Chapter 6
Sparse LU Factorizations

The closer one looks, the more subtle and remarkable Gaussian elimination appears – Trefethen (1985)

Gaussian elimination is living mathematics. It has mutated successfully for the last two hundred years to meet changing social needs – Grcar (2011)

This chapter considers the LU factorization of a general nonsymmetric nonsingular sparse matrix A. In practice, numerical pivoting for stability and/or ordering of A to limit fill-in in the factors is often needed and the computed factorization is then of a permuted matrix PAQ. Pivoting is discussed in Chapter 7 and ordering algorithms in Chapter 8.

6.1 Sparse LU Factorizations and Their Graph Models

In Chapter 4, graphs were used to describe structural changes during a sparse Cholesky factorization. In particular, the elimination tree was shown to play a key role and, in the previous chapter, the use of DAGs was discussed. For general matrices, there are a number of ways that graphs can be employed.

6.1.1 Use of Elimination DAGs

The first graph model uses the elimination DAGs associated with L and U that were defined in (2.1)–(2.2). The following observation, which is illustrated in Figure 6.1, generalizes Observation 4.1 to nonsymmetric matrices.

Observation 6.1 *If $i > j$ and $u_{ji} \neq 0$, then the* **column replication principle** *states*

J. Scott, M. Tůma, *Algorithms for Sparse Linear Systems*, Nečas Center Series,
https://doi.org/10.1007/978-3-031-25820-6_6

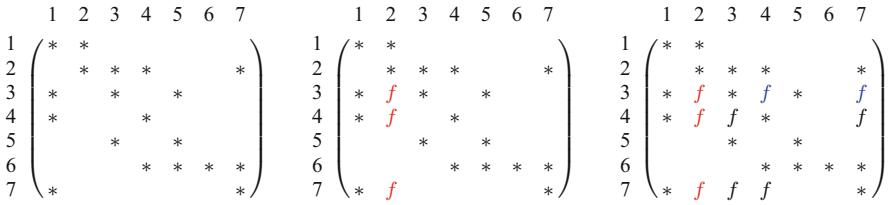

Figure 6.1 An illustration of the column and row replication principles of sparse LU factorizations. The matrix A is on the left. In the centre, we show in red the filled entries in L resulting from the replication of the first column in the second column because $u_{12} \neq 0$. On the right, we show in blue the filled entries in U resulting from the replication of the second row in the third row because $l_{32} \neq 0$. Other filled entries resulting from subsequent steps of the factorization are denoted in black.

$$\mathcal{S}\{L_{i:n,j}\} \subseteq \mathcal{S}\{L_{i:n,i}\},$$

that is, the pattern of column j of L (rows i to n) is replicated in the pattern of column i of L. Analogously, if $i > j$ and $l_{ij} \neq 0$, then the **row replication principle** *states*

$$\mathcal{S}\{U_{j,i:n}\} \subseteq \mathcal{S}\{U_{i,i:n}\},$$

that is, the pattern of row j of U (columns i to n) is replicated in the pattern of row i of U.

Algorithm 6.1 outlines a basic sparse LU factorization. Here it is assumed that A is factorizable so that pivoting is not needed. The remainder of this chapter looks at techniques that can be used to develop the approach into an efficient one.

The following theorem formulates the recursive column replication and the replication of nonzeros along rows of L using directed paths in $\mathcal{G}(U)$; an analogous result holds for the rows of U and directed paths in $\mathcal{G}(L^T)$.

Theorem 6.1 (Gilbert & Liu 1993) *Assume that for some $k < j$ there is a directed path $k \overset{\mathcal{G}(U)}{\Longrightarrow} j$. Then*

$$\mathcal{S}\{L_{j:n,k}\} \subseteq \mathcal{S}\{L_{j:n,j}\}. \tag{6.1}$$

Moreover, if $l_{ik} \neq 0$ for some $i > j$, then $l_{is} \neq 0$ for all vertices s on this path.

The next two theorems generalize Theorem 4.3 to A being a general nonsymmetric matrix.

Theorem 6.2 (Gilbert & Liu 1993) *If $a_{ij} = 0$ and $i > j$, then there is a filled entry $l_{ij} \neq 0$ if and only if there exists $k < j$ such that $a_{ik} \neq 0$ and there is a directed path $k \overset{\mathcal{G}(U)}{\Longrightarrow} j$.*

ALGORITHM 6.1 Basic sparse LU factorization
Input: Nonsymmetric and factorizable matrix $A = L_A + D_A + U_A$.
Output: LU factorization $A = LU$.

```
1: L = I + L_A                          ▷ Identity plus strictly lower triangular part of A
2: U = D_A + U_A                        ▷ Diagonal plus strictly upper triangular part of A
3: for k = 1 : n − 1 do
4:     for i ∈ {i > k | l_ik ≠ 0} do
5:         l_ik = l_ik / u_kk
6:         U_{i,i:n} = U_{i,i:n} − U_{k,i:n} l_ik        ▷ Update row i of U
7:     end for
8:     for j ∈ {j > k | u_kj ≠ 0} do
9:         L_{j+1:n,j} = L_{j+1:n,j} − L_{j+1:n,k} u_kj  ▷ Update column j of L
10:    end for
11: end for
```

Theorem 6.3 (Gilbert & Liu 1993) *If $a_{ij} = 0$ and $i < j$, then there is a filled entry $u_{ij} \neq 0$ if and only if there exists $k < i$ such that $a_{kj} \neq 0$ and there is a directed path $k \xrightarrow{\mathcal{G}(L^T)} i$.*

Theorems 6.2 and 6.3 are demonstrated in Figure 6.2. Consider the directed path $1 \to 3 \to 5 \to 6$ in $\mathcal{G}(U)$. Existence of this path implies the fill-in in L, first in

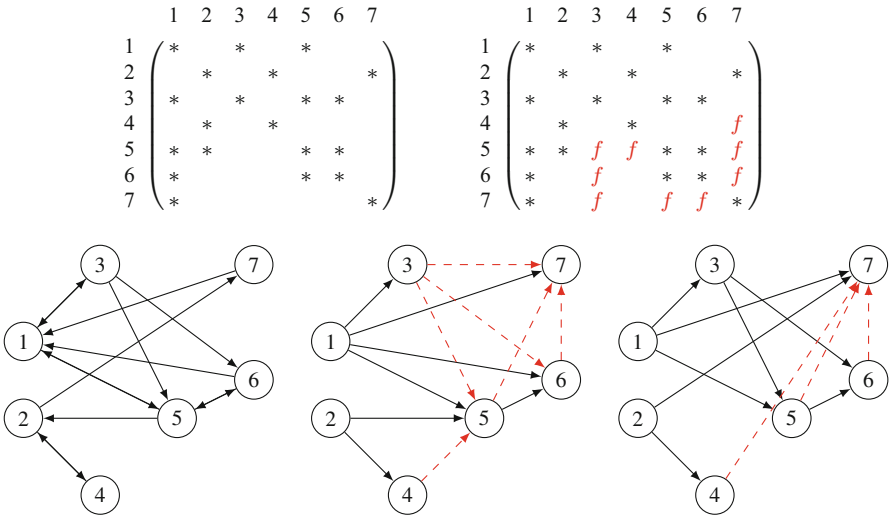

Figure 6.2 The sparsity patterns of A (left) and $L+U$ (right) together with the graphs $\mathcal{G}(A)$ (left), $\mathcal{G}(L^T)$ (centre) and $\mathcal{G}(U)$ (right). The filled entries are denoted by f and the corresponding edges are the red dashed lines.

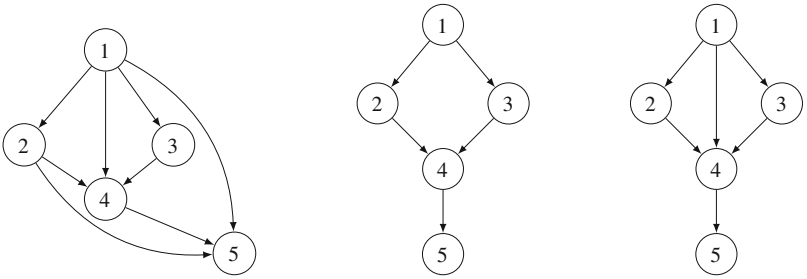

Figure 6.3 Example to show the transitive reduction of a DAG. $\mathcal{G}$ is on the left, its transitive reduction $\mathcal{G}^0$ is in the centre, and one possible $\mathcal{G}'$ that is equireachable with $\mathcal{G}$ is on the right.

column 3, then in columns 5 and 6. Similarly, the directed path $2 \rightarrow 4 \rightarrow 5 \rightarrow 6$ in $\mathcal{G}(L^T)$ implies fill-in at positions (4, 7), (5, 7) and (6, 7) in U.

