Sparse Direct Methods:
A Tutorial Overview

Michael T. Heath

Sparse Linear Systems

Boundary value problems and implicit methods for time-dependent PDEs yield systems of linear algebraic equations to solve.

Finite difference schemes involving only a few variables each, or localized basis functions in finite element approach, cause linear system to be *sparse*, with relatively few nonzero entries.

Sparsity can be exploited to use much less than $\mathcal{O}(n^2)$ storage and $\mathcal{O}(n^3)$ work required in naive approach to solving system.

Sparse Factorization Methods

Gaussian elimination and Cholesky factorization are applicable to large sparse systems, but care is required to achieve reasonable efficiency in solution time and storage requirements.

Key to efficiency is to store and operate on only nonzero entries of matrix.

Special data structures are required instead of simple 2-D arrays for storing dense matrices.

Band Systems

For 1-D problems, equations and unknowns can usually be ordered so that nonzeros are concentrated in narrow band, which can be stored efficiently in rectangular 2-D array by diagonals.

Bandwidth can often be reduced by reordering rows and columns of matrix.

For problems in two or more dimensions, even narrowest possible band often contains mostly zeros, so 2-D array storage is very wasteful.

General Sparse Data Structures

In general, sparse systems require data structures that store only nonzero entries, along with indices to identify their locations in matrix.

Explicitly storing indices incurs additional storage overhead and makes arithmetic operations on nonzeros less efficient due to indirect addressing to access operands.

Data structure is worthwhile only if matrix is sufficiently sparse, which is often true for very large problems arising from PDEs and many other applications.

Fill

When applying LU or Cholesky factorization to sparse matrix, taking linear combinations of rows or columns to annihilate unwanted nonzero entries can introduce new nonzeros into matrix locations that were initially zero.

Such new nonzeros, called *fill*, must be stored and may eventually be annihilated themselves in order to obtain triangular factors.

Resulting triangular factors can be expected to contain at least as many nonzeros as original matrix and usually a significant amount of fill as well.

Reordering to Limit Fill

Amount of fill is sensitive to order in which rows and columns of matrix are processed, so basic problem in sparse factorization is reordering matrix to limit fill during factorization.

Exact minimization of fill is hard combinatorial problem (NP-complete), but heuristic algorithms such as minimum degree and nested dissection do well at limiting fill for many types of problems.

Example: Laplace Equation

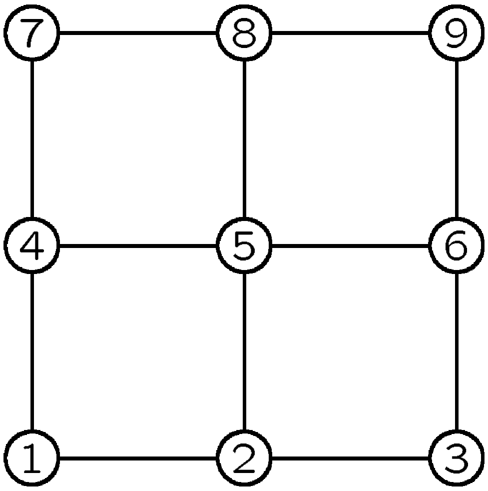
Discretization of Laplace equation on square by second-order finite difference approximation to second derivatives yields system of linear equations whose unknowns correspond to mesh points (nodes) in square grid.

A pair of nodes is connected by an *edge* if both appear in the same equation in this system.

Two nodes are *neighbors* if they are connected by an edge.

Diagonal entries of matrix correspond to nodes in mesh, and nonzero off-diagonal entries correspond to edges in mesh ($a_{ij} \neq 0 \Leftrightarrow$ nodes i and j are neighbors).

Grid and Matrix



$$\begin{bmatrix}
\times & \times & & \times & & & & & \\
\times & \times & \times & & \times & & & & \\
& \times & \times & & & \times & & & \\
\times & & & \times & \times & & \times & & \\
& \times & & \times & \times & \times & & \times & \\
& & \times & & \times & \times & & & \times \\
& & & \times & & & \times & \times & \\
& & & & \times & & \times & \times & \times \\
& & & & & \times & & \times & \times
\end{bmatrix}$$

With nodes numbered row-wise (or column-wise), matrix is block tridiagonal, with each nonzero block either tridiagonal or diagonal.

