Combinatorial Problems in Matrix Computation

Anders Edenbrandt

TR 85-695
July 1985
(Ph. D. Thesis)

Department of Computer Science
Cornell University
Ithaca, NY 14853

COMBINATORIAL PROBLEMS IN MATRIX COMPUTATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Anders Gunnar Edenbrandt

August, 1985

Acknowledgements

Thanks first of all to my advisor, John Gilbert, with whom it has been a pleasure to collaborate and who has been the source of inspiration for much of the material in this work. John has also taught me a lot about writing, and his careful reading of my drafts has contributed greatly to the readability of this thesis.

Thanks to David Gries and Kenneth Brown for serving on my committee, and to Tom Coleman for many helpful ideas. Sten Henriksson and Bengt Aspvall both encouraged my endeavor and put me on the right track: Sten in Lund, and Bengt at Cornell.

I am grateful to my parents, Barbro and Gunnar Edenbrandt, and to my brother Lars, for their enthusiasm and interest in my life at Cornell. A special thanks goes to my brother, Carl-Magnus Edenbrandt, who has always supported me and helped me regain my self-confidence in times of need. Finally, I want to thank Aliza Wechsler for her unfailing love and trust, and for patiently listening to and critisizing my unpolished ideas.

I also wish to express my gratitude for the generous financial support provided by graduate fellowships from the University of Lund and from IBM. This work was also supported in part by National Science Foundation grants MCS 81-01220 and MCS 82-02948.

Contents

List of Figures

Chapter 1

Introduction

A fundamental task in numerical computing, with applications in almost any area of science, is the solution of a system of linear equations. This problem is often cast in terms of matrices: the statement of the problem is written

$$A\mathbf{x} = \mathbf{b}$$

and the way to a solution of this system of equations involves factoring A into two triangular matrices L and U, satisfying $A = LU$.

Polynomial time algorithms for this task are many and well-known; typically they have a running time of $O(n^3)$, for A an $n \times n$ matrix. A running time of this magnitude may not seem like a very high price to pay, considering that the size of A is really $\Theta(n^2)$. But for many classes of problems encountered in practice the real size of A is considerably smaller; in fact, A may have as few as $3n$ or $5n$ nonzero entries. *Sparseness* becomes a keyword; systems of 1000 unknowns and equations become feasible only if the matrix is very sparse. One would hope that this sparsity of A could be exploited in the factoring process, to give algorithms with a running time lower than $O(n^3)$. The wide use that such an algorithm would find is evident from the broad range of applications of large sparse systems.

It is at this point that what seems to be a purely numerical problem becomes a combinatorial one. We can break it down into two major parts: the first is to correctly compute the nonzero structure of U, the factor of A; the second part is to find a reordering of the columns and rows of A that will give the sparsest possible structure for U. The importance of the second task is obvious: if we fail to produce a sparse factor U, then all the advantages of A's sparseness have been lost. But the first task is equally important: if we don't have a method for correctly computing the structure of U, then our efforts to make U sparse may be wasted.

The first problem, that of correctly computing the structure of A's factor, has received little attention [3], but is of great importance—especially in the case of QR factorization, where the straightforward algorithms often predict a much too large structure for the factor [6]. In Chapter 3 we introduce the notion of correct structure prediction and establish some basic results. It turns out that the straightforward algorithm for structure prediction under LU-decomposition works well on a large class of matrices. For QR factorization, however, many commonly suggested algorithms often exaggerate the amount of fill in the factor. A class of structures having the so called "strong Hall property" is singled out as one for which the common algorithms work well.

The greater part of results in this area deal with the second problem, that of minimizing fill in the factor U. An early contribution here was made by Parter [47], and since then important work has been done by Rose, Tarjan, Gilbert, Yannakakis, and many others [49,50,51,52,23,24,57]. The basic problem of minimizing fill is NP-complete [51,24,57], although in its more general form the complexity of the problem remains an open question. This is discussed in Chapter 4, together with an examination of some common heuristics for the problem.

In Chapter 5 is established the NP-completeness of minimizing fill in QR factorization, together with an algorithm for QR factorization that avoids the problem of predicting a too large structure for the factored matrix.

