Chapter 8
Sparse Matrix Ordering Algorithms

The computational complexity of obtaining optimal reorderings for performing sparse Gaussian elimination justifies the heuristic nature of all practical reordering algorithms. – Erisman et al. (1987).

So far, our focus has been on the theoretical and algorithmic principles involved in sparse Gaussian elimination-based factorizations. To limit the storage and the work involved in the computation of the factors and in their use during the solve phase it is generally necessary to reorder (permute) the matrix before the factorization commences. The complexity of the most critical steps in the factorization is highly dependent on the amount of fill-in, as can be seen from the following observation.

Observation 8.1 *The operations to perform the sparse LU factorization $A = LU$ and the sparse Cholesky factorization $A = LL^T$ are $O(\sum_{j=1}^{n} |col_L\{j\}||row_U\{j\}|)$ and $O(\sum_{j=1}^{n} |col_L\{j\}|^2)$ respectively, where $|row_U\{j\}|$ and $|col_L\{j\}|$ are the number of off-diagonal entries in row j of U and column j of L, respectively.*

The problem of finding a permutation to minimize fill-in is NP complete and thus heuristics are used to determine orderings that limit the amount of fill-in; we refer to these as fill-reducing orderings. Frequently, this is done using the sparsity pattern $\mathcal{S}\{A\}$ alone, although sometimes for non-definite matrices, it is combined with the numerical factorization because additional permutations of A may be needed to make the matrix factorizable. Two main classes of methods that work with $\mathcal{S}\{A\}$ are commonly used.

Local orderings attempt to limit fill-in by repeated local decisions based on $\mathcal{G}(A)$ (or a relevant quotient graph).

Global orderings consider the whole sparsity pattern of A and seek to find a permutation using a divide-and-conquer approach. Such methods are normally used in conjunction with a local fill-reducing ordering, as the latter generally works well for problems that are not really large.

J. Scott, M. Tůma, *Algorithms for Sparse Linear Systems*, Nečas Center Series,
https://doi.org/10.1007/978-3-031-25820-6_8

It is assumed throughout this chapter that A is irreducible. Otherwise, if $\mathcal{S}\{A\}$ is symmetric, the algorithms are applied to each component of $\mathcal{G}(A)$ independently and n is then the number of vertices in the component. If $\mathcal{S}\{A\}$ is nonsymmetric, we assume that A is in block triangular form and the algorithms are used on the graph of each block on the diagonal. We also assume that A has **no rows or columns that are (almost) dense**. If it does, a simple strategy is to remove them before applying the ordering algorithm to the remaining matrix; the variables corresponding to the dense rows and columns can be appended to the end of the computed ordering to give the final ordering.

Historically, ordering the matrix A before using a direct solver to factorize it was generally cheap compared to the numerical factorization cost. However, in the last couple of decades, the development of more sophisticated factorization algorithms and their implementations in parallel on modern architectures has affected this balance so that the ordering can be the most expensive step. If a sequence of matrices having the same sparsity pattern is to be factorized, then the ordering cost and the cost of the symbolic factorization can be amortized over the numerical factorizations. If not, it is important to have available a range of ordering algorithms because using a cheap but less effective algorithm may lead to faster complete solution times compared to using an expensive approach that gives some savings in the memory requirements and operation counts but not enough to offset the ordering cost.

8.1 Local Fill-Reducing Orderings for Symmetric $\mathcal{S}\{A\}$

In the symmetric case, the diagonal entries of A are required to be present in $\mathcal{S}\{A\}$ (thus zeros on the diagonal are included in the sparsity structure). The aim is to limit fill-in in the L factor of an LL^T (or LDL^T) factorization of A. Two greedy heuristics are the minimum degree (MD) criterion and the local minimum fill (MF) criterion.

8.1.1 Minimum Fill-in (MF) Criterion

One way to reduce fill-in is to use a local **minimum fill-in** (MF) criterion that, at each step, selects as the next variable in the ordering one that will introduce the least fill-in in the factor at that step. This is sometimes called the **minimum deficiency** approach. While MF can produce good orderings, its cost is often considered to be prohibitive because it requires the updated sparsity pattern and the fill-in associated with the possible candidates must be determined. The runtime can be reduced using an approximate variant (AMF) but it is not widely implemented in modern sparse direct solvers.

8.1.2 Basic Minimum Degree (MD) Algorithm

The minimum degree (MD) algorithm is the best-known and most widely used greedy heuristic for limiting fill-in. It seeks to find a permutation such that at each step of the factorization the number of entries in the corresponding column of L is minimized. This metric is easier and less expensive to compute compared to that used by the minimum fill-in criterion. If $\mathcal{G}(A)$ is a tree, then the MD algorithm results in no fill-in but, in most real applications, it does not minimize the amount of fill-in exactly.

The MD algorithm can be implemented using $\mathcal{G}(A)$ and it can predict the required factor storage without generating the structure of L. The basic approach is given in Algorithm 8.1. At step k, the number of off-diagonal nonzeros in a row or column of the active submatrix is the **current degree** of the corresponding vertex in the elimination graph $\mathcal{G}^k$. The algorithm selects a vertex of minimum current degree in $\mathcal{G}^k$ and labels it v_k, i.e. next for elimination. The set of vertices adjacent to v_k in $\mathcal{G}(A)$ is $\mathcal{R}each(v_k, \mathcal{V}_k)$, where $\mathcal{V}_k$ is the set of $k-1$ vertices that have already been eliminated. These are the only vertices whose degrees can change at step k. If $u \in \mathcal{R}each(v_k, \mathcal{V}_k)$, $u \neq v_k$, then its updated current degree is $|\mathcal{R}each(u, \mathcal{V}_{k+1})|$, where $\mathcal{V}_{k+1} = \mathcal{V}_k \cup v_k$.

At Step 4 of Algorithm 8.1, a tie-breaking strategy is needed when there is more than one vertex of current minimum degree. A straightforward strategy is to select the vertex that lies first in the original order. For the example in Figure 8.1, vertices 2, 3, and 6 are initially all of degree 2 and could be selected for elimination; as the lowest-numbered vertex, 2 is chosen. After it has been eliminated, vertices 3, 5, and 6 have current degree 2 and so vertex 3 is next. As all the remaining vertices have current degree 2, vertex 1 is eliminated, followed by 4, 5, and 6. It is possible to construct artificial matrices showing that some systematic tie-breaking choices can lead to a large amount of fill-in but such behaviour is not typical.

ALGORITHM 8.1 Basic minimum degree (MD) algorithm
Input: Graph $\mathcal{G}$ of a symmetrically structured matrix.
Output: A permutation vector p that defines a new labelling of the vertices of $\mathcal{G}$.

1: Set $\mathcal{G}^1 = \mathcal{G}$ and compute the degree $deg_{\mathcal{G}^1}(u)$ of all $u \in \mathcal{V}(\mathcal{G}^1)$
2: **for** $k = 1 : n-1$ **do**
3: Compute $mdeg = \min\{deg_{\mathcal{G}^k}(u) \mid u \in \mathcal{V}(\mathcal{G}^k)\}$ ▷ $mdeg$ is the current minimum degree
4: Choose $v_k \in \mathcal{V}(\mathcal{G}^k)$ such that $deg_{\mathcal{G}^k}(v_k) = mdeg$
5: $p(k) = v_k$ ▷ v_k is the next vertex in the elimination order
6: Construct $\mathcal{G}^{k+1}$ and update the current degrees of its vertices
7: **end for**
8: $p(n) = v_n$ where v_n is the only vertex in $\mathcal{G}^n$

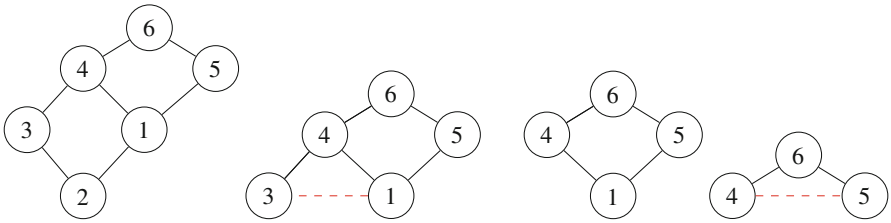


Figure 8.1 An illustration of three steps of the MD algorithm. The original graph $\mathcal{G}$ and the elimination graphs $\mathcal{G}^2$, $\mathcal{G}^3$ and $\mathcal{G}^4$ that result from eliminating vertex 2, then vertex 3 and then vertex 1 are shown red dashed lines denote fill edges.

The construction of each elimination graph $\mathcal{G}^{k+1}$ is central to the implementation of the MD algorithm. Because eliminating a vertex potentially creates fill-in, an efficient representation of the resulting elimination graph that accommodates this (either implicitly or explicitly) is needed. Moreover, recalculating the current degrees is time consuming. Consequently, various approaches have been developed to enhance performance; these are discussed in the following subsections.