Matrix is banded but has many zero entries inside band.

Matrix and Factor

$$
\underset{A}{\begin{bmatrix}
\times & \times & & \times & & & & & \\
\times & \times & \times & & \times & & & & \\
& \times & \times & & & \times & & & \\
\times & & & \times & \times & & \times & & \\
& \times & & \times & \times & \times & & \times & \\
& & \times & & \times & \times & & & \times \\
& & & \times & & & \times & \times & \\
& & & & \times & & \times & \times & \times \\
& & & & & \times & & \times & \times
\end{bmatrix}}
\qquad
\underset{L}{\begin{bmatrix}
\times & & & & & & & & \\
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
\times & + & + & \times & & & & & \\
& \times & + & \times & \times & & & & \\
& & \times & + & \times & \times & & & \\
& & & \times & + & + & \times & & \\
& & & & \times & + & \times & \times & \\
& & & & & \times & + & \times & \times
\end{bmatrix}}
$$

Cholesky factorization fills in band almost completely.

Graph Model of Elimination

Each step of factorization process corresponds to elimination of one node from mesh.

Eliminating a node causes its neighboring nodes to become connected to each other.

If any such neighbors were not already connected, then *fill* results (new edges in mesh and new nonzeros in matrix).

Minimum Degree Ordering

Good heuristic for limiting fill is to eliminate first those nodes having fewest neighbors.

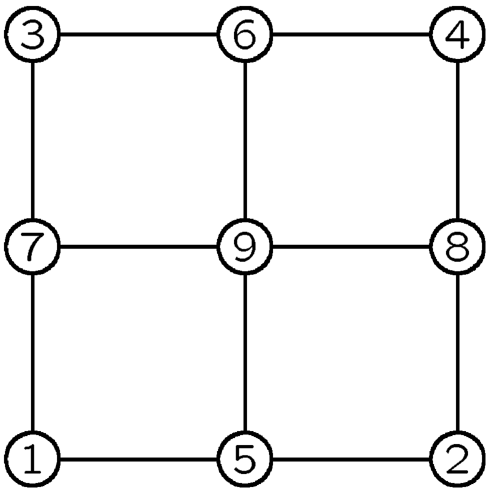
Number of neighbors is called *degree* of node, so heuristic is known as *minimum degree*.

At each step, minimum degree selects for elimination a node of smallest degree, breaking ties arbitrarily.

After node has been eliminated, its neighbors become connected to each other, so degrees of some nodes may change.

Process is then repeated, with a new node of minimum degree eliminated next, and so on until all nodes have been eliminated.

Grid and Matrix with Minimum Degree Ordering



$$\begin{bmatrix}
\times & & & & \times & & \times & & \\
& \times & & & \times & & & \times & \\
& & \times & & & \times & \times & & \\
& & & \times & & \times & & \times & \\
\times & \times & & & \times & & & & \times \\
& & \times & \times & & \times & & & \times \\
\times & & \times & & & & \times & & \times \\
& \times & & \times & & & & \times & \times \\
& & & & \times & \times & \times & \times & \times
\end{bmatrix}$$

Matrix and Factor with Minimum Degree Ordering

$$
\begin{array}{cc}
\begin{bmatrix}
\times & & & & \times & & \times & & \\
& \times & & & \times & & & \times & \\
& & \times & & & \times & \times & & \\
& & & \times & & \times & & \times & \\
\times & \times & & & \times & & & & \times \\
& & \times & \times & & \times & & & \times \\
\times & & \times & & & & \times & & \times \\
& \times & & \times & & & & \times & \times \\
& & & & \times & \times & \times & \times & \times
\end{bmatrix}
&
\begin{bmatrix}
\times & & & & & & & & \\
& \times & & & & & & & \\
& & \times & & & & & & \\
& & & \times & & & & & \\
\times & \times & & & \times & & & & \\
& & \times & \times & & \times & & & \\
\times & & \times & & + & + & \times & & \\
& \times & & \times & + & + & + & \times & \\
& & & & \times & \times & \times & \times & \times
\end{bmatrix}
\\
A & L
\end{array}
$$

Cholesky factor suffers much less fill than with original ordering, and advantage grows with problem size.