6.1.2 Transitive Reduction and Equireachability

To employ $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$ in efficient algorithms, they need to be simplified. One possibility is to use transitive reductions that are sparser and preserve reachability within the graphs. A subgraph $\mathcal{G}^0 = (\mathcal{V}, \mathcal{E}^0)$ is a **transitive reduction** of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ if the following conditions hold:

($T1$) there is a path from vertex i to vertex j in $\mathcal{G}$ if and only if there is a path from i to j in $\mathcal{G}^0$ (reachability condition), and
($T2$) there is no subgraph with vertex set $\mathcal{V}$ that satisfies ($T1$) and has fewer edges (minimality condition).

A transitive reduction is unique for a DAG, as shown in the following theorem and illustrated in Figure 6.3.

Theorem 6.4 (Aho et al. 1972) *Let $\mathcal{G}$ be a DAG. The transitive reduction $\mathcal{G}^0$ of $\mathcal{G}$ is unique and is the subgraph that has an edge for every path in $\mathcal{G}$ and has no proper subgraph with this property.*

If $\mathcal{S}\{A\}$ is symmetric, then, as illustrated in Figure 6.4, the role of the transitive reduction is played by the elimination tree.

Theorem 6.5 (Liu 1990; Eisenstat & Liu 2005a) *If A is symmetrically structured, then the transitive reduction of the DAG $\mathcal{G}(L^T)$ (= $\mathcal{G}(U)$) is the elimination tree $\mathcal{T}(A)$.*

Obtaining the exact transitive reduction of a DAG can be expensive. Instead, approximate reductions that drop the minimality condition may be computed. A directed graph $\mathcal{G}'$ with the same vertex set as $\mathcal{G}$ that satisfies condition ($T1$) is said

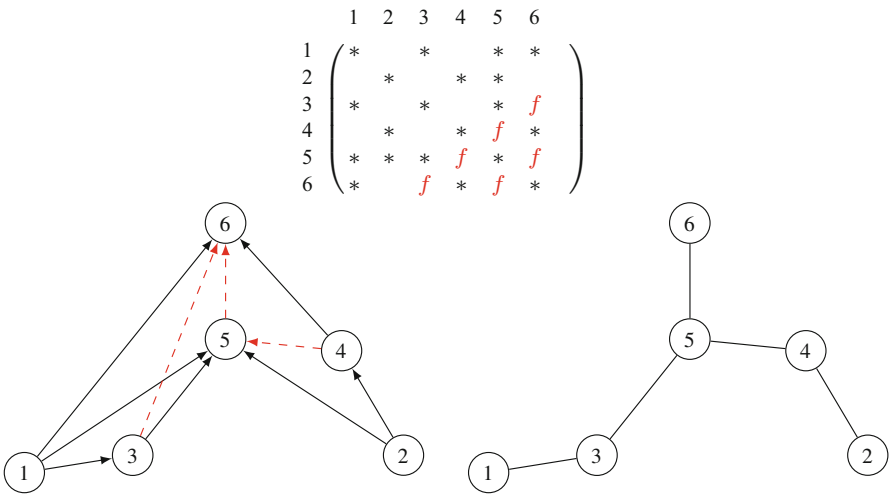

Figure 6.4 The sparsity patterns of $L+U$ of a symmetrically structured A together with the DAG $\mathcal{G}(L^T)$ (left) and the elimination tree $\mathcal{T}(A)$ (right). The filled entries are denoted by f and the corresponding edges are the red dashed lines. It is straightforward to see that $\mathcal{T}(A)$ is obtained as the transitive reduction of $\mathcal{G}(L^T)$.

to be **equireachable** with $\mathcal{G}$. The next result is a simplification of Theorem 6.1; an analogous result holds for the sparsity patterns of the rows of U.

Theorem 6.6 (Gilbert & Liu 1993) *Assume $\mathcal{G}'$ is equireachable with $\mathcal{G}(U)$ and for some $k < j$ there is a directed path $k \overset{\mathcal{G}'}{\Rightarrow} j$. Then (6.1) holds. Moreover, if $l_{ik} \neq 0$ for some $i > j$, then $l_{is} \neq 0$ for all vertices s on the directed path.*

Equireachability enables sparse triangular linear systems to be solved more efficiently. In Chapter 5, Theorem 5.2 describes how to obtain the sparsity pattern $\mathcal{J}$ of the solution of a lower triangular system using paths in $\mathcal{G}(L^T)$. This graph can be replaced by any graph that is equireachable with $\mathcal{G}(L^T)$. Equireachability also allows Theorems 6.2 and 6.3 to be rewritten using paths in a graph $\mathcal{G}'$ that is equireachable with $\mathcal{G}$.

Theorem 6.7 (Gilbert & Liu 1993) *If $a_{ij} = 0$ and $i > j$, then there is a filled entry $l_{ij} \neq 0$ if and only if there exists $k < j$ such that $a_{ik} \neq 0$ and a directed path $k \overset{\mathcal{G}'(U)}{\Longrightarrow} j$, where $\mathcal{G}'(U)$ is equireachable with $\mathcal{G}(U)$.*

Theorem 6.8 (Gilbert & Liu 1993) *If $a_{ij} = 0$ and $i < j$, then there is a filled entry $u_{ij} \neq 0$ if and only if there exists $k < i$ such that $a_{kj} \neq 0$ and a directed path $k \overset{\mathcal{G}'(L^T)}{\Longrightarrow} i$, where $\mathcal{G}'(L^T)$ is equireachable with $\mathcal{G}(L^T)$.*

Figure 6.5 depicts $\mathcal{G}(U)$ and $\mathcal{G}'(U)$ for the matrix in Figure 6.2.

A description of the sparsity patterns of the columns of L can be obtained from the Schur complement (3.2) as follows:

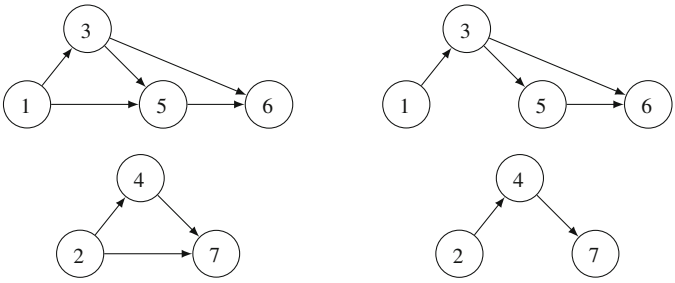

Figure 6.5 The DAG $\mathcal{G}(U)$ for the matrix from Figure 6.2 (left) and $\mathcal{G}'(U)$ which is equireachable with $\mathcal{G}(U)$ (right).

$$\mathcal{S}\{L_{j:n,j}\} = \mathcal{S}\{A_{j:n,j}\} \bigcup_{k<j, u_{kj}\neq 0} \mathcal{S}\{L_{j:n,k}\}, \quad 1 \leq j \leq n.$$

Theorem 6.7 implies that not all the terms in this union are needed to obtain $\mathcal{S}\{L_{j:n,j}\}$. This result is given in Theorem 6.9, which shows how $\mathcal{S}\{L\}$ can be computed by columns if $\mathcal{G}'(U)$ that is equireachable with $\mathcal{G}(U)$ is known.

Theorem 6.9 (Gilbert & Liu 1993) *If $\mathcal{G}'(U)$ is equireachable with $\mathcal{G}(U)$, then*

$$\mathcal{S}\{L_{j:n,j}\} = \mathcal{S}\{A_{j:n,j}\} \bigcup_{(k\rightarrow j)\in\mathcal{E}(\mathcal{G}'(U))} \mathcal{S}\{L_{j:n,k}\}, \quad 1 \leq j \leq n. \tag{6.2}$$

Proof Consider an edge $(k \rightarrow j)$ in $\mathcal{G}(U)$ but not in $\mathcal{G}'(U)$. Repeatedly applying (6.1) along the directed path $k \overset{\mathcal{G}'(U)}{\Longrightarrow} j$, we see that $L_{j:n,k}$ is contained in the right-hand side of (6.2) and therefore $\mathcal{S}\{L_{j:n,j}\}$ is contained in the right-hand side of (6.2). Because the right-hand side of (6.2) is trivially contained in the left-hand side, the result follows. □

An analogous result holds for the rows of U.