Finally, Chapter 6 contains some results on chordal graphs—a class of graphs of special interest in this context, since they represent matrix structures that can be factored without any filling in of zeros with nonzeros. Chordal graphs are shown to have separators of size $O(\sqrt{m})$ that can be found in linear time. And chordal graph recognition is shown to be in NC, the class of problems with fast ($O(\log^k n)$) parallel algorithms.

Chapter 2

The nonzero structure of a matrix

2.1 Abstract representation

Let A be an $m \times n$ real-valued matrix. The nonzero structure of A is a combinatorial concept that can be represented in many different ways. We will denote the nonzero structure of a matrix by putting a tilde above it: $\tilde{A}$ denotes the nonzero structure of A. It will also be useful to be able to talk about substructures. We say that $\tilde{A}$ is a *substructure* of $\tilde{B}$ if A and B are matrices of the same dimensions and $A_{ij} \neq 0$ implies that $B_{ij} \neq 0$. This is also written as $\tilde{A} \subseteq \tilde{B}$.

The following two conditions must be met by any structural representation:

1. The representation, S, is a set of cardinality $\aleph_0$.
2. With each matrix is associated an element r from the set S, such that given r, for any i, j, it is decidable whether A_{ij} is nonzero.

The first of these requirements states that we are interested only in representatives chosen from a discrete collection of objects. This disqualifies the matrix itself as a representative of its nonzero structure. The second requirement is the bare minimum of what we must be able to deduce from a representative.

$$\begin{bmatrix} 12.5 & 0.0 & 0.29 \\ 0.0 & 35.19 & 0.0 \\ 0.0 & 6.76 & 4.2 \\ 1.0 & 0.0 & 0.0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Real-valued matrix | Boolean matrix

Figure 2.1: Boolean matrix representation of a matrix structure.

With these as our only requirements for a representation, there remain numerous possible representations. Some representations are better than others. For example, we may be interested in representations that take up small space; or representations on which we can perform certain operations efficiently; or representations that facilitate reasoning about structures. In the rest of this chapter are defined and discussed briefly some useful representations.

2.2 Boolean matrices

A Boolean matrix is a matrix of only zeros and ones. Such a matrix can be used to represent the nonzero structure of an arbitrary matrix.

If A is an m by n real-valued matrix, let $B(A)$ be an m by n Boolean matrix. For each nonzero entry in the matrix put a 1 at the corresponding entry in the Boolean matrix; and for each zero in the matrix put a zero at the corresponding entry in the Boolean matrix. See Figure 2.1. Matrices of arbitrary dimensions and structures can have their nonzero structure represented by a Boolean matrix.

2.3 Graphs

Another way of representing structures is by using a graph. Below we give some basic graph-theoretic definitions, and show how graphs are used to represent the nonzero structure of a matrix.

2.3.1 Undirected graphs

An *undirected graph*, $G = (V, E)$, consists of a set V of *vertices* and a set E of *edges*. An *edge* (v, w) is an unordered pair of distinct vertices. If (v, w) is an edge the vertices v and w are *adjacent*. The edge (v, w) is *incident* on vertices v and w, which are its *endpoints*. The *degree* of a vertex, written $\deg(v)$, is the number of edges incident on v. Given a vertex v, the set $\Gamma(v) = \{x : (v, x) \in E\}$ is called the *adjacency set* of v. Similarly, for a set of vertices, $X \subseteq V$, we say that $\Gamma(X) = \{y : (x, y) \in E \text{ for some } x \in X\}$ is the *adjacency set* of X.

The *deficiency* of v is the set of non-edges between neighbors of v,

$$D(v) = \{\{x, y\} : \{v, x\} \in E, \{v, y\} \in E, \{x, y\} \notin E\}$$

The deficiency of v corresponds to the zeroes of the coefficient matrix that become nonzero when the equation in v's row is used to eliminate the variable in v's column.

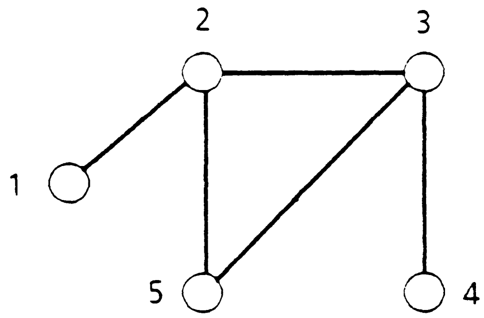
A *labeled graph* is a graph together with a mapping $\pi : \{1, 2, \ldots, N\} \rightarrow V$, where $N = |V|$. Unless stated otherwise, it is henceforth assumed that all graphs are labeled.