8.1.3 Use of Indistinguishable Vertices

In Section 3.5.1, we introduced indistinguishable vertices and supervariables. The importance of exploiting these in MD algorithms is emphasized by the next two results. Here $\mathcal{G}_v$ denotes the elimination graph obtained from $\mathcal{G}$ when vertex $v \in \mathcal{V}(\mathcal{G})$ is eliminated.

Theorem 8.1 (George & Liu 1980b, 1989) *Let u and w be indistinguishable vertices in $\mathcal{G}$. If $v \in \mathcal{V}(\mathcal{G})$ with $v \neq u, w$, then u and w are indistinguishable in $\mathcal{G}_v$.*

Proof Two cases must be considered. First, let $u \notin adj_{\mathcal{G}}\{v\}$. Then $w \notin adj_{\mathcal{G}}\{v\}$ and if v is eliminated, the adjacency sets of u and w are unchanged and these vertices remain indistinguishable in the resulting elimination graph $\mathcal{G}_v$. Second, let $u, w \in adj_{\mathcal{G}}\{v\}$. When v is eliminated, because u and w are indistinguishable in $\mathcal{G}$, their adjacency sets in $\mathcal{G}_v$ will be modified in the same way, by adding the entries of $adj_{\mathcal{G}}\{v\}$ that are not already in $adj_{\mathcal{G}}\{u\}$ and $adj_{\mathcal{G}}\{w\}$. Consequently, u and w are indistinguishable in $\mathcal{G}_v$. □

Figure 8.2 demonstrates the two cases in the proof of Theorem 8.1. Here, u and w are indistinguishable vertices in $\mathcal{G}$. Setting $v \equiv v'$ illustrates $u \notin adj_{\mathcal{G}}\{v\}$. If v' is eliminated, then the adjacency sets of u and w are clearly unchanged. Setting $v \equiv v''$ illustrates $u, w \in adj_{\mathcal{G}}\{v\}$. In this case, if v'' is eliminated, then vertices s and t are added to both $adj_{\mathcal{G}}\{u\}$ and $adj_{\mathcal{G}}\{w\}$.

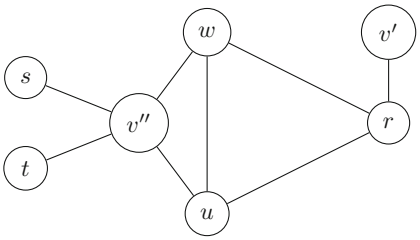


Figure 8.2 An example to illustrate Theorem 8.1. u and w are indistinguishable vertices in $\mathcal{G}$; $adj_{\mathcal{G}}\{u\} = \{r, w, v''\}$ and $adj_{\mathcal{G}}\{w\} = \{r, u, v''\}$.

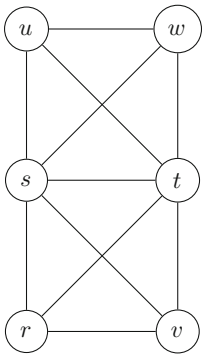


Figure 8.3 An illustration of Theorem 8.2. Vertices u and w are of minimum degree (with degree $mdeg = 3$) and are indistinguishable in $\mathcal{G}$. After elimination of w, the current degree of u is $mdeg - 1$ and the current degree of each of the other vertices is at most $mdeg - 1$. Therefore, u is of current minimum degree in $\mathcal{G}_w$. Note that vertices r and v are also of minimum degree and indistinguishable in $\mathcal{G}$; they are not neighbours of w and their degrees do not change when w is eliminated.

Theorem 8.2 (George & Liu 1980b, 1989) *Let u and w be indistinguishable vertices in $\mathcal{G}$. If w is of minimum degree in $\mathcal{G}$, then u is of minimum degree in $\mathcal{G}_w$.*

Proof Let $deg_{\mathcal{G}}(w) = mdeg$. Then $deg_{\mathcal{G}}(u) = mdeg$. Indistinguishable vertices are always neighbours. Eliminating w gives $deg_{\mathcal{G}_w}(u) = mdeg - 1$ because w is removed from the adjacency set of u and there is no neighbour of u in $\mathcal{G}_w$ that was not its neighbour in $\mathcal{G}$. If $x \neq w$ and $x \in adj_{\mathcal{G}}\{u\}$, then the number of neighbours of x in $\mathcal{G}_w$ is at least $mdeg - 1$. Otherwise, if $x \notin adj_{\mathcal{G}}\{u\}$, then its adjacency set in $\mathcal{G}_w$ is the same as in $\mathcal{G}$ and is of the size at least $mdeg$. The result follows. □

Theorem 8.2 is illustrated in Figure 8.3.

Theorems 8.1 and 8.2 can be extended to more than two indistinguishable vertices, which allows indistinguishable vertices to be selected one after another in the MD ordering. This is referred to as **mass elimination**. Treating indistinguishable vertices as a single supervariable cuts the number of vertices and edges in the elimination graphs, which reduces the work needed for degree updates.

In the basic MD algorithm, the current degree of a vertex is the number of adjacent vertices in the current elimination graph. The **external degree** of a vertex is the number of vertices adjacent to it that are not indistinguishable from it. The motivation comes from the underlying reason for the success of the minimum degree ordering in terms of fill reduction. Eliminating a vertex of minimum degree implies the formation of the smallest possible clique resulting from the elimination. If mass elimination is used, then the size of the resulting clique is equal to the external degree of the vertices eliminated by the mass elimination step. Using the external degree can speed up the time for computing the ordering and give worthwhile savings in the number of entries in the factors.

8.1.4 Degree Outmatching

A concept that is closely related to that of indistinguishable vertices is **degree outmatching**. This avoids computing the degrees of vertices that are known not to be of current minimum degree. Vertex w is said to be outmatched by vertex u if

$$adj_{\mathcal{G}}\{u\} \cup \{u\} \subseteq adj_{\mathcal{G}}\{w\} \cup \{w\}.$$

It follows that $deg_{\mathcal{G}}(u) \leq deg_{\mathcal{G}}(w)$. A simple example is given in Figure 8.4. Importantly, degree outmatching is preserved when vertex $v \in \mathcal{G}$ of minimum degree is eliminated, as stated in the following result.

Theorem 8.3 (George & Liu 1980b, 1989) *In the graph $\mathcal{G}$ let vertex w be outmatched by vertex u and vertex v ($v \neq u, w$) be of minimum degree. Then w is outmatched in $\mathcal{G}_v$ by u.*

Proof Three cases must be considered. First, if $u \notin adj_{\mathcal{G}}\{v\}$ and $w \notin adj_{\mathcal{G}}\{v\}$, then the adjacency sets of u and w in $\mathcal{G}_v$ are the same as in $\mathcal{G}$. Second, if v is a neighbour of both u and w in $\mathcal{G}$, then any neighbours of v that were not neighbours of u and

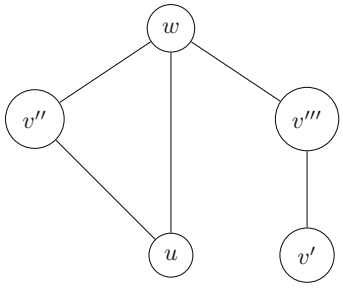


Figure 8.4 An example $\mathcal{G}$ in which vertex w is outmatched by vertex u. v' is not a neighbour of u or w; vertex v'' is a neighbour of both u and w; v''' is a neighbour of w but not of u.

w are added to their adjacency sets in $\mathcal{G}_v$. Third, if $u \notin adj_{\mathcal{G}}\{v\}$ and $w \in adj_{\mathcal{G}}\{v\}$, then the adjacency set of u in $\mathcal{G}_v$ is the same as in $\mathcal{G}$ but any neighbours of v that were not neighbours of w are added to the adjacency set of w in $\mathcal{G}_v$. In all three cases, w is still outmatched by u in $\mathcal{G}_v$. □

The three possible cases for v in the proof of Theorem 8.3 are illustrated in Figure 8.4 by setting $v \equiv v'$, v'' and v''', respectively. An important consequence of Theorem 8.3 is that if w is outmatched by u, then it is not necessary to consider w as a candidate for elimination and all updates to the data structures related to w can be postponed until u has been eliminated.

8.1.5 Cliques and Quotient Graphs

From Parter's rule, if vertex v is selected at step k, then the elimination matrix that corresponds to $\mathcal{G}^{k+1}$ contains a dense submatrix of size equal to the number of off-diagonal entries in row and column v in the matrix that corresponds to $\mathcal{G}^k$. For large matrices, creating and explicitly storing the edges in the sequence of elimination graphs is impractical and a more compact and efficient representation is needed. Each elimination graph can be interpreted as a collection of cliques, including the original graph $\mathcal{G}$, which can be regarded as having $|\mathcal{E}|$ cliques, each consisting of two vertices (or, equivalently, an edge). This gives a conceptually different view of the elimination process and provides a compact scheme to represent the elimination graphs. The advantage in terms of storage is based on the following.