Theorem 6.10 (Gilbert & Liu 1993) *If $\mathcal{G}'(L)$ is equireachable with $\mathcal{G}(L)$, then*

$$\mathcal{S}\{U_{i,i:n}\} = \mathcal{S}\{A_{i,i:n}\} \bigcup_{(k\rightarrow i)\in\mathcal{E}(\mathcal{G}'(L^T))} \mathcal{S}\{U_{k,i:n}\}, \quad 1 \leq i \leq n.$$

As an example of Theorem 6.9, consider the matrix in Figure 6.2. Because $(3 \rightarrow 5)$ is the only edge of $\mathcal{G}'(U)$ in the union on the right-hand side of (6.2), $\mathcal{S}\{L_{5:7,5}\}$ is given by

$$\mathcal{S}\{L_{5:7,5}\} = \mathcal{S}\{A_{5:7,5}\} \cup \mathcal{S}\{L_{5:7,3}\}.$$

We can see this from the graph $\mathcal{G}'(U)$ in Figure 6.5 (top right).

6.1.3 Symbolic LU Factorizations Using DAGs

Factorization by bordering can be used to obtain $\mathcal{S}\{L\}$ by rows and $\mathcal{S}\{U\}$ by columns. Assume the sparsity patterns of the first $k-1$ rows of L and the first $k-1$ columns of U ($1 < k \leq n$) have been computed. At step k, the factors satisfy

$$A_{1:k,1:k} = \begin{pmatrix} A_{1:k-1,1:k-1} & A_{1:k-1,k} \\ A_{k,1:k-1} & a_{kk} \end{pmatrix} = \begin{pmatrix} L_{1:k-1,1:k-1} & 0 \\ L_{k,1:k-1} & 1 \end{pmatrix} \begin{pmatrix} U_{1:k-1,1:k-1} & U_{1:k-1,k} \\ 0 & u_{kk} \end{pmatrix}. \tag{6.3}$$

Equating terms for the (2, 1) block, row k of L satisfies

$$L_{k,1:k-1}U_{1:k-1,1:k-1} = A_{k,1:k-1},$$

or, equivalently, if y denotes the off-diagonal part of the column k of L^T, then it is the solution of the lower triangular system

$$U^T_{1:k-1,1:k-1}y = A^T_{k,1:k-1}.$$

From Theorem 5.2, the sparsity pattern of y is the set of all vertices reachable in the DAG $\mathcal{G}(U_{1:k-1,1:k-1})$ (or in a graph that is equireachable with it) from the nonzeros in $A_{k,1:k-1}$. Similarly, equating terms in (6.3) for the (1, 2) block, column k of U satisfies

$$L_{1:k-1,1:k-1}U_{1:k-1,k} = A_{1:k-1,k}.$$

Again, its sparsity pattern can be determined using Theorem 5.2 and the DAG $\mathcal{G}(L^T_{1:k-1,1:k-1})$. The diagonal entry u_{kk} is then computed as $a_{kk} - L_{k,1:k-1}U_{1:k-1,k}$. This shows that determining the sparsity patterns of L and U and computing their numerical values is coupled: computation of the factors needs be mutually interleaved because computing part of one requires information from a part of the other.

6.1.4 Graph Pruning

Consider the matrices in Figure 6.6. The one in the centre is the same as the one on the left except that the entries in positions (4, 6) and (6, 4) have been removed (that is, pruned). Both matrices have the same sets of reachable vertices in $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$. This suggests how to find $\mathcal{G}'(L^T)$ and $\mathcal{G}'(U)$ that are equireachable with $\mathcal{G}(L^T)$ and $\mathcal{G}(U)$, respectively.

Theorem 6.11 (Eisenstat & Liu 1992) *If for some $j < s$ both $l_{sj} \neq 0$ and $u_{js} \neq 0$, then there are no edges $(j \to k)$ with $k > s$ in the transitive reductions of $\mathcal{G}(U)$ and $\mathcal{G}(L^T)$.*

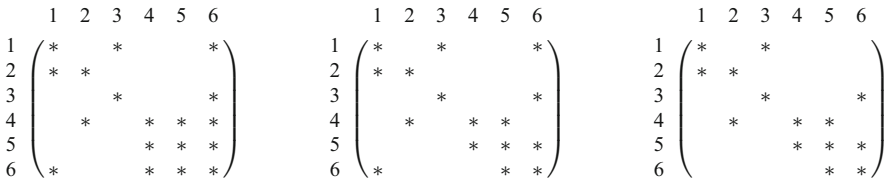

Figure 6.6 An example of symmetric pruning. On the left is $\mathcal{S}\{L+U\}$. In the centre is the reduced sparsity pattern obtained by symmetric pruning. On the right is the reduced sparsity pattern that results from symmetric path pruning.

Proof Let $(j \to k)$ be an edge of $\mathcal{G}(U)$, that is, $u_{jk} \neq 0$. Because $l_{sj} \neq 0$ and $u_{jk} \neq 0$ implies that $u_{sk} \neq 0$, there is a path $j \to s \to k$ in $\mathcal{G}(U)$ and the edge $(j \to k)$ does not belong to the transitive reduction of $\mathcal{G}(U)$. The result for $\mathcal{G}(L^T)$ can be seen analogously. □

This theorem implies that if for some $s > 1$ there are edges

$$j \xrightarrow{\mathcal{G}(L^T)} s \quad \text{and} \quad j \xrightarrow{\mathcal{G}(U)} s,$$

then all edges $(j \to k)$ in $\mathcal{G}(U)$ and $\mathcal{G}(L^T)$ with $k > s$ can be pruned. The resulting DAGs $\mathcal{G}'(U)$ and $\mathcal{G}'(L^T)$ have fewer edges and are equireachable with $\mathcal{G}(U)$ and $\mathcal{G}(L^T)$, respectively. The removal of redundant edges based on Theorem 6.11 is called **symmetric pruning**.

There are other ways to perform pruning. For example, if for some $s > 1$ there are paths

$$j \xRightarrow{\mathcal{G}(L^T)} s \quad \text{and} \quad j \xRightarrow{\mathcal{G}(U)} s,$$

then for all $k > s$ **symmetric path pruning** removes the edges $(j \to k)$ from $\mathcal{G}(U)$ and $\mathcal{G}(L^T)$. Consider again Figure 6.6. In the centre is the sparsity pattern after symmetric pruning and on the right is the reduced sparsity pattern that results from symmetric path pruning. The edge $(1 \to 6)$ is not required in $\mathcal{G}'(L^T)$ or $\mathcal{G}'(U)$ because there are paths

$$1 \xrightarrow{\mathcal{G}(L^T)} 2 \xrightarrow{\mathcal{G}(L^T)} 4 \xrightarrow{\mathcal{G}(L^T)} 5 \xrightarrow{\mathcal{G}(L^T)} 6 \quad \text{and} \quad 1 \xrightarrow{\mathcal{G}(U)} 3 \xrightarrow{\mathcal{G}(U)} 6.$$

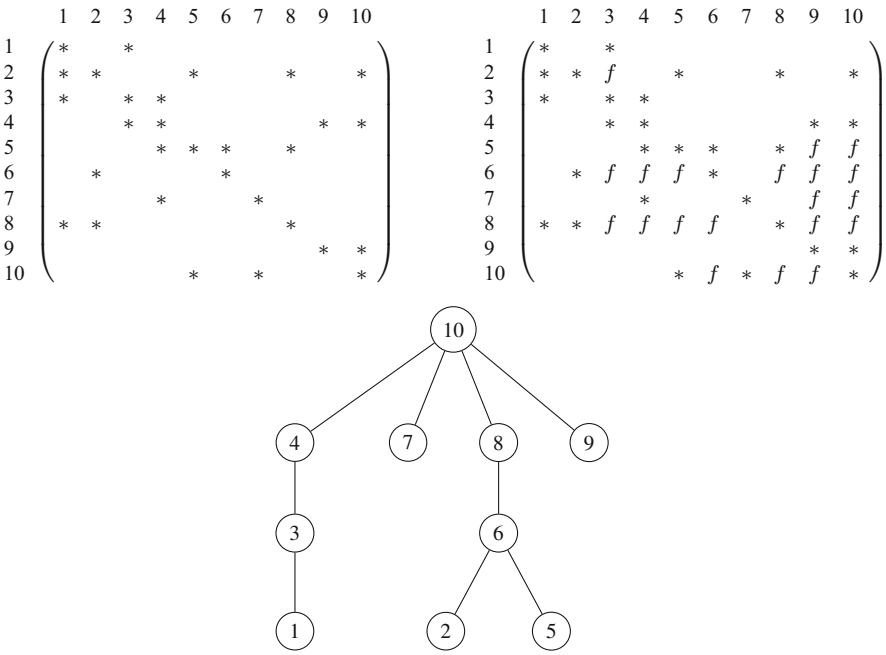

Figure 6.7 An example of the sparsity pattern of a nonsymmetric matrix A (left), $\mathcal{S}\{L+U\}$ with filled entries denoted by f (right) and its elimination tree.