A *matching* in a graph G is a set of edges, $F \subseteq E$, such that no two edges in F have an endpoint in common. A vertex that is the endpoint of some edge in F is *covered*; otherwise it is *exposed*. If every vertex is covered by F then F is a *perfect matching*.

A *path* (sometimes called a *simple path*) between v and w in G is a sequence of distinct edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, where $v = v_0$ and $w = v_k$ and all the

$$\begin{bmatrix} 3.66 & 45.1 & 0.0 & 0.0 & 0.0 \\ 45.1 & 12.0 & 3.2 & 0.0 & 2.9 \\ 0.0 & 3.2 & 6.29 & 4.5 & 17.0 \\ 0.0 & 0.0 & 4.5 & 1.5 & 0.0 \\ 0.0 & 2.9 & 17.0 & 0.0 & 1.0 \end{bmatrix}$$

A real-valued, symmetric matrix with a nonzero diagonal



The corresponding undirected graph

Figure 2.2: Undirected graph representation of a matrix structure.

vertices are distinct except possibly $v_0 = v_k$. If F is a matching in G, an *alternating path* is a path in which every other edge is in F.

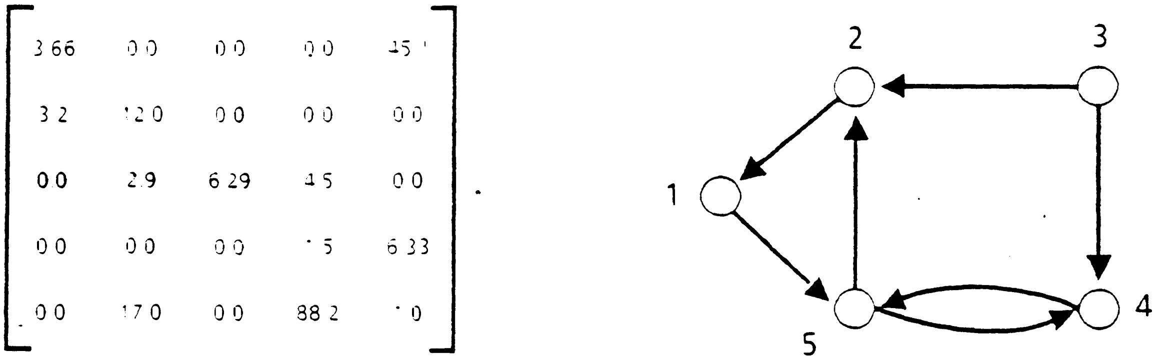
A *subgraph*, $G' = (V', E')$, of $G = (V, E)$ is a graph where $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$ then G' is a *spanning subgraph*. The set of edges of G with both endpoints in V' is written as $E(V')$. If $E' = E(V')$ then G' is the subgraph of G *induced* by the vertex set V'.

If our matrix A is square, symmetric, and has a nonzero diagonal, then it is convenient to use an undirected graph $G(A)$ to represent its nonzero structure.

The graph $G(A)$ will have n vertices and an edge $(\pi(i), \pi(j)), i \neq j$, if and only if entries $A_{ij} = A_{ji}$ are nonzero. The corresponding Boolean matrix is also known as the *adjacency matrix* of this graph. See Figure 2.2.

2.3.2 Directed graphs

A *directed graph* $D = (V, E)$ is a graph whose edges are ordered pairs of distinct vertices, $\langle v_i, v_j \rangle$, called *arcs*. A *directed path* from v to w in D is a sequence of distinct arcs $\langle v_0, v_1 \rangle, \langle v_1, v_2 \rangle, \ldots, \langle v_{k-1}, v_k \rangle$, where $v = v_0$ and $w = v_k$ and all the



A real-valued, unsymmetric matrix with a nonzero diagonal.