Sophisticated versions of minimum degree are among most effective general-purpose orderings known.

Nested Dissection Ordering

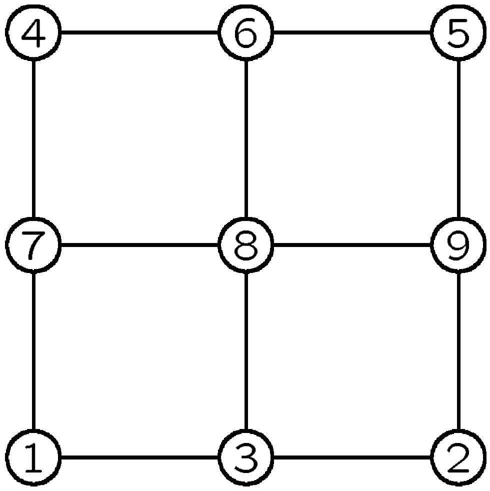
Nested dissection is based on a divide-and-conquer strategy.

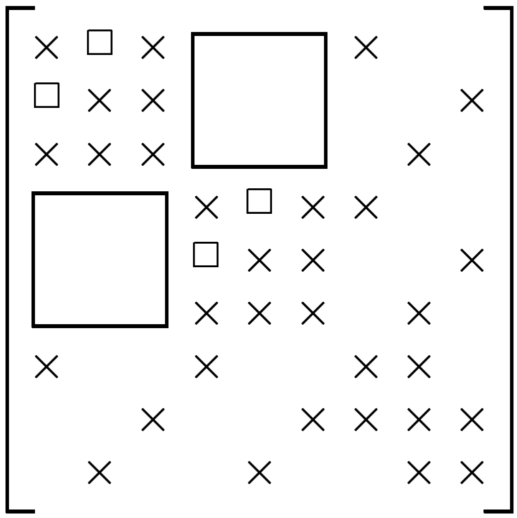
First, a small set of nodes is selected whose removal splits mesh into two pieces of roughly equal size.

No node in either piece is connected to any node in the other, so no fill occurs in either piece due to elimination of any node in the other.

Separator nodes are numbered *last*, then process is repeated recursively on each remaining piece of mesh until all nodes have been numbered.

Grid and Matrix with Nested Dissection Ordering





Dissection induces blocks of zeros in matrix (indicated by squares in diagram) that are automatically preserved during factorization.

Recursive nature of algorithm can be seen in hierarchical block structure of matrix, which would involve many more levels in larger problems.

Matrix and Factor with Nested Dissection Ordering

$$
A = \begin{bmatrix}
\times & \Box & \times & & & & \times & & \\
\Box & \times & \times & & & & & & \times \\
\times & \times & \times & & & & & \times & \\
 & & & \times & \Box & \times & \times & & \\
 & & & \Box & \times & \times & & & \times \\
 & & & \times & \times & \times & & \times & \\
\times & & & \times & & & \times & \times & \\
 & & \times & & & \times & \times & \times & \times \\
 & \times & & & \times & & & \times & \times
\end{bmatrix}
\qquad
L = \begin{bmatrix}
\times & & & & & & & & \\
\Box & \times & & & & & & & \\
\times & \times & \times & & & & & & \\
 & & & \times & & & & & \\
 & & & \Box & \times & & & & \\
 & & & \times & \times & \times & & & \\
\times & & + & \times & & + & \times & & \\
 & & \times & & & \times & \times & \times & \\
 & \times & + & & \times & + & + & \times & \times
\end{bmatrix}
$$

A $\qquad\qquad$ L

Again, Cholesky factor suffers much less fill than with original ordering, and advantage grows with problem size.