Let $\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_q\}$ be the set of cliques for the current graph and let v be a vertex of current minimum degree that is selected for elimination. Let $\{\mathcal{V}_{s_1}, \mathcal{V}_{s_2}, \ldots, \mathcal{V}_{s_t}\}$ be the subset of cliques to which v belongs. Two steps are then required.

1. Remove the cliques $\{\mathcal{V}_{s_1}, \mathcal{V}_{s_2}, \ldots, \mathcal{V}_{s_t}\}$ from $\{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_q\}$.
2. Add the new clique $\mathcal{V}_v = \{\mathcal{V}_{s_1} \cup \ldots \cup \mathcal{V}_{s_t}\} \setminus \{v\}$ into the set of cliques.

Hence

$$deg_{\mathcal{G}}(v) = |\mathcal{V}_v| < \sum_{i=1}^{t} |\mathcal{V}_{s_i}|,$$

and because $\{\mathcal{V}_{s_1}, \mathcal{V}_{s_2}, \ldots, \mathcal{V}_{s_t}\}$ can now be discarded, the storage required for the representation of the sequence of elimination graphs never exceeds that needed for $\mathcal{G}(A)$. The storage to compute an MD ordering is therefore known beforehand in spite of the rather dynamic nature of the elimination process. The index of the eliminated vertex can be used as the index of the new clique. This is called an **element** or **enode** (the terminology comes from finite-element methods), to distinguish it from an uneliminated vertex, which is termed an **snode**.

A sequence of special quotient graphs $\mathcal{G}^{[1]} = \mathcal{G}(A), \mathcal{G}^{[2]}, \ldots, \mathcal{G}^{[n]}$ with the two types of vertices is generated in place of the elimination graphs. Each $\mathcal{G}^{[k]}$ has n vertices that satisfy

$$\mathcal{V}(\mathcal{G}) = \mathcal{V}_{snodes} \cup \mathcal{V}_{enodes}, \qquad \mathcal{V}_{snodes} \cap \mathcal{V}_{enodes} = \emptyset,$$

where $\mathcal{V}_{snodes}$ and $\mathcal{V}_{enodes}$ are the sets of snodes and enodes, respectively. When v is eliminated, any enodes adjacent to it are no longer required to represent the sparsity pattern of the corresponding active submatrix and so they can be removed. This is called **element absorption**.

Working with these graphs can be demonstrated by considering the computation of the vertex degrees. To compute the degree of an uneliminated vertex, the set of neighbouring snodes is counted. Then, if a neighbour of one of these snodes is an enode, its neighbours are also counted (avoiding double counting). More formally, if $v \in \mathcal{V}_{snodes}$, then the adjacency set of v is the union of its neighbours in $\mathcal{V}_{snodes}$ and the vertices reachable from v via its neighbours in $\mathcal{V}_{enodes}$. In this way, vertex degrees are computed by considering fill-paths, avoiding storing the fill-in entries explicitly. This reduces memory requirements and contributes to the computational efficiency, which can be further improved by amalgamating sets of indistinguishable enodes and snodes.

The sequences of elimination graphs and quotient graphs are illustrated in Figure 8.5. The top line shows $\mathcal{G}$ together with $\mathcal{G}^2$ and $\mathcal{G}^3$ after the elimination of vertices 1 and 2, respectively. When vertex 1 is eliminated, a new edge is added to make its neighbours into a clique. The elimination of vertex 2 creates no additional fill and the graph $\mathcal{G}^3$ with three nodes represents the sparsity structure of the corresponding active submatrix $A^{(3)}$. The bottom line shows the corresponding quotient graphs. After the first elimination, vertex 1 is an enode and the fill edge is represented implicitly. After the second elimination, the enodes 1 and 2 can be amalgamated and so too can the snodes 3 and 4 because they are indistinguishable.

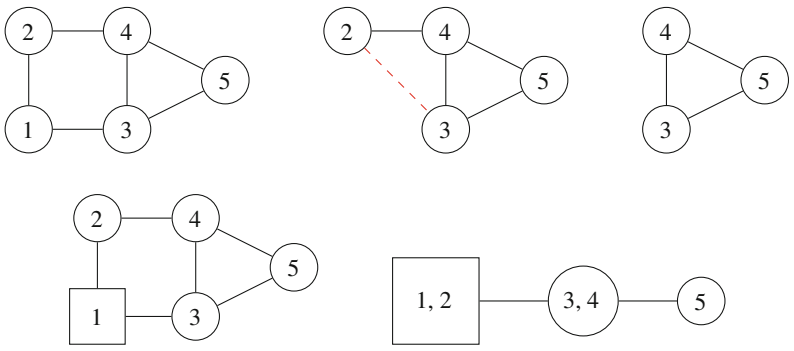


Figure 8.5 The top line shows $\mathcal{G} = \mathcal{G}^1$, $\mathcal{G}^2$ and $\mathcal{G}^3$. The red dashed line denotes a fill edge. The bottom line shows the quotient graphs $\mathcal{G}^{[2]}$ and $\mathcal{G}^{[3]}$ after the first and second elimination steps. A circle represents a vertex in $\mathcal{G}$ (an snode), while a square represents an enode.

ALGORITHM 8.2 Basic multiple minimum degree (MMD) algorithm
Input: Graph $\mathcal{G}$ of a symmetrically structured matrix.
Output: A permutation vector p that defines a new labelling of the vertices of $\mathcal{G}$.

Set $k = 1$, $\mathcal{G}^1 = \mathcal{G}$ and compute the degree $deg_{\mathcal{G}^1}(u)$ of all $u \in \mathcal{V}(\mathcal{G}^1)$
while $k \leq n$ **do**
    Compute $mdeg = \min\{deg_{\mathcal{G}^k}(u) \mid u \in \mathcal{V}(\mathcal{G}^k)\}$
    Find all mutually non-adjacent $v_j \in \mathcal{V}(\mathcal{G}^k)$, $j = 1, \ldots, t$ with $deg_{\mathcal{G}^k}(v_j) = mdeg$
    for $j = 1 : t$ **do**
        $p(k) = v_j$ ▷ Vertex v_j is the next vertex in the elimination order
        $k = k + 1$
    end for
    if $k < n$ **then**
        Construct $\mathcal{G}^{k+1}$ and update the current degrees of its vertices
    end if
end while

8.1.6 *Multiple Minimum Degree (MMD) Algorithm*

The multiple minimum degree (MMD) algorithm aims to improve efficiency by processing several independent vertices that are each of minimum current degree together in the same step, before the degree updates are performed. The basic approach is outlined as Algorithm 8.2. At each outer loop, $t \geq 1$ denotes the number of vertices of minimum current degree that are mutually non-adjacent and so can be put into the elimination order one after another. An example in which the four corner vertices have the same minimum degree is depicted in Figure 8.6. Here, on the first step, $mdeg = 2$ and $t = 4$. Note that the MMD strategy is complementary to the mass elimination approach in which the set S of indistinguishable vertices that can be eliminated one after another is fully interconnected and all vertices of S have the same set of neighbours outside S.

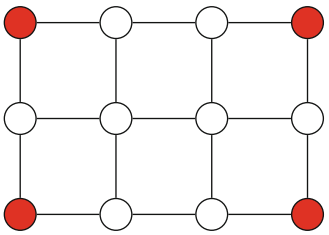


Figure 8.6 The red (corner) vertices of $\mathcal{G}$ are each of degree 2 and are ordered consecutively during the first step of Algorithm 8.2.

The complexity of the MD and MMD algorithms is $O(nz(A)n^2)$ but because for MMD the outer loop of the algorithm update is performed fewer times, it can be significantly faster than MD. MMD orderings can also lead to less fill-in, possibly a consequence of introducing some kind of regularity into the ordering sequence.

8.1.7 Approximate Minimum Degree (AMD) Algorithm

The idea behind the widely used **approximate minimum degree** (AMD) algorithm is to inexpensively compute an upper bound on a vertex degree in place of the degree, and to use this bound as an approximation to the external degree when selecting vertices within the MD algorithm. Even though vertex degrees are not determined exactly, the quality of the orderings obtained using the AMD algorithm are competitive with those computed using the MD algorithm and can surpass them. The complexity of AMD is $O(nz(A)n)$ and, in practice, its runtime is typically significantly less than that of the MD and MMD approaches.

8.2 Minimizing the Bandwidth and Profile

An alternative way of reducing the fill-in locally is to add another criterion to the relabelling of the vertices, such as restricting the nonzeros of the permuted matrix to specific positions. The most popular approach is to force them to lie close to the main diagonal. If Gaussian elimination is applied without further permutations, then all fill-in takes place between the first entry of a row and the diagonal or between the first entry of a column and the diagonal. It is therefore sufficient to store all the entries in the lower triangular part from the first entry in each row to the diagonal and all the entries in the upper triangular part from the first entry in each column to the diagonal. This allows straightforward implementations of Gaussian elimination that employ static data structures. Here we again consider symmetric and, for simplicity, we assume that $\mathcal{G}(A)$ is connected; generalizations of the terminology and ideas to nonsymmetric matrices are possible.