6.1.5 *Elimination Trees for Nonsymmetric Matrices*

The elimination DAGs $\mathcal{G}(L)$ and $\mathcal{G}(U)$ can be combined into a single structure called the **nonsymmetric elimination tree** in which edges are replaced by paths. This can be advantageous because it is more compact. From (4.3), if $\mathcal{S}\{A\}$ is symmetric, then its elimination tree is defined in terms of the mapping

$$parent(j) = \min\{i \mid i > j \text{ and } l_{ij} \neq 0\}.$$

The condition $l_{ij} \neq 0$ is equivalent to $i \xrightarrow{\mathcal{G}(L)} j \xrightarrow{\mathcal{G}(L^T)} i$. In the nonsymmetric case, the definition can be generalized using directed paths

$$parent(j) = \min\{i \mid i > j \text{ and } i \xRightarrow{\mathcal{G}(L)} j \xRightarrow{\mathcal{G}(U)} i\}. \tag{6.4}$$

This is illustrated in Figure 6.7. Vertices 6, 8, and 10 are the only ones with cycles of the form

$$i \xRightarrow{\mathcal{G}(L)} 2 \xRightarrow{\mathcal{G}(U)} i,$$

namely,

ALGORITHM 6.2 Basic computation of the elimination tree for nonsymmetric A
Input: Digraph $\mathcal{G}(A)$.
Output: The elimination tree given by the mapping $parent$.

```
1:  parent(1 : n) = 0
2:  for i = 1 : n do
3:      Find the vertex set V_C of the strong component of G(A_{1:i,1:i}) that contains i
4:      for j ∈ V_C \ {i} do
5:          if parent(j) = 0 then
6:              parent(j) = i
7:          end if
8:      end for
9:      parent(i) = 0
10: end for
```

$$6 \xrightarrow{\mathcal{G}(L)} 2 \xrightarrow{\mathcal{G}(U)} 5 \xrightarrow{\mathcal{G}(U)} 6, \quad 8 \xrightarrow{\mathcal{G}(L)} 2 \xrightarrow{\mathcal{G}(U)} 8 \quad \text{and} \quad 10 \xrightarrow{\mathcal{G}(L)} 6 \xrightarrow{\mathcal{G}(L)} 2 \xrightarrow{\mathcal{G}(U)} 10.$$

In this example, $parent(2) = 6$.

Theorem 6.12, which can be regarded as a generalization of Corollary 4.6, shows how the elimination tree for nonsymmetric A can be constructed.

Theorem 6.12 (Eisenstat & Liu 2005a) *Let A be a nonsymmetric matrix. $i = parent(j)$ if and only if $i > j$ and i is the smallest vertex that belongs to the same strong component of $\mathcal{G}(A_{1:i,1:i})$ as vertex j.*

This result is employed in Algorithm 6.2. The complexity of finding the strong components of a digraph with m edges and n vertices is $O(n + m)$. Hence, the complexity of Algorithm 6.2 is $O(nz(A)\, n)$. More sophisticated approaches with complexity $O(nz(A) \log n)$ exist.

To illustrate Algorithm 6.2, consider the matrix and its elimination tree depicted in Figure 6.7. The main loop sets the first nonzero value in the array $parent$ when $i = 3$ because this is the first i for which the set $\mathcal{V}_C \setminus \{i\}$ is non empty; it is equal to $\{1\}$ and thus $parent(1) = i = 3$. For $i = 4$, the vertex set $\{1, 3, 4\}$ forms a strong component of $\mathcal{G}(A_{1:4,1:4})$ and so $parent(3) = 4$. For $i = 5$, the single vertex $\{5\}$ is a strong component of $\mathcal{G}(A_{1:5,1:5})$ and, therefore, 5 is not a parent of any other vertex (it is a leaf vertex). $\mathcal{G}(A_{1:6,1:6})$ has two strong components with vertex sets $\{1, 3, 4\}$ and $\{2, 5, 6\}$. $i = 6$ belongs to the second of these and thus the algorithm sets $parent(j) = i = 6$ for $j = 2$ and 5.

An attractive idea for constructing $\mathcal{S}\{L + U\}$ and subsequently computing the LU factorization is based on using the **column elimination tree** $\mathcal{T}(A^T A)$.

Theorem 6.13 (George & Ng 1985; Grigori et al. 2009) *Assume all the diagonal entries of A are nonzero and let $\widetilde{L}\widetilde{L}^T$ be the Cholesky factorization of $A^T A$. Then for any row permutation matrix P such that $PA = LU$ the following holds:*

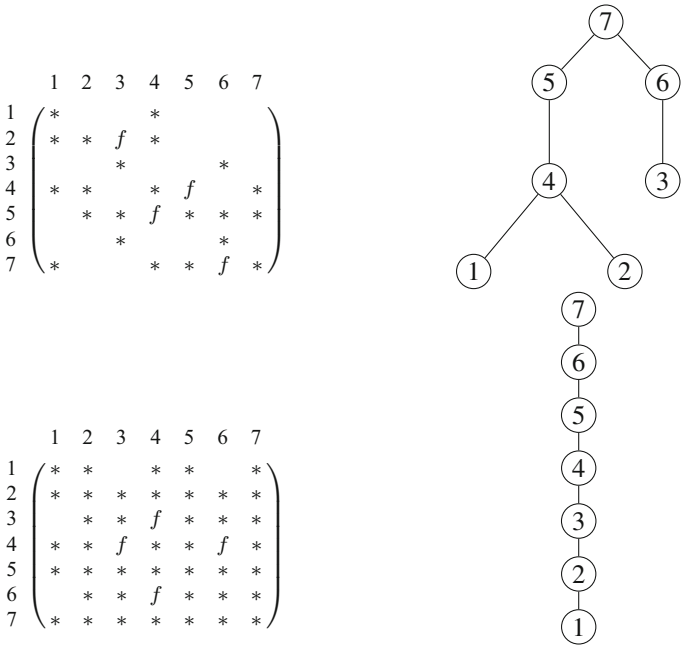

Figure 6.8 The sparsity patterns of A and $L+U$ (top) and of A^TA and $\widehat{L}+\widehat{L}^T$, where $A^TA = \widehat{L}\widehat{L}^T$ (bottom). Filled entries are denoted by f. The corresponding elimination trees are also given.

$$\mathcal{S}\{L+U\} \subseteq \mathcal{S}\{\widehat{L}+\widehat{L}^T\}.$$

An important feature of Theorem 6.13 is that it holds for *any* row permutation matrix P applied to A. This allows partial pivoting (Section 3.1.2) to be used. The following result states that $\mathcal{T}(A^TA)$ represents the potential dependencies among the columns in an LU factorization and that for strong Hall matrices no tighter prediction is possible from the sparsity structure of A.

Theorem 6.14 (Gilbert & Ng 1993) *If $PA = LU$ is any factorization of A with partial pivoting, then the following hold.*

1. *If vertex i is an ancestor of vertex j in $\mathcal{T}(A^TA)$, then $i > j$.*
2. *If $l_{ij} \neq 0$, $i \neq j$, then vertex i is an ancestor of vertex j in $\mathcal{T}(A^TA)$.*
3. *If $u_{ij} \neq 0$, $i \neq j$, then vertex j is an ancestor of vertex i in $\mathcal{T}(A^TA)$.*
4. *Suppose in addition that A is a strong Hall matrix. If $l = parent(k)$ in $\mathcal{T}(A^TA)$, then there are values of the nonzero entries of A for which $u_{kl} \neq 0$.*

Figure 6.8 illustrates the differences in the sparsity patterns of A and A^TA and of their factors; the corresponding elimination trees are also given. This reveals a potential problem with the column elimination tree: $\mathcal{S}\{A^TA\}$ can have significantly more entries than $\mathcal{S}\{L+U\}$. An extreme example is when A has one or more dense rows because A^TA is then fully dense.

