectral, iterative swapping (K-L), multilevel (Metis), BFS, ...