8.2.1 The Band and Envelope of a Matrix

To characterize the positions within $\mathcal{S}\{A\}$ that are close to the main diagonal, we denote the leftmost entries in the lower triangular part of A using the mapping η_i as follows:

$$\eta_i(A) = \min\{j \mid 1 \le j \le i \text{ with } a_{ij} \neq 0\}, \quad 1 \le i \le n, \tag{8.1}$$

that is, $\eta_i(A)$ is the column index of the first entry in the i-th row of A.

Define

$$\beta_i(A) = i - \eta_i(A), \quad 1 \le i \le n.$$

The **semibandwidth** of A is

$$\max\{\beta_i(A) |\ 1 \le i \le n\},$$

and the **bandwidth** is

$$\beta(A) = 2 * \max\{\beta_i(A) \mid\ 1 \le i \le n\} + 1.$$

The **band** of A is the following set of index pairs in A

$$band(A) = \{(i, j) \mid\ 0 < i - j \le \beta(A)\}.$$

The **envelope** is the set of index pairs that lie between the first entry in each row and the diagonal

$$env(A) = \{(i, j) \mid\ 0 < i - j \le \beta_i(A)\}.$$

Note that the band and envelope of a sparse symmetrically structured matrix A include only entries of the strict lower triangular part of A. The envelope is easily visualized: picture the lower triangular part of A, and remove the diagonal and the leading zero entries in each row. The remaining entries (whether nonzero or zero) comprise the envelope of A. The **profile** of A is defined to be the number of entries in the envelope (the envelope size) plus n.[1] An illustrative example is given in Figure 8.7. Here $\eta_1(A) = 1$, $\beta_1(A) = 0$, $\eta_2(A) = 1$, $\beta_2(A) = 1$, $\eta_3(A) = 2$, $\beta_3(A) = 1$, and so on.

Figure 8.7 Illustration of the band and envelope of a matrix A whose sparsity pattern is on the left. In the centre, the positions of $band(A)$ are circled and on the right, the positions of $env(A)$ are circled. The bandwidth is 5 and the envelope size and the profile are 7 and 14, respectively.

[1] Sometimes in the literature the profile is defined to be the envelope size.

The next result shows that the static data structures determined for A are sufficient for its Cholesky factors and by permuting A to minimize its band or profile, the fill-in is also approximately minimized.

Theorem 8.4 (Liu & Sherman 1976; George & Liu 1981) *If L is the Cholesky factor of A, then*

$$env(A) = env(L).$$

Proof The proof uses mathematical induction on the principal leading submatrices of A of order k. The result is clearly true for $k = 1$ and $k = 2$. Assume it holds for $2 \leq k < n$ and consider the block factorization

$$\begin{pmatrix} A_{1:k,1:k} & u_{1:k} \\ u_{1:k}^T & \alpha \end{pmatrix} = \begin{pmatrix} L_{1:k,1:k} & 0 \\ v_{1:k}^T & \beta \end{pmatrix} \begin{pmatrix} L_{1:k,1:k}^T & v_{1:k} \\ 0 & \beta \end{pmatrix},$$

where α and β are scalars. Equating the left and right sides, $L_{1:k,1:k} v_{1:k} = u_{1:k}$. Because $u_j = 0$ for $j < \eta_{k+1}(A)$ and $u_{\eta_{k+1}} \neq 0$, it follows that $v_j = 0$ for $j < \eta_{k+1}(A)$ and $v_{\eta_{k+1}} \neq 0$. This proves the induction step. □

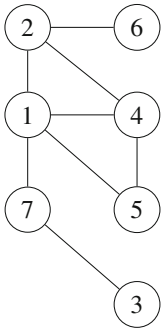
A straightforward corollary of Theorem 8.4 is that $band(A) = band(L)$.

8.2.2 Level-Based Orderings

Finding a permutation P to minimize the band or profile of PAP^T is combinatorially hard and again heuristics are used to efficiently find an acceptable P. The popular Cuthill McKee (CM) approach chooses a suitable starting vertex s and labels it 1. Then, for $i = 1, 2, \ldots, n-1$, all vertices adjacent to vertex i that are still unlabelled are labelled successively in order of increasing degree, as described in Algorithm 8.3. A very important variation is the Reverse Cuthill McKee (RCM) algorithm, which incorporates a final step in which the CM ordering is reversed. The CM- and RCM-permuted matrices have the same bandwidth but the latter can decrease the envelope, as demonstrated in Figure 8.8.

The importance of the CM and RCM orderings is expressed in the following theorem. The full envelope of the Cholesky factor of the permuted matrix implies cache efficiency when performing the triangular solves once the factorization is complete.

Theorem 8.5 (Liu & Sherman 1976; George & Liu 1981) *Let A be symmetrically structured and irreducible. If P corresponds to the CM labelling obtained from Algorithm 8.3 and L is the Cholesky factor of $P^T AP$, then $env(L)$ is full, that is, all entries of the envelope are nonzero.*



$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
* & * & & * & * & & * \\
* & * & & * & & * & \\
 & & * & & & & * \\
* & * & & * & * & & \\
* & & & * & * & & \\
 & * & & & & * & \\
* & & * & & & & *
\end{pmatrix}
\end{array}
,
\quad
\begin{array}{c} \\ 3 \\ 7 \\ 1 \\ 5 \\ 2 \\ 4 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} 3 & 7 & 1 & 5 & 2 & 4 & 6 \end{array} \\
\begin{pmatrix}
* & * & & & & & \\
* & * & * & & & & \\
 & * & * & * & * & * & \\
 & & * & * & & * & \\
 & & * & & * & * & * \\
 & & * & * & * & * & \\
 & & & & * & & *
\end{pmatrix}
\end{array}
,
\quad
\begin{array}{c} \\ 6 \\ 4 \\ 2 \\ 5 \\ 1 \\ 7 \\ 3 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} 6 & 4 & 2 & 5 & 1 & 7 & 3 \end{array} \\
\begin{pmatrix}
* & & * & & & & \\
 & * & * & * & * & & \\
* & * & * & & * & & \\
 & * & & * & * & & \\
 & * & * & * & * & * & \\
 & & & & * & * & * \\
 & & & & & * & *
\end{pmatrix}
\end{array}
,
$$

Figure 8.8 An example to illustrate Algorithm 8.3. The starting vertex is $s = 3$; it has degree 1. The graph $\mathcal{G}(A)$ is given and the sparsity patterns of A (left), A symmetrically permuted by the CM algorithm (centre) and A symmetrically permuted by the RCM algorithm (right). The profiles of these matrices are 25, 17, and 16, respectively.

A crucial difference between profile reduction ordering algorithms and minimum degree strategies is that the former is based solely on $\mathcal{G}$: the costly construction of quotient graphs is not needed. However, unless the profile after reordering is very small, there can be significantly more fill-in in the factor.

Key to the success of Algorithm 8.3 is the choice of the starting vertex s: the quality of the ordering is highly dependent on s. A good candidate is a vertex for which the maximum distance between it and some other vertex in $\mathcal{G}$ is large. Formally, the **eccentricity** $\epsilon(u)$ of the vertex u in the connected undirected graph $\mathcal{G}$ is defined to be

$$\epsilon(u) = \max\{d(u, v) \mid v \in \mathcal{V}\},$$

where $d(u, v)$ is the distance between the vertices u and v (the length of the shortest path between these vertices). The maximum eccentricity taken over all the vertices is the **diameter** of $\mathcal{G}$ (that is, the maximum distance between any pair of vertices). The endpoints of a diameter (also termed **peripheral vertices**) provide good starting vertices. The complexity of finding a diameter is $O(n^3)$ because the shortest paths amongst all the vertices have to be checked. Thus, a pseudo-diameter defined by any pair of vertices for which $d(u, v)$ is close to the diameter is used instead. The vertices defining a pseudo-diameter are **pseudo-peripheral** vertices.

ALGORITHM 8.3 CM and RCM algorithms for band and profile reduction
Input: Graph $\mathcal{G}$ of a symmetrically structured irreducible matrix and a starting vertex s.
Output: Permutation vectors p_{cm} and p_{rcm} that define new labellings of the vertices of $\mathcal{G}(A)$.

1: $label(1:n) = false$
2: Compute $adj_{\mathcal{G}}\{u\}$ and $deg_{\mathcal{G}}(u)$ for all $u \in \mathcal{V}(\mathcal{G})$
3: $k = 1,\ v_1 = s,\ p_{cm}(1) = v_1,\ label(v_1) = true$
4: **for** $i = 1 : n-1$ **do**
5: **for** $w \in adj_{\mathcal{G}}\{v_i\}$ with $label(w) = false$ in order of increasing degree **do**
6: $k = k+1,\ v_k = w,\ p_{cm}(k) = v_k,\ label(v_k) = true$
7: **end for**
8: **end for**
9: For the RCM ordering, $p_{rcm}(i) = p_{cm}(n-i+1), i = 1, 2, \ldots, n$.