6.1.6 Supernodes in LU Factorizations

Supernodes group together columns of the factors with the same nonzero structure, allowing them to be treated as a dense submatrix for storage and computation. When solving SPD systems, supernodes can be determined during the symbolic phase. For nonsymmetric matrices, supernodes are harder to characterize. The need to incorporate pivoting means it may not be possible to predict the sparsity structures of the factors before the numerical factorization and they must be identified on-the-fly. While there are several possible ways to define supernodes, the simplest (which is widely used in practice) follows the symmetric case and defines a supernode to be a set of contiguously numbered columns of L with the triangular diagonal block treated as dense and the columns as having the same structure below the diagonal block.

In a Cholesky solver, fundamental supernodes (Section 4.6.1) are made contiguous by symmetrically permuting the matrix according to a postordering of its elimination tree; this does not change the sparsity of the Cholesky factor. For nonsymmetric A, before the numerical factorization, $\mathcal{T}(A^TA)$ can be constructed and the columns of A then permuted according to its postordering to bring together supernodes. The following result extends Theorem 4.9.

Theorem 6.15 (Li 1996) *Let A have column elimination tree $\mathcal{T}(A^TA)$. Let p be a permutation vector such that if p_i is an ancestor of p_j in $\mathcal{T}(A^TA)$, then $i > j$. Let P be the permutation matrix corresponding to p and let $\widehat{A} = PAP^T$. Then $\mathcal{T}(\widehat{A}^T\widehat{A})$ is isomorphic to $\mathcal{T}(A^TA)$; in particular, relabelling each vertex i of $\mathcal{T}(\widehat{A}^T\widehat{A})$ as p_i yields $\mathcal{T}(A^TA)$. If, in addition, $\widehat{A} = \widehat{L}\widehat{U}$ is an LU factorization without pivoting then $P^T\widehat{L}P$ and $P^T\widehat{U}P$ are lower triangular and upper triangular matrices, respectively, so that $A = (P^T\widehat{L}P)(P^T\widehat{U}P)$ is also an LU factorization.*

In practice, for many matrices the average size of a supernode is only 2 or 3 columns and many comprise a single column. Larger artificial supernodes may be created by merging vertex j with its parent vertex i in $\mathcal{T}(A^TA)$ if the subtree rooted at i has fewer than some chosen number of vertices.

6.2 LU Multifrontal Method

The multifrontal method (Section 5.4) can be generalized to nonsymmetric A by modifying the definitions of the frontal matrices and generated elements to conform to an LU factorization. But natural generalizations to rectangular frontal and generated element matrices do not simultaneously satisfy the statements from Observation 5.1. These statements can be rewritten for the LU factorization as follows.

(a) Each generated element V_j is used only once to contribute to a frontal matrix.

(b) The row and column index lists for the rectangular frontal matrix F_j correspond to the nonzeros in column $L_{j:n,j}$ and nonzeros in row $U_{j,j:n}$, respectively.

These conditions cannot both hold. An approach that satisfies (a) can be based on the sparsity pattern of $\mathcal{S}\{A + A^T\}$ and storing some explicit zeros if $\mathcal{S}\{A\}$ is not symmetric. It is then analogous to the symmetric multifrontal method. In this case, although the frontal and generated elements may be numerically nonsymmetric, they are square and structurally symmetric. This approach performs well if $\mathcal{S}\{A\}$ is close to symmetric, that is, the symmetry index of A is close to unity.

An approach that satisfies (b) and not necessarily (a) splits the generated elements into smaller ones that are embedded into further rectangular frontal matrices. We illustrate this using the example from Figure 6.7, that is,

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\
\begin{pmatrix}
* & & * & & & & & & & \\
* & * & f & & * & & & * & & * \\
* & & * & * & & & & & & \\
& & * & * & & & & & * & * \\
& & & * & * & * & & * & f & f \\
& * & f & f & f & * & & f & f & f \\
& & & * & & & * & & f & f \\
* & * & f & f & f & f & & * & f & f \\
& & & & & & & & * & * \\
& & & & * & f & * & f & f & *
\end{pmatrix}
\end{array},$$

where $*$ are entries in A and filled entries in $L + U$ are denoted by f. Taking the entries in the first row and column, the sparsity patterns of the first frontal matrix and the corresponding generated element are

$$F_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cc} 1 & 3 \end{array} \\
\begin{pmatrix} * & * \\ * & \\ * & \\ * & \end{pmatrix}
\end{array}, \quad V_1 = \begin{array}{c} \\ 2 \\ 3 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{c} 3 \end{array} \\
\begin{pmatrix} f \\ * \\ f \end{pmatrix}
\end{array}.$$

To construct F_2 that satisfies (b) we can only use part of V_1. From the row and column replication principles, because $a_{13} \neq 0$, the sparsity pattern of column 1 is replicated in that of column 3 of the factors. While the entry in position $(2, 3)$ belongs to F_2, because of the row replication of the sparsity pattern of the first row in that of the second row, the remaining entries contribute to F_3 and so we split V_1 into two as follows

$$V_1^2 = \begin{array}{c} 2 \\ \\ \\ \\ \\ \\ \\ \end{array}
\begin{array}{c}
\begin{array}{c} 3 \end{array} \\
\begin{pmatrix} f \end{pmatrix}
\end{array}, \quad V_1^3 = \begin{array}{c} 3 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{c} 3 \end{array} \\
\begin{pmatrix} * \\ f \end{pmatrix}
\end{array}, \quad V_1 = V_1^2 \Updownarrow V_1^3,$$

where $\boxplus$ is the extend-add operator and V_1^2 and V_1^3 contribute to F_2 and F_3, respectively. Then F_2 and the corresponding generated element V_2 are

$$F_2 = \begin{matrix} \\ 2 \\ 6 \\ 8 \end{matrix} \begin{matrix} \begin{matrix} 2 & 5 & 8 & 10 \end{matrix} \\ \begin{pmatrix} * & * & * & * \\ * & & & \\ * & & & \end{pmatrix} \end{matrix} \boxplus V_1^2 = \begin{matrix} \\ 2 \\ 6 \\ 8 \end{matrix} \begin{matrix} \begin{matrix} 2 & 3 & 5 & 8 & 10 \end{matrix} \\ \begin{pmatrix} * & f & * & * & * \\ * & & & & \\ * & & & & \end{pmatrix} \end{matrix}, \quad V_2 = \begin{matrix} \\ 6 \\ 8 \end{matrix} \begin{matrix} \begin{matrix} 3 & 5 & 8 & 10 \end{matrix} \\ \begin{pmatrix} f & f & f & f \\ f & f & * & f \end{pmatrix} \end{matrix}.$$

Consider the following splitting of V_2

$$V_2 = \begin{matrix} \\ 6 \\ 8 \end{matrix} \begin{matrix} 3 \\ \begin{pmatrix} f \\ f \end{pmatrix} \end{matrix} \boxplus \begin{matrix} \\ 6 \\ 8 \end{matrix} \begin{matrix} 5 \\ \begin{pmatrix} f \\ f \end{pmatrix} \end{matrix} \boxplus \begin{matrix} \\ 6 \\ 8 \end{matrix} \begin{matrix} 8 & 10 \\ \begin{pmatrix} f & f \\ * & f \end{pmatrix} \end{matrix} \equiv V_2^3 \boxplus V_2^5 \boxplus V_2^6.$$

The next frontal matrix is

$$F_3 = \begin{matrix} \\ 3 \\ 4 \end{matrix} \begin{matrix} 3 & 4 \\ \begin{pmatrix} * & * \\ * & * \end{pmatrix} \end{matrix} \boxplus V_1^3 \boxplus V_2^3 = \begin{matrix} \\ 3 \\ 4 \\ 6 \\ 8 \end{matrix} \begin{matrix} 3 & 4 \\ \begin{pmatrix} * & * \\ * & * \\ f & \\ f & \end{pmatrix} \end{matrix}, \quad V_3 = \begin{matrix} \\ 4 \\ 6 \\ 8 \end{matrix} \begin{matrix} 4 \\ \begin{pmatrix} * \\ f \\ f \end{pmatrix} \end{matrix}.$$

The subsequent steps can be described in a similar way.

Theorem 6.16 expresses the nested relationship between the nonsymmetric multifrontal method and the nonsymmetric elimination tree.

Theorem 6.16 (Eisenstat & Liu 2005b) *Assume A is a general nonsymmetric matrix and $t = parent(k)$ in $\mathcal{T}(A)$. Then*

$$\mathcal{S}\{L_{t:n,k}\} \subseteq \mathcal{S}\{L_{t:n,t}\} \quad \textit{and} \quad \mathcal{S}\{U_{k,t:n}\} \subseteq \mathcal{S}\{U_{t,t:n}\}.$$

Proof Because t is the parent of k, by definition $t \overset{\mathcal{G}(L)}{\Longrightarrow} k \overset{\mathcal{G}(U)}{\Longrightarrow} t$. If $u_{ij} \neq 0$, then a multiple of column i is added to column j during the LU factorization. Thus, by a simple induction argument, for each j on the path $k \overset{\mathcal{G}(U)}{\Longrightarrow} t$, we must have $\mathcal{S}\{L_{j:n,k}\} \subseteq \mathcal{S}\{L_{j:n,j}\}$. In particular, this holds for column t. The second part follows by a similar argument using the path $t \overset{\mathcal{G}(L)}{\Longrightarrow} k$. □

This result shows that the parent relationship in the nonsymmetric elimination tree guarantees that both row and column replications can be applied at the same time. Hence all entries of the submatrices of the generated element V_k with indices greater than or equal to $parent(k)$ can be added to $V_{parent(k)}$ using the operation $\boxplus$. To illustrate this, consider again the 10×10 example above for which $parent(1) = 3$. Theorem 6.16 guarantees that V_1 can be embedded into F_3 because $\mathcal{S}\{L_{3:n,1}\} \subseteq \mathcal{S}\{L_{3:n,3}\}$ and $\mathcal{S}\{U_{1,3:n}\} \subseteq \mathcal{S}\{U_{3,3:n}\}$.

6.3 Preprocessing Sparse Matrices

We now turn our attention to preprocessing techniques that can help in computing an LU factorization. In particular, we consider when A does not have a full transversal (that is, it has one or more zeros on the diagonal). For numerical stability and to reduce the number of permutations required during the factorization, it can be useful to permute A before the factorization begins to put large nonzero entries on the diagonal. As an example, consider the matrix A in Figure 6.9. It has $a_{22} = 0$ and we want to know whether it can be permuted so that all the diagonal entries are nonzero. This question and its answer can be formulated in terms of matchings and bipartite graphs.

6.3.1 Bipartite Graphs and Matchings

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an edge subset $\mathcal{M} \subseteq \mathcal{E}$ is called a **matching** (or assignment) if no two edges in $\mathcal{M}$ are incident to the same vertex. In matrix terms, a matching corresponds to a set of nonzero entries with no two belonging to the same row or column. A vertex is matched if there is an edge in the matching incident on the vertex, and is unmatched (or free) otherwise. The **cardinality** of a matching is the number of edges in it. A **maximum cardinality matching** (or **maximum matching**) is a matching of maximum cardinality. A matching is **perfect** if all the vertices are matched.

A **bipartite graph** is an undirected graph whose vertices can be partitioned into two disjoint sets such that no two vertices within the same set are adjacent, that is, each set is an **independent set**. Let the $n \times n$ matrix A have entries $\{a_{ij'}\}$. Associated with A is a bipartite graph defined as a triple $\mathcal{G}_b = (\mathcal{V}_{row}, \mathcal{V}_{col}, \mathcal{E})$, where the row vertex set $\mathcal{V}_{row} = \{i \,|a_{ij'} \neq 0\}$ and the column vertex set $\mathcal{V}_{col} = \{j' \,|a_{ij'} \neq 0\}$ correspond to the rows and columns of A and there is an (undirected) edge $(i, j') \in \mathcal{E}$ if and only if $a_{ij'} \neq 0$. This is illustrated in Figure 6.9. We use prime to distinguish between the independent set of row vertices and the independent set of column vertices, that is, i denotes a row vertex and i' denotes a column vertex.

If A is structurally nonsingular, a matching $\mathcal{M}$ in $\mathcal{G}_b$ is perfect if it has cardinality n. A perfect matching defines an $n \times n$ permutation matrix Q with entries q_{ij} given by

$$q_{ij} = \begin{cases} 1, & \text{if } (j, i') \in \mathcal{M}, \\ 0, & \text{otherwise.} \end{cases}$$

Both QA and AQ have the matching entries on the (zero-free) diagonal. Q and the column permuted matrix AQ for the example in Figure 6.9 are given in Figure 6.10.

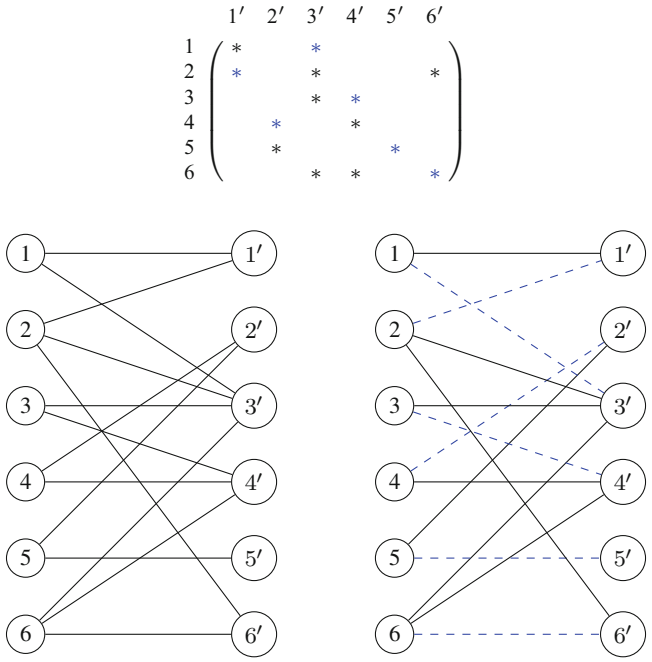

Figure 6.9 A sparse matrix and its bipartite graph $\mathcal{G}_b$ (left). The matched matrix entries are in blue and edges that belong to a perfect matching in $\mathcal{G}_b$ are given by the blue dashed lines (right). Note that the perfect matching is not unique (an alternative is in Figure 6.11).

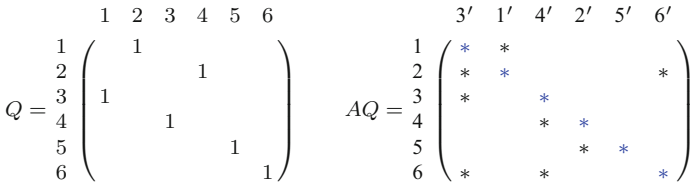

Figure 6.10 The permutation matrix Q and the column permuted matrix AQ corresponding to the matrix in Figure 6.9. The matched entries are on the diagonal of AQ.

6.3.2 *Augmenting Paths*

If a perfect matching exists, it can be found using augmenting paths. A path $\mathcal{P}$ in a graph is an ordered set of edges in which successive edges are incident to the same vertex. $\mathcal{P}$ is called an $\mathcal{M}$**-alternating path** if the edges of $\mathcal{P}$ are alternately in $\mathcal{M}$ and not in $\mathcal{M}$. An $\mathcal{M}$-alternating path is an $\mathcal{M}$**-augmenting path** in $\mathcal{G}_b$ if it connects an unmatched column vertex with an unmatched row vertex. Note that the length of an $\mathcal{M}$-augmenting path is an odd integer.

ALGORITHM 6.3 Maximum matching algorithm
Input: An undirected graph.
Output: Output maximum matching.

1: Find an initial matching $\mathcal{M}$ ▷ For example, $\mathcal{M} = \emptyset$
2: **while** there exists a $\mathcal{M}$-augmenting path $\mathcal{P}$ **do**
3: $\mathcal{M} = \mathcal{M} \oplus \mathcal{P}$ ▷ Augment the matching along $\mathcal{P}$
4: **end while**

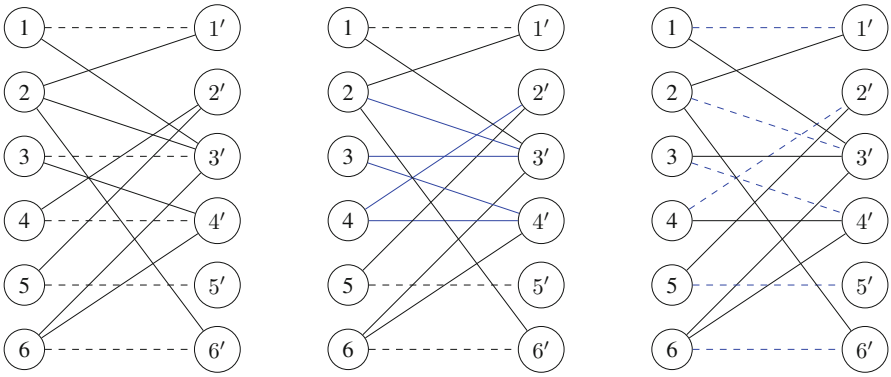

Figure 6.11 An illustration of the search for a perfect matching using augmenting paths. On the left, the dashed lines represent a matching with cardinality 5. In the centre, the blue line is an augmenting path with end vertices 2 and $2'$. On the right is the perfect matching with cardinality 6 that is obtained using the augmenting path.