A heuristic algorithm is used to find pseudo-peripheral vertices. A commonly used approach is based on level sets. A level structure rooted at a vertex r is defined as the partitioning of $\mathcal{V}$ into disjoint **levels** $\mathcal{L}_1(r), \mathcal{L}_2(r), \ldots, \mathcal{L}_h(r)$ such that

(i) $\mathcal{L}_1(r) = \{r\}$ and
(ii) for $1 < i \leq h$, $\mathcal{L}_i(r)$ is the set of all vertices that are adjacent to vertices in $\mathcal{L}_{i-1}(r)$ but are not in $\mathcal{L}_1(r), \mathcal{L}_2(r), \ldots, \mathcal{L}_{i-1}(r)$.

The level structure rooted at r may be expressed as the set $\mathcal{L}(r) = \{\mathcal{L}_1(r), \mathcal{L}_2(r), \ldots, \mathcal{L}_h(r)\}$, where h is the total number of levels and is termed the **depth**. The level sets can be found using a breadth-first search that starts at the root r. The Gibbs-Poole-Stockmeyer (GPS) algorithm presented as Algorithm 8.4 can be used to finding pseudo-peripheral vertices, one of which may then be used as a starting vertex for the CM and RCM algorithms. Here the root vertex r is normally taken to be an arbitrary vertex of minimum degree. $\mathcal{L}(r)$ is constructed and then the level structures rooted at each of the vertices in the last level set $\mathcal{L}_h(r)$. If, for some $w \in \mathcal{L}_h(r)$, the depth of $\mathcal{L}_w$ exceeds that of $\mathcal{L}(r)$, w replaces r as the root vertex, and the procedure is repeated. If no such vertex is found, r is chosen as a pseudo-peripheral vertex.

A simple example is given in Figure 8.9. Starting with $r = 2$, after two passes through the while loop, the GPS algorithm returns $s = 8$ and $t = 1$ as pseudo-peripheral vertices.

To obtain an efficient implementation of the GPS algorithm, it is necessary to limit the number of level set structures that are fully constructed. For example, "short circuiting" can be incorporated in which wide level structures are rejected as soon as they are detected (wide levels will not lead to a deep level structure which is

ALGORITHM 8.4 Basic GPS algorithm to find a pair of pseudo-peripheral vertices
Input: Graph $\mathcal{G}$ of a symmetrically structured irreducible matrix and a root vertex r.
Output: Pseudo-peripheral vertices s, t.

1: Construct $\mathcal{L}(r)$ and set $flag = false$
2: **while** $flag = false$ **do**
3: $flag = true$
4: **for** $i = 1 : |\mathcal{L}(r)|$ **do**
5: $w_i \in \mathcal{L}(r)$ ▷ Select vertex w_i from last level set
6: **if** $flag = true$ **then**
7: Construct $\mathcal{L}(w_i)$
8: **if** $depth(\mathcal{L}(w_i)) > depth(\mathcal{L}(r))$ **then**
9: $flag = false$ ▷ Flag that w_i will be used as new initial vertex
10: **end if**
11: **end if**
12: **end for**
13: **if** $flag = true$ **then**
14: $s = r$ and $t = w_i$ ▷ s is chosen; while loop will terminate algorithm
15: **else**
16: $r = w_i$
17: **end if**
18: **end while**

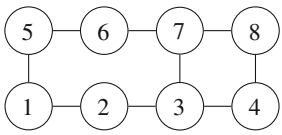


Figure 8.9 An example to illustrate Algorithm 8.4 for finding pseudo-peripheral vertices. With root vertex $r = 2$, the first level set structure is $\mathcal{L}(2) = \{\{2\}, \{1, 3\}, \{4, 5, 7\}, \{6, 8\}\}$. Setting $r = 8$ at Step 16, the second level set structure is $\mathcal{L}(8) = \{\{8\}, \{4, 7\}, \{3, 6\}, \{2, 5\}, \{1\}\}$ and the algorithm terminates with $s = 8$ and $t = 1$.

needed for a narrow band). Furthermore, to reduce the number of vertices in the last level set $\mathcal{L}_h(r)$ for which it is necessary to generate the rooted level structures, a "shrinking" strategy can be used. This typically involves considering the degrees of the vertices in $\mathcal{L}_h(r)$ (for example, only those of smallest degree will be tried). Such modifications can lead to significant time savings while still returning a good starting vertex for the CM and RCM algorithms. As with the MD algorithm, tie-breaking rules must be built into any implementation.

8.2.3 Spectral Orderings

Spectral methods offer an alternative approach that does not use level structures. The spectral algorithm associates a positive semidefinite Laplacian matrix L_p with the symmetric matrix A as follows:

$$(L_p)_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } a_{ij} \neq 0, \\ deg_{\mathcal{G}}(i) & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

An eigenvector corresponding to the smallest positive eigenvalue of the Laplacian matrix is called a **Fiedler vector.** If $\mathcal{G}$ is connected, L_p is irreducible and the second smallest eigenvalue is positive. The vertices of $\mathcal{G}$ are ordered by sorting the entries of the Fiedler vector into monotonic order. Applying the permutation symmetrically to A yields the spectral ordering.

The use of the Fiedler vector for reordering A comes from considering the matrix envelope. The size of the envelope can be written as

$$|env(A)| = \sum_{i=1}^{n} \beta_i = \sum_{i=1}^{n} \max_{\substack{k<i \\ (k,i)\in\mathcal{G}}} (i-k).$$

Observation 8.1 implies that the asymptotic upper bound on the operation count for the factorization based on $env(A)$ is

$$work_{env} = \sum_{i=1}^{n} \beta_i^2 = \sum_{i=1}^{n} \max_{\substack{k<i \\ (k,i)\in\mathcal{G}}} (i-k)^2.$$

Ordering the vertices using the Fiedler vector is closely related to minimizing $weight_{env}$ over all possible vertex reorderings, where

$$weight_{env} = \sum_{i=1}^{n} \sum_{\substack{k<i \\ (k,i)\in\mathcal{G}}} (i-k)^2.$$

Thus, while minimizing the profile and envelope is related to the infinity norm, minimizing $weight_{env}$ is related to the Euclidean norm of the distance between graph vertices.

Although computing the Fiedler vector can be computationally expensive it does have the advantage of easy vectorization and parallelization and the resulting ordering can give small profiles and low operation counts.

8.3 Local fill-reducing orderings for nonsymmetric $\mathcal{S}\{A\}$

If $\mathcal{S}\{A\}$ is nonsymmetric, then an often-used strategy is to apply the minimum degree algorithm (or one of its variants) or a band or profile-reducing ordering to the undirected graph $\mathcal{G}(A+A^T)$. This can work well if the symmetry index $s(A)$ is close to 1. But if A is highly nonsymmetric (typically, for values of $s(A)$ less than 0.5, A is considered to be highly nonsymmetric), then a different approach is required. **Markowitz pivoting** generalizes the MD algorithm by choosing the pivot entry based on vertex degrees computed directly from the nonsymmetric $\mathcal{S}\{A\}$; the result is a nonsymmetric permutation. It can be described using a sequence of bipartite graphs of the active submatrices but here we use a matrix-based description that permutes A on-the-fly. Note that Markowitz pivoting is generally incorporated into the numerical factorization phase of an LU solver, rather than being used to derive an initial reordering of A.

At step k of the LU factorization, consider the $(n-k+1) \times (n-k+1)$ active submatrix, that is, the Schur complement $S^{(k)}$ given by (3.2). Let $nz(row_i)$ and $nz(col_j)$ denote the number of entries in row i and column j of $S^{(k)}$ $(1 \le i, j \le n-k+1)$. Markowitz pivoting selects as the k-th pivot the entry of $S^{(k)}$ that minimizes the **Markowitz count** given by the product

$$(nz(row_i)-1)(nz(col_j)-1).$$

This strategy is summarized in Algorithm 8.5 and illustrated in Figure 8.10. Here the first pivot is a_{24} with Markowitz count 1; it does not cause fill-in. The second pivot has Markowitz count 2 in $S^{(2)}$; it results in one filled entry. Note that the interchanges of rows and columns that are potentially performed at each of the first $n-1$ steps of the factorization give the row and column permutation matrices on the output of Algorithm 8.5. Implementation of the algorithm requires access to the rows and the columns of the matrix.

ALGORITHM 8.5 Markowitz pivoting

Input: Matrix A with a nonsymmetric sparsity pattern.

Output: $A' = PAQ$, where P and Q are permutation matrices chosen to limit fill in.