Let $\mathcal{M}$ and $\mathcal{P}$ be subsets of $\mathcal{E}$ and define the symmetric difference

$$\mathcal{M} \oplus \mathcal{P} := (\mathcal{M} \setminus \mathcal{P}) \cup (\mathcal{P} \setminus \mathcal{M}),$$

that is, the set of edges that belongs to either $\mathcal{M}$ or $\mathcal{P}$ but not to both. If $\mathcal{M}$ is a matching and $\mathcal{P}$ is an $\mathcal{M}$-augmenting path, then $\mathcal{M} \oplus \mathcal{P}$ is a matching with cardinality $|\mathcal{M}|+1$. Growing the matching in this way is called augmenting along $\mathcal{P}$. The next result shows that augmenting paths can be used to find a maximum matching (Algorithm 6.3).

Theorem 6.17 (Berge 1957) *A matching $\mathcal{M}$ in an undirected graph is a maximum matching if and only if there is no $\mathcal{M}$-augmenting path*

Figure 6.11 demonstrates the procedure. On the left is a bipartite graph with a matching with cardinality 5. In the centre, an augmenting path $2 \Longrightarrow 3' \Longrightarrow 3 \Longrightarrow 4' \Longrightarrow 4 \Longrightarrow 2'$ is shown. Augmenting the matching along this path, the cardinality of the matching increases to 6 and $\mathcal{M} \oplus \mathcal{P}$ is a perfect matching.

6.3.3 Weighted Matchings

While the maximum matching algorithm finds a permutation of A such that the permuted matrix has nonzero diagonal entries, there are more sophisticated variations that aim to ensure the absolute values of the diagonal entries of the permuted matrix (or their product) are in some sense large. This can increase the likelihood that the permuted matrix is strongly regular and reduce the need for partial pivoting during the LU factorization. The core problem is as follows: given an $n \times n$ matrix A, find a matching of the rows to the columns such that the product of the matched entries is maximized. That is, find a permutation vector q that maximizes

$$\prod_{i=1}^{n} |a_{iq_i}|. \tag{6.5}$$

Define a matrix C corresponding to A with entries $c_{ij'} \geq 0$ as follows:

$$c_{ij'} = \begin{cases} \log(\max_i |a_{ij'}|) - \log |a_{ij'}|, & \text{if } a_{ij'} \neq 0 \\ \infty, & \text{otherwise.} \end{cases} \tag{6.6}$$

It is straightforward to see that finding a q that solves (6.5) is equivalent to finding a q that minimizes

$$\sum_{i=1}^{n} |c_{iq_i}|, \tag{6.7}$$

which is equivalent to finding a minimum weight perfect matching in an edge weighted bipartite graph. This is a well-studied problem and is known as the bipartite weighted matching or linear sum assignment problem.

If $\mathcal{G}_b = (\mathcal{V}_{row}, \mathcal{V}_{col}, \mathcal{E})$ is the bipartite graph associated with A then let $\mathcal{G}_b(C) = (\mathcal{V}_{row}, \mathcal{V}_{col}, \mathcal{E})$ be the corresponding weighted bipartite graph in which each edge $(i, j') \in \mathcal{E}$ has a weight $c_{ij'} \geq 0$. The weight (or cost) of a matching $\mathcal{M}$ in $\mathcal{G}_b(C)$, denoted by $csum(\mathcal{M})$, is the sum of its edge weights; i.e.

$$csum(\mathcal{M}) = \sum_{(i,j') \in \mathcal{M}} c_{ij'}.$$

A perfect matching $\mathcal{M}$ in $\mathcal{G}_b(C)$ is said to be a **minimum weight perfect matching** if it has smallest possible weight, i.e. $csum(\mathcal{M}) \leq csum(\widehat{\mathcal{M}})$ for all possible perfect matchings $\widehat{\mathcal{M}}$.

The key concept for finding a minimum weight perfect matching is that of a **shortest augmenting path**. An $\mathcal{M}$-augmenting path $\mathcal{P}$ starting at an unmatched column vertex is called **shortest** if

$$csum(\mathcal{M} \oplus \mathcal{P}) \leq csum(\mathcal{M} \oplus \widehat{\mathcal{P}})$$

for all other possible $\mathcal{M}$-augmenting paths $\widehat{\mathcal{P}}$ starting at the same column vertex. A matching $\mathcal{M}_e$ is **extreme** if and only if there exist u_i and $v_{j'}$ (which are termed **dual variables**) satisfying

$$\begin{cases} c_{ij'} = u_i + v_{j'}, & \text{if } (i, j') \in \mathcal{M}_e, \\ c_{ij'} \geq u_i + v_{j'}, & \text{otherwise.} \end{cases} \tag{6.8}$$

This is employed by the `MC64` algorithm. The dual variables will be important when we discuss scaling sparse matrices in Section 7.4.2. The `MC64` algorithm is outlined here as Algorithm 6.4. It starts with a feasible solution and corresponding extreme matching and then proceeds to iteratively increase its cardinality by one by constructing a sequence of shortest augmenting paths until a perfect extreme matching is found. The algorithm can be made more efficient if a large initial extreme matching can be found. For example, Step 3 can be replaced by setting $u_i = \min\{c_{ij'} | \, j' \in \mathcal{S}\{A_{i,1:n}\}\}$ for $i \in \mathcal{V}_{row}$ and $v_{j'} = \min\{c_{ij'} - u_i | \, i \in \mathcal{S}\{A_{1:n,j'}\}\}$ for $j' \in \mathcal{V}_{col}$. In Step 4, an initial extreme matching can be determined from the edges for which $c_{ij'} - u_i - v_{j'} = 0$.

There are a number of potential problems with the `MC64` algorithm. First, the runtime is hard to predict and depends on the initial ordering of A. Second, it is a serial algorithm and as such it can represent a significant fraction of the total factorization time of a direct solver. Because the complexity of Step 6 of Algorithm 6.4 is $O((n + nz(A)) \log n)$ and the complexity of Step 7 is $O(n)$ and of Step 8 is $O(n+nz(A)$, `MC64` has a worst-case complexity of $O(n(n+nz(A)) \log n)$. In practice, this bound is not achieved and the algorithm is widely used.

ALGORITHM 6.4 Outline of the `MC64` algorithm
Input: Matrix A.
Output: A matching $\mathcal{M}$ and dual variables u_i, $v_{j'}$.

1: Define the weights $c_{ij'}$ using (6.6)
2: Construct the weighted bipartite graph $\mathcal{G}_b(C) = (\mathcal{V}_{row}, \mathcal{V}_{col}, \mathcal{E})$
3: Set $u_i = 0$ for $i \in \mathcal{V}_{row}$ and $v_{j'} = \min\{c_{ij'} : (i, j') \in \mathcal{E}\}$ for $j' \in \mathcal{V}_{col}$ ▷ Initial solution
4: Set $\mathcal{M} = \{(i, j') | \, u_i + v_{j'}\}$ ▷ Initial extreme matching
5: **while** $\mathcal{M}$ is not perfect **do**
6: Find the shortest augmenting path $\mathcal{P}$ with respect to $\mathcal{M}$
7: Augment the matching $\mathcal{M} = \mathcal{M} \oplus \mathcal{P}$
8: Update u_i, $v_{j'}$ so that (6.8) is satisfied for new $\mathcal{M}$ ▷ Make $\mathcal{M}$ extreme
9: **end while**

6.3.4 Dulmage-Mendelsohn Decompositions

The importance of preordering A to block triangular form was discussed in Section 3.4. The **Dulmage-Mendelsohn decomposition** is based on matchings and is a generalization of the block triangular form. It provides a precise characterization of structurally rank deficient matrices and it can be used to reduce the work required for an LU factorization. It comprises row and column permutations P and Q such that

$$PAQ = \begin{array}{c} \\ \mathcal{R}_1 \\ \mathcal{R}_2 \\ \mathcal{R}_3 \end{array} \begin{array}{c} \begin{array}{ccc} \mathcal{C}_1 & \mathcal{C}_2 & \mathcal{C}_3 \end{array} \\ \begin{pmatrix} A_1 & A_4 & A_6 \\ 0 & A_2 & A_5 \\ 0 & 0 & A_3 \end{pmatrix} \end{array}. \tag{6.9}$$

Here A_1 is an $m_1 \times n_1$ underdetermined matrix ($m_1 < n_1$ or $m_1 = n_1 = 0$), A_2 is an $m_2 \times m_2$ square matrix and A_3 is an $m_3 \times n_3$ overdetermined matrix ($m_3 > n_3$ or $m_3 = n_3 = 0$). It can be shown that A_1^T and A_3 are strong Hall matrices but A_2 need not be a strong Hall matrix, in which case it can be permuted to block upper triangular form.