1: Set $S^{(1)} = A$ and $A' = A$
2: **for** $k = 1 : n-1$ **do**
3: Compute $nz(row_i)$ and $nz(col_j)$ $(1 \le i, j \le n-k+1)$
4: Find an entry $s_{ij}^{(k)}$ of $S^{(k)}$ that minimizes $(nz(row_i)-1)(nz(col_j)-1)$
5: Permute the rows and columns so that $s_{ij}^{(k)}$ is the (1, 1) entry of the permuted $S^{(k)}$
6: Compute Schur complement $S^{(k+1)}$ of the permuted $S^{(k)}$ with respect to its (1, 1) entry
7: **end for**

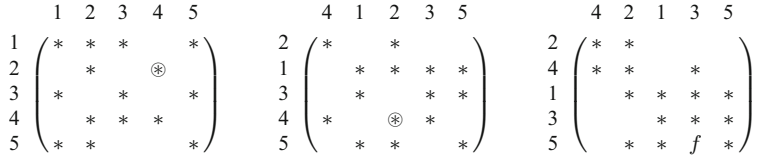


Figure 8.10 Illustration of Markowitz pivoting. The first and second pivots are circled. The sparsity pattern of $A = S^{(1)}$ is on the left. In the centre is the sparsity pattern after permuting the pivot in position $(2, 4)$ to the $(1, 1)$ position of $S^{(1)}$. There is no fill-in after the first factorization step. On the right is the sparsity pattern after selecting the second pivot that has the original position $(4, 2)$ and permuting it to the $(1, 1)$ position of $S^{(2)}$. The resulting filled entry is denoted by f. Note that the nonsymmetric permutations transform the originally irreducible matrix into a reducible one.

Markowitz pivoting as described here only considers the sparsity of A and the subsequent Schur complements. In practice, the pivoting strategy also needs to avoid small pivots because, as discussed in the last chapter, they can lead to numerical instability. A simple improvement is to break ties in Step 4 by choosing from the entries with the minimum Markowitz count the one of largest absolute value.

Because computing row and column counts is expensive, practical implementations may restrict computing them to a limited number of rows and columns. Alternatively, the search may be restricted to a predetermined number of rows of lowest row count (typically two or three rows), choosing entries with best Markowitz count and breaking ties on numerical grounds. Another option is to restrict the pivot choice to diagonal entries, in which case A is permuted symmetrically.

Algorithm 8.5 needs storage formats that can accommodate dynamic changes to the Schur complements. For example, the DS format described in Section 1.3.2, which allows access to both the rows and the columns. However, this format is only feasible if the amount of fill-in during the factorization is not large.

8.4 Global Nested Dissection Orderings

Nested dissection is the most important and widely used global ordering strategy for direct methods when $\mathcal{S}\{A\}$ is symmetric; it is particularly effective for ordering very large matrices. It proceeds by identifying a small set of vertices $\mathcal{V}_\mathcal{S}$ (known as a **vertex separator**) that if removed separates the graph into two disjoint subgraphs described by the vertex subsets $\mathcal{B}$ and $\mathcal{W}$ (commonly called "black" and "white", respectively). The rows and columns belonging to $\mathcal{B}$ are labelled first, then those belonging to $\mathcal{W}$ and finally those in $\mathcal{V}_\mathcal{S}$. The reordered matrix has the form

$$\begin{pmatrix} A_{\mathcal{B},\mathcal{B}} & 0 & A_{\mathcal{B},\mathcal{V}_\mathcal{S}} \\ 0 & A_{\mathcal{W},\mathcal{W}} & A_{\mathcal{W},\mathcal{V}_\mathcal{S}} \\ A^T_{\mathcal{B},\mathcal{V}_\mathcal{S}} & A^T_{\mathcal{W},\mathcal{V}_\mathcal{S}} & A_{\mathcal{V}_\mathcal{S},\mathcal{V}_\mathcal{S}} \end{pmatrix}. \tag{8.2}$$

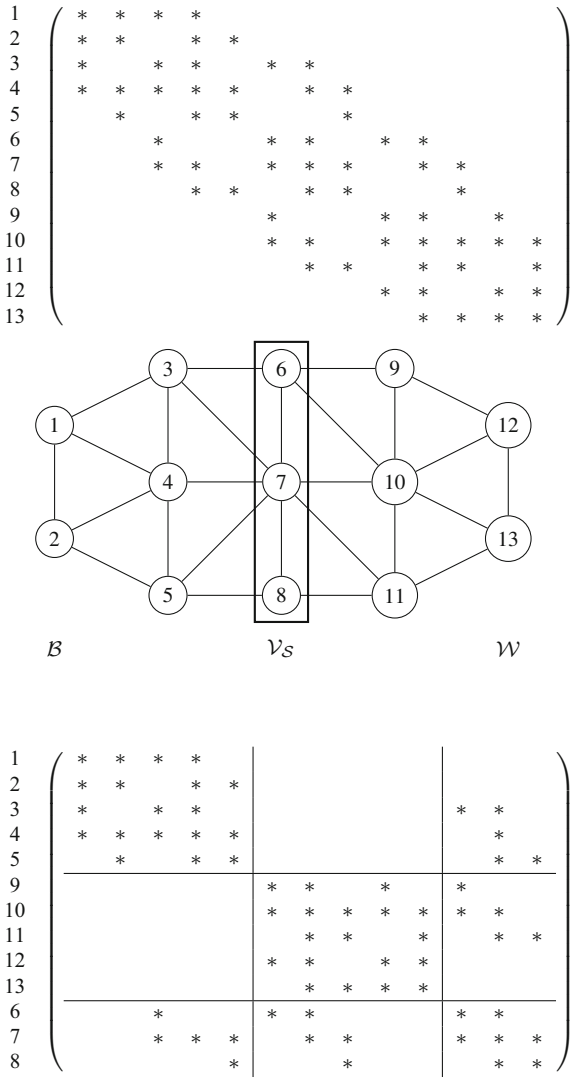


Figure 8.11 A simple example to illustrate nested dissection. The pattern of the original matrix (top), the partitioned graph (centre), and the corresponding symmetrically permuted matrix (bottom) are given.

This is shown for a 13 × 13 example in Figure 8.11. Provided the variables are eliminated in the permuted order, no fill occurs within the zero off-diagonal blocks. If $|\mathcal{V}_S|$ is small and $|\mathcal{B}|$ and $|\mathcal{W}|$ are similar, these zero blocks account for approximately half the possible entries in the matrix. The reordering can be applied recursively to the submatrices $A_{\mathcal{B},\mathcal{B}}$ and $A_{\mathcal{W},\mathcal{W}}$ until the vertex subsets

ALGORITHM 8.6 Nested dissection algorithm
Input: Graph $\mathcal{G}$ of a symmetrically structured matrix A and a partitioning algorithm **PartitionAlg**.
Output: A permutation vector p that defines a new labelling of the vertices of $\mathcal{G}$.

recursive function (p = **nested_dissection**(A, **PartitionAlg**))
  if dissection has terminated **then** ▷ Vertex subsets are smaller than some threshold
    $p = \text{AMD}(\mathcal{V}, \mathcal{E})$ ▷ Compute an AMD ordering
  else
    Use **PartitionAlg**($\mathcal{V}, \mathcal{E}$) to obtain the vertex partitioning ($\mathcal{B}, \mathcal{W}, \mathcal{V}_\mathcal{S}$)
    $p_\mathcal{B}$ = **nested_dissection**($A_{\mathcal{B},\mathcal{B}}$, **PartitionAlg**)
    $p_\mathcal{W}$ = **nested_dissection**($A_{\mathcal{W},\mathcal{W}}$, **PartitionAlg**)
    $p_{\mathcal{V}_\mathcal{S}}$ is an ordering of $\mathcal{V}_\mathcal{S}$
    Set $p = \begin{pmatrix} p_\mathcal{B} \\ p_\mathcal{W} \\ p_{\mathcal{V}_\mathcal{S}} \end{pmatrix}$
  end if
end recursive function

are of size less than some prescribed threshold. At this stage, a local ordering technique (such as AMD) is normally more effective than nested dissection, and so a switch is made. The general form of the nested dissection algorithm is summarized in Algorithm 8.6. The parameter **PartitionAlg** specifies the algorithm used in determining the partitioning of the vertices. The performance and efficacy is highly dependent on the choice of **PartitionAlg**. Originally, level set based methods were used but most current approaches use multilevel techniques that create a hierarchy of graphs, each representing the original graph, but with a smaller dimension. The smallest (that is, the coarsest) graph in the sequence is partitioned. This partition is propagated back through the sequence of graphs, while being periodically refined.

8.5 Bordered Forms

Another possibility to exploit the global matrix structure is to use bordered block forms. These forms can arise naturally in some practical applications.

8.5.1 Doubly Bordered Form

The matrix (8.2) is an example of a **doubly bordered block diagonal (DBBD)** form. More generally, a matrix is said in DBBD form if it has the block structure

$$A_{DB} = \begin{pmatrix} A_{1,1} & & & & C_1 \\ & A_{2,2} & & & C_2 \\ & & \ldots & & . \\ & & & A_{Nb,Nb} & C_{Nb} \\ R_1 & R_2 & \ldots & R_{Nb} & B \end{pmatrix}, \tag{8.3}$$

where $Nb > 1$, the blocks $A_{lb,lb}$ on the diagonal are **square** $n_{lb} \times n_{lb}$ matrices and the border blocks C_{lb} and R_{lb} are $n_{lb} \times n_S$ and $n_S \times n_{lb}$ matrices, respectively, with $n_S \ll n_{lb}$ ($1 \leq lb \leq Nb$). B is an $n_S \times n_S$ matrix. The blocks can have very different sizes. A nested dissection ordering can be used to permute a symmetrically structured matrix A to a symmetrically structured DBBD form ($\mathcal{S}\{R_i\} = \mathcal{S}\{C_i^T\}$). If $\mathcal{S}\{A\}$ is close to symmetric, then nested dissection can be applied to $\mathcal{S}\{A + A^T\}$. In finite-element applications, the DBBD form corresponds to partitioning the underlying finite-element domain into non-overlapping subdomains; each $A_{lb,lb}$ represents the interior of a subdomain and the variables in the borders are those that lie on an interface between two or more subdomains.

Coarse-grained parallel approaches aim to factorize the $A_{lb,lb}$ blocks in parallel before solving the interface problem that connects the blocks. The block factorization of A_{DB} is

$$A_{DB} = \begin{pmatrix} L_1 & & & & \\ & L_2 & & & \\ & & \ldots & & \\ & & & L_{Nb} & \\ \widehat{R}_1 & \widehat{R}_2 & \ldots & \widehat{R}_{Nb} & L_S \end{pmatrix} \begin{pmatrix} U_1 & & & & \widehat{C}_1 \\ & U_2 & & & \widehat{C}_2 \\ & & \ldots & & . \\ & & & U_{Nb} & \widehat{C}_{Nb} \\ & & & & U_S \end{pmatrix},$$

where

$$\widehat{R}_{lb} = R_{lb} U_{lb}^{-1}, \quad \widehat{C}_{lb} = L_{lb}^{-1} C_{lb} \ (1 \leq lb \leq Nb), \quad L_S U_S = B - \sum_{lb=1}^{Nb} \widehat{R}_{lb} \widehat{C}_{lb}.$$

The process is summarized in Algorithm 8.7. Here, for simplicity of notation, the permutation matrices for the block factorizations are set to the identity; in practice, $A_{lb,lb} = P_{lb} L_{lb} U_{lb} Q_{lb}$ for some permutation matrices P_{lb} and Q_{lb} ($1 \leq lb \leq Nb$) and $S = P_S L_S U_S Q_S$ for some permutation matrices P_s and Q_S.

There are several opportunities to incorporate parallelism. First, the factorizations of the blocks $A_{lb,lb}$ on the diagonal are completely independent. In addition,

ALGORITHM 8.7 Coarse-grained parallel LU factorization using DBBD form
Input: Matrix A_{DB} in DBBD form (8.3).
Output: Block LU factorization.

1: Initialise $S = B$
2: **for** $lb = 1 : Nb$ **do**
3: $A_{lb,lb} = L_{lb}U_{lb}$ ▷ LU factorization of square block on diagonal
4: $\widehat{R}_{lb} = R_{lb}U_{lb}^{-1}$ ▷ Triangular solve for bottom-border blocks
5: $\widehat{C}_{lb} = L_{lb}^{-1}C_{lb}$ ▷ Triangular solve for right-border blocks
6: **end for**
7: $S = S - \sum_{lb=1}^{Nb} \widehat{R}_{lb}\widehat{C}_{lb}$ ▷ Assemble updates to interface block
8: $S = L_S U_S$ ▷ Factorize updated interface block (Schur complement)

the factorization of each individual $A_{lb,lb}$ can be parallelized. The same is true for the triangular solves that update the border blocks. Second, the assembly of the interface block S can be partially parallelized (it can be started as soon as the first updated border blocks are available). Third, the LU factorization of S can be parallelized.

Observe that S is generally significantly denser than the other blocks and can present a computational bottleneck. In fact, not only is factorizing S expensive in terms of the memory and operations required, assembly updates to it can be time consuming. This is because multiple submatrices may contribute to the same entry of S, and these cannot be performed at the same time. Furthermore, for an efficient parallel implementation, load balance must be considered. If the work required for factorizing each of the blocks on the diagonal is not similar, then the time will be dominated by the most expensive block. One possible solution is to choose Nb to be greater than the number of processors and use dynamic scheduling to achieve good load balance. Unfortunately, if the number of blocks increases, so too does the size of S.

If A is not SPD, then factorizing the $A_{lb,lb}$ blocks without considering the entries in the border can potentially lead to stability problems. Consider the first step in factorizing $A_{lb,lb}$ and the threshold pivoting test (7.5) for a sparse LU factorization. The pivot candidate $(A_{lb,lb})_{11}$ must satisfy

$$\max\{\max_{i>1} |(A_{lb,lb})_{i1}|, \max_{k} |(R_{lb})_{k1}|\} \leq \gamma^{-1}|(A_{lb,lb})_{11}|,$$

where $\gamma \in (0, 1]$ is the threshold parameter. Large entries in the row border matrix R_{lb} can prevent pivots being selected within $A_{lb,lb}$. Stability can be maintained by moving rows and columns that cannot be eliminated to the borders. This increases the border size and may adversely affect the a priori sparse data structures for holding the factors, increase the work required to perform the factorization, and reduce the potential for parallelism within the factorization of the block.

8.5.2 Singly Bordered Form

An alternative strategy is to permute A to **singly bordered block diagonal (SBBD)** form

$$A_{SB} = \begin{pmatrix} A_{1,1} & & & C_1 \\ & A_{2,2} & & C_2 \\ & & \cdots & \cdot \\ & & A_{Nb,Nb} & C_{Nb} \end{pmatrix},$$

where the blocks $A_{lb,lb}$ are **rectangular** $m_{lb} \times n_{lb}$ matrices with $m_{lb} \geq n_{lb}$ and $\sum_{lb=1}^{Nb} m_l = n$, and the border blocks C_{lb} are of order $m_{lb} \times n_I$ ($n_I \ll n_{lb}$), where $n_I = \sum_{bl=1}^{Nb} (m_{lb} - n_{lb})$. The linear system becomes

$$\begin{pmatrix} A_{1,1} & & & C_1 \\ & A_{2,2} & & C_2 \\ & & \cdots & \cdot \\ & & A_{Nb,Nb} & C_{Nb} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{Nb} \\ x_I \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{Nb} \end{pmatrix}, \tag{8.4}$$

where x_{lb} is of length n_{lb}, x_I is a vector of length n_I of interface variables, and the right-hand side vectors b_{lb} are of length m_{lb}, such that

$$\begin{pmatrix} A_{lb,lb} & C_{lb} \end{pmatrix} \begin{pmatrix} x_{lb} \\ x_I \end{pmatrix} = b_{lb}, \quad 1 \leq lb \leq Nb.$$

A partial factorization of each block matrix is performed, that is,

$$\begin{pmatrix} A_{lb,lb} & C_{lb} \end{pmatrix} = P_{lb} \begin{pmatrix} L_{lb} & \\ \bar{L}_{lb} & I \end{pmatrix} \begin{pmatrix} U_{lb} & \bar{U}_{lb} \\ & S_{lb} \end{pmatrix} Q_{lb}, \tag{8.5}$$

where P_{lb} and Q_{lb} are permutation matrices, L_{lb} and U_{lb} are $n_{lb} \times n_{lb}$ lower and upper triangular matrices, respectively, and if q_{lb} is the number of columns in C_{lb} with at least one entry, S_{lb} is a $(m_{lb} - n_{lb}) \times q_{lb}$ local Schur complement matrix. Pivots can only be chosen from the columns of $A_{lb,lb}$ because the columns of C_{lb} have entries in at least one other border block C_{jb} ($jb \neq lb$). The pivot candidate $(A_{lb,lb})_{11}$ at the first elimination step must satisfy

$$\max_{i>1} |(A_{lb,lb})_{i1}| \leq \gamma^{-1} |(A_{lb,lb})_{11}|,$$

and provided A is nonsingular, there will always be a numerically satisfactory pivot in column 1 of $A_{lb,lb}$. The same is true at each elimination step so that n_{lb} pivots can be chosen. An $n_I \times n_I$ matrix S is obtained by assembling the Nb local

ALGORITHM 8.8 Coarse-grained parallel LU factorization and solve using SBBD form
Input: Linear system in SBBD form (8.4).
Output: Block LU factorization and computed solution x.