If row and column sets $\mathcal{R}$ and $\mathcal{C}$ form a maximum matching of A, then $\mathcal{R}_1$ and $\mathcal{R}_2$ are subsets of $\mathcal{R}$ and $|\mathcal{R}_3 \cap \mathcal{R}| = n_3$, and $\mathcal{C}_2$ and $\mathcal{C}_3$ are subsets of $\mathcal{C}$ and $|\mathcal{C}_1 \cap \mathcal{C}| = m_1$. An example decomposition for a 10×10 matrix is given in Figure 6.12. Here $\mathcal{R} = \{1, 2, \ldots, 9\}$ and $\mathcal{C} = \{2, 3, \ldots, 10\}$.

The **coarse Dulmage-Mendelsohn decomposition** orders the unmatched columns as the first columns in PAQ and orders the unmatched rows as the last rows in PAQ. If A is square and has a perfect matching then its coarse decomposition has only the matrix A_2; otherwise, both A_1 and A_3 are present. The coarse decomposition is computed by first finding a maximum matching. Assuming it is not a perfect matching, the rows in A_1 are determined by performing depth-first searches from the unmatched columns to find all of the row vertices that

$PAQ =$

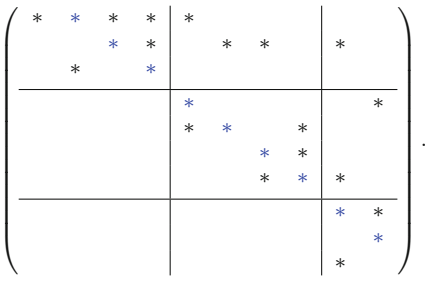

Figure 6.12 An example of a coarse Dulmage-Mendelsohn decomposition. The blue entries belong to the maximum matching. $m_1 = 3$, $m_2 = 4$, $m_3 = 3$, $n_1 = 4$, $n_2 = 4$, $n_3 = 2$. Column 1 and row 10 are unmatched.

are reachable from the unmatched columns via alternating augmenting paths. The columns in A_1 are defined to be the union of the set of unmatched columns and the set of columns matched with the rows in A_1. Similarly, the columns in A_3 are determined by performing depth-first searches from the unmatched rows to find all of the column vertices that are reachable from the unmatched rows via alternating augmenting paths. The rows in A_3 are defined to be the union of the set of rows matched to the columns in A_3 and the set of unmatched rows.

It may be possible to further permute the matrix to obtain the **fine Dulmage-Mendelsohn decomposition**. The fine Dulmage-Mendelsohn decomposition computes P and Q such that A_1 and A_3 are block diagonal matrices in which each diagonal block is irreducible, and A_2 is block upper triangular with strongly connected (square) diagonal blocks. Once the coarse decomposition has been computed, A_1 and A_3 are searched to find any irreducible blocks and the permutations required to place these on the diagonals of A_1 and A_3 are computed. Finally, following Section 3.4, strongly connected components in A_2 are found and a permutation is formed to reduce A_2 to block upper triangular form (with the strongly connected components lying on the diagonal). If A is reducible and nonsingular, the fine Dulmage-Mendelsohn decomposition can be used to solve the linear system $Ax = b$ using block back-substitution.

6.4 Notes and References

Early theoretical results related to sparse LU factorizations can be found in Rose & Tarjan (1978), which extends the systematic understanding of the symbolic elimination introduced in Rose et al. (1976). A key paper that influenced the discussion and development of both the theory and algorithms for predicting sparsity structures in LU factorizations is Gilbert (1994) (first available in 1986 as a Cornell technical report). As the primary and still very useful resource on transitive reduction, we refer to Aho et al. (1972); Gilbert & Liu (1993) extend the concept of an elimination tree to study sparse LU factorizations of nonsymmetric matrices and present theoretical concepts based on DAGs; see also the parallel counterpart in Grigori et al. (2007). Ways to simplify symbolic factorizations and prune DAGs are discussed in Eisenstat & Liu (1992, 1993a). An elegant treatment of both the theoretical and practical aspects of LU factorizations based on DAGs and the nonsymmetric elimination tree (including pruning and pivoting) is given in a series of three papers by Eisenstat & Liu (2005a,b, 2007).

Partial pivoting within the sparse column LU factorization is introduced in Gilbert & Peierls (1988). This paper influenced not only further developments in sparse LU factorizations but also the development of incomplete factorizations. Partial pivoting based on the column elimination tree is first discussed in George & Ng (1985); see also Gilbert & Ng (1993) and Li (1996) for further use of column elimination trees. Further research on exactness of structural predictions is presented by Grigori et al. (2009).

The proof of Theorem 6.17 is given by Berge (1957) but the result was observed earlier (for example, König (1931)). Preordering nonsymmetric matrices using matching algorithms is explained in Duff & Koster (1999, 2001). It is based on the Hungarian algorithm of Kuhn (1955) and a sparse variant of the shortest path algorithm of Dijkstra (1959). Duff and Koster implemented their algorithm in the widely used software package `MC64`. Because `MC64` can be expensive to run, there has been interest in developing efficient parallel algorithms for finding a perfect matching in a weighted bipartite graph (Azad et al., 2020) and also in relaxing the optimality requirement to allow the development of cheaper algorithms that can be parallelised; see, for example, Hogg & Scott (2015). A classical paper that describes the Dulmage-Mendelsohn decomposition is Pothen & Fan (1990).

The development of supernodal LU factorizations is closely connected with that of column LU factorizations. A key paper is by Demmel et al. (1999), in which different types of supernodes for nonsymmetric matrices are considered.

Duff & Reid (1984) describe a symmetric-pattern multifrontal algorithm for nonsymmetric matrices that generates an assembly tree based on the structure of $A+A^T$. This employs square frontal matrices and can incur a substantial overhead for highly nonsymmetric matrices because of unnecessary data dependencies in the assembly tree and extra explicit zeros in the artificially symmetrized frontal matrices. Davis & Duff (1997) introduce an nonsymmetric-pattern multifrontal algorithm that seeks to overcome these deficiencies by using rectangular frontal matrices. This work later developed into the package UMFPACK of Davis (2004), while Amestoy & Puglisi (2002) propose an nonsymmetric version of the multifrontal method that can be regarded as being intermediate between the nonsymmetric-pattern variant of UMFPACK and the symmetric-pattern multifrontal method. The Watson Sparse Matrix Package (WSMP, 2020) also uses a nonsymmetric multifrontal algorithm.

Notable early sparse LU solvers were the Yale Sparse Matrix Package (YSMP) of Eisenstat et al. (1977) and the Harwell Subroutine Library code `MA28` written by Duff (1980), followed later by `MA48` of Duff & Reid (1996). These codes address important practical considerations (for serial computations). Furthermore, the right-looking Markowitz packages `MA28` and `MA48`, which are designed particularly for highly nonsymmetric matrices, combine the symbolic and numerical factorization phases into a single analyse-factorize phase. Contemporary software packages such as PARDISO (2022), SuperLU (Li et al., 1999), UMFPACK and WSMP have been developed over many years. They provide one of the best ways of understanding the practical value of the ideas presented in research papers and technical reports. PARDISO combines left and right-looking updates in a parallel shared-memory code that assumes a symmetric nonzero sparsity pattern. SuperLU offers a left-looking supernodal variant for sequential machines, SuperLU_MT for shared-memory parallel machines, and the right-looking supernodal SuperLU_DIST (Li & Demmel, 2003) for highly parallel distributed memory hybrid systems. Demmel et al. (1999) and Li (2008) describe the algorithms and performance on various machines. The WSMP software is split into a serial and multithreaded single-process library for use on a single core or multiple cores on a shared-memory machine, and a separate library for distributed memory environments.