1: $S = 0$ and $z_I = 0$
2: **for** $lb = 1 : Nb$ **do**
3: Perform a partial LU factorization (8.5) of $(A_{lb,lb},\ C_{lb})$.
4: Solve $P_{lb} \begin{pmatrix} L_{lb} \\ \bar{L}_{lb} & I \end{pmatrix} \begin{pmatrix} y_{lb} \\ \bar{y}_{lb} \end{pmatrix} = b_{lb}$
5: $S = S + S_{lb}$ and $z_I = z_I + \bar{y}_{lb}$ ▷ Assemble S and z_I
6: **end for**
7: $S = P_s L_s U_s Q_s$ ▷ P_s and Q_s are permutation matrices
8: Solve $P_s L_s\, y_I = z_I$ and then $U_s Q_s\, x_I = y_I$ ▷ Forward then back substitution
9: **for** $lb = 1 : Nb$ **do**
10: Solve $U_{lb}\, Q_{lb}\, x_{lb} = y_{lb} - \bar{U}_{lb}\, Q_{lb}\, x_I$
11: **end for**

Schur complement matrices S_{lb}. The approach is summarized as Algorithm 8.8. The operations on the submatrices can be performed in parallel.

8.5.3 *Ordering to Singly Bordered Form*

The objective is to permute A to an SBBD form with a narrow column border. One way to do this is to choose the number $Nb > 1$ of required blocks and use nested dissection to compute a vertex separator $\mathcal{V}_S$ of $\mathcal{G}(A + A^T)$ such that removing $\mathcal{V}_S$ and its incident edges splits $\mathcal{G}(A + A^T)$ into Nb components. Then initialize the set $\mathcal{S}_C$ of border columns to $\mathcal{V}_S$ and let $\mathcal{V}_{1b}, \mathcal{V}_{2b}, \ldots, \mathcal{V}_{Nb}$ be the subsets of column indices of A that correspond to the Nb components and let $n_{i,kb}$ be the number of column indices in row i that belong to $\mathcal{V}_{kb}$. If $lb = \arg\max_{1 \le kb \le Nb} |n_{i,kb}|$, then row i is assigned to partition lb. All column indices in row i that do not belong to $\mathcal{V}_{lb}$ are moved into $\mathcal{S}_C$. Once all the rows have been considered, the only rows that remain unassigned are those that have all their nonzero entries in $\mathcal{V}_S$. Such rows can be assigned equally to the Nb partitions. If $j \in \mathcal{S}_C$ is such that column j of A has nonzero entries only in rows belonging to partition kb, then j can be removed from $\mathcal{S}_C$ and added to $\mathcal{V}_{kb}$. The procedure is outlined as Algorithm 8.9. The computed vector $block$ and set $\mathcal{S}_C$ can be used to define permutation matrices P and Q such that $PAQ = A_{SB}$. In practice, it may be necessary to modify the algorithm to ensure a good row balance between the number of rows in the blocks; this may lead

ALGORITHM 8.9 SBBD ordering of a general matrix

Input: Matrix A, the number $Nb > 1$ of blocks and corresponding vertex separator $\mathcal{V}_S$ of $\mathcal{G}(A + A^T)$.

Output: Vector $block$ such that $block(i)$ denotes the partition in the SBBD form to which row i is assigned ($1 \leq i \leq n$) and $\mathcal{S}_C$ is the set of border columns.

Initialise $\mathcal{S}_C = \mathcal{V}_S$ and $block(1:n) = 0$
Initialise $\mathcal{V}_{kb}$ to hold the column indices of A that correspond to component kb of $\mathcal{G}(A + A^T)$ after the removal of $\mathcal{V}_S$, $1 \leq kb \leq Nb$
for each row i **do**
    Add up the number $n_{i,kb}$ of column indices belonging to $\mathcal{V}_{kb}$, $1 \leq kb \leq Nb$
    Find $lb = \arg\max_{1 \leq kb \leq Nb} n_{i,kb}$
    $block(i) = lb$
    for each column index j in row i **do**
        if $j \in \mathcal{V}_{kb}$ and $kb \neq lb$ **then**
            Remove j from $\mathcal{V}_{kb}$ and add to $\mathcal{S}_C$
        end if
    end for
end for
Assign the rows i for which $block(i) = 0$ equally between the Nb partitions.
If some column $j \in \mathcal{S}_C$ has nonzero entries only in rows belonging to partition kb then remove j from $\mathcal{S}_C$ and add to $\mathcal{V}_{kb}$

to a larger $\mathcal{S}_C$. It is also necessary to avoid adding in duplicate column indices into $\mathcal{S}_C$ (alternatively, a final step can be added that removes duplicates).

The matching-based orderings discussed in Section 6.3 that permute off-diagonal entries onto the diagonal can increase the symmetry index of the resulting reordered matrix, particularly in cases where A is very sparse with a large number of zeros on the diagonal. Frequently, applying a matching ordering before ordering to SBBD form reduces the number of columns in $\mathcal{S}_C$.

8.6 Notes and References

The most influential early paper on orderings for sparse symmetric matrices is that of Tinney & Walker (1967). It first proposed the minimum degree algorithm (referred to as scheme 2) and the minimum fill-in algorithm (referred to as scheme 3). The fast implementation of the minimum degree algorithm using quotient graphs is summarized by George & Liu (1980a). Further developments were made throughout the 1980s, including the multiple minimum degree variant, mass

elimination and external degree; key references are Liu (1985) and George & Liu (1989). An important development in the 1990s was the approximate minimum degree algorithm of Amestoy et al. (1996). Modifying the AMD algorithm for matrices with some dense rows is discussed in Dollar & Scott (2010). For a careful description of different variants of the minimum degree strategy and their complexity we recommend Heggernes et al. (2001). Rothberg & Eisenstat (1998) consider both minimum degree and minimum fill strategies and (Erisman et al., 1987) provide an early evaluation of different strategies for nonsymmetric matrices.

Jennings (1966) presents the first envelope method for sparse Cholesky factorizations. The Cuthill-McKee algorithm comes from the paper by Cuthill & McKee (1969). The GPS algorithm was originally introduced in Gibbs et al. (1976). The book by George & Liu (1981) gives a detailed description of the algorithm while Meurant (1999) includes an enlightening discussion of the relation between the CM and RCM algorithms. A quick search of the literature shows that a large number of bandwidth and profile reduction algorithms have been (and continue to be) reported. Many have their origins in the Cuthill-McKee and GPS algorithms. A widely used two-stage variant that employs level sets is the so-called Sloan algorithm (Sloan, 1986); see also Reid & Scott (1999) for details of an efficient implementation. The use of the Fiedler vector to obtain spectral orderings is introduced in Barnard et al. (1995), with analysis given in George & Pothen (1997). A hybrid algorithm that combines the spectral method with the second stage of Sloan's algorithm to further reduce the profile is proposed in Kumfert & Pothen (1997) and a multilevel variant is given by Hu & Scott (2001). de Oliveira et al. (2018) provide a recent comparison of many bandwidth and profile reduction algorithms.

Reducing the bandwidth when A is nonsymmetric is discussed by Reid & Scott (2006). For highly nonsymmetric A, Scott (1999) applies a modified Sloan algorithm applied to the row graph (that is, $\mathcal{G}(AA^T)$) to derive an effective ordering of the rows of A for use with a frontal solver. The approach originally proposed by Markowitz (1957) for finding pivots during an LU factorization is incorporated (in modified form) in a number of serial LU factorization codes, including the early solvers `MA28` and Y12M (Duff, 1980 and Zlatev, 1991, respectively) as well as `MA48` (Duff & Reid, 1996). The book of Duff et al. (2017) includes detailed discussions. To limit permutations to being symmetric, Amestoy et al. (2007) propose minimizing the Markowitz count among the diagonal entries.

A seminal paper on global orderings is George (1973), but a real revolution in the field followed the theoretical analysis of the application of nested dissection for general symmetrically structured sparse matrices given in Lipton et al. (1979). For subsequent extensions discussing separator sizes we suggest Agrawal et al. (1993), Teng (1997), and Spielman & Teng (2007).

From the early 1990s onwards, there have been numerous contributions to graph partitioning algorithms. Significant developments, including multilevel algorithms, have been driven in part by the design and development of mathematical software, notably the well-established packages METIS (2022) and Scotch (2022); both offer versions for sequential and parallel graph partitioning (see also the papers by Karypis & Kumar, 1998a,b and Chevalier & Pellegrini, 2008). The book by

Bichot & Siarry (2013) discusses a number of contributions, including hypergraph partitioning, which is well suited to parallel computational models (see, for example, Uçar & Aykanat, 2007 and references to the use of hypergraphs given in the survey article of Davis et al., 2016; they can also be used for profile reduction Acer et al., 2019).

Hu et al. (2000) present a serial algorithm for ordering nonsymmetric A to SBBD form; an implementation is available as `HSL_MC66` within the HSL mathematical software library. Algorithm 8.9 is from Hu & Scott (2005) (see also Duff & Scott, 2005). Alternatively, hypergraphs can be used for SBBD orderings. The best-known packages are the serial code PaToH of Aykanat et al. (2004) and the parallel code PHG from Zoltan (2022).