The corresponding directed graph

Figure 2.3: Directed graph representation of a matrix structure.

vertices are distinct except possibly $v_0 = v_k$. If for every two vertices v and w in D there is a directed path from v to w, then D is *strongly connected.*

If our matrix A is not symmetric, but square and with a nonzero diagonal, then we can use a directed graph $D(A)$ to represent its structure. This is a graph of n vertices and has a directed edge $\langle \pi(i), \pi(j) \rangle$ if and only if A_{ij} is nonzero. Again, the corresponding Boolean matrix is the adjacency matrix of the directed graph. See Figure 2.3.

2.3.3 Bipartite graphs

To represent the structure of a matrix of arbitrary dimensions and nonzero structure, a bipartite graph can be used.

An undirected graph whose vertex set can be divided into two disjoint sets V_1 and V_2, such that every edge in E has one endpoint in V_1 and the other in V_2, is called a *bipartite graph.* This graph is sometimes written $H = (V_1, V_2, E)$ to stress the partition of V.

A *labeled bipartite graph* is a bipartite graph together with two mappings, namely

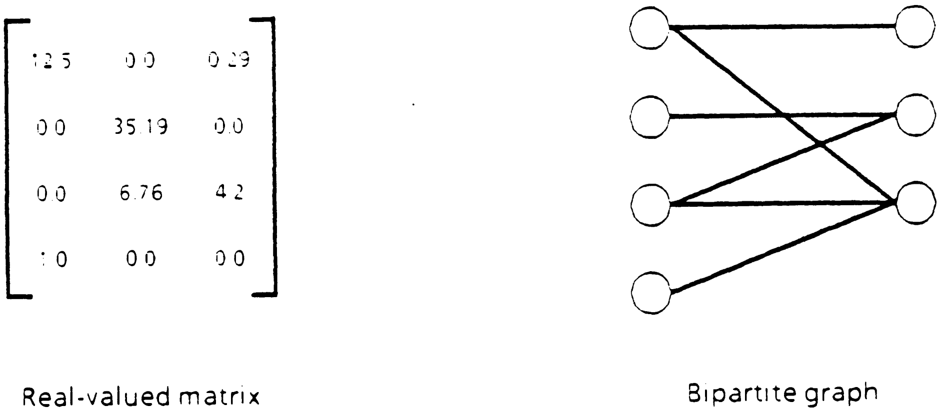


Figure 2.4: Bipartite graph representation of a matrix structure.

$\pi : \{1, 2, \ldots, N_1\} \rightarrow V_1$, where $N_1 = |V_1|$, and $\tau : \{1, 2, \ldots, N_2\} \rightarrow V_2$, where $N_2 = |V_2|$.

Representing the structure of a matrix with a bipartite graph

Let A be an $m \times n$ real matrix. We represent the nonzero structure of A with a graph, $H(A)$. This is a labeled bipartite graph $H = (V_1, V_2, E)$, where $(\pi(i), \tau(j)) \in E$ if and only if $A_{ij} \neq 0$. We refer to the vertices of V_1 as *row-vertices*, and to the vertices of V_2 as *column-vertices*. See Figure 2.4.

For a matrix A, we define the *structural rank* of A to be the size of a maximum matching in $H(A)$. The *numeric rank* of A is the number of linearly independent columns of A. We state some straightforward consequences of these definitions.

Lemma 2.1 *The structural rank of A is always at least as large as its numeric rank.*

Lemma 2.2 *If A is square and of full numeric rank then $H(A)$ has a perfect matching.*

Lemma 2.3 *If A is square and $H(A)$ has a perfect matching, then we can find values for the entries of A to make A nonsingular.*

Proof: Let F be a perfect matching in $H(A)$. Choose the entries of A that correspond to edges in F to have the value 1, and let all other entries have value 0. Then there will be exactly one nonzero term in the determinant of A, hence A will be nonsingular. □

The strong Hall property

Let $H = (V, W, E)$ be a bipartite graph with $|V| \geq |W|$. If, for every set of vertices $X \subseteq W$, X is adjacent to at least $|X|$ vertices in V, then we say that the bipartite graph H has the *Hall property*. If every proper subset X is adjacent to more than $|X|$ vertices, we say that H has the *strong Hall property*, or, for more precise definitions:

Definition 2.1 *A bipartite graph $H = (V, W, E)$ with $|V| \geq |W|$ has the* Hall property *if*

$$\forall X \subseteq W : |\Gamma(X)| \geq |X|.$$

Definition 2.2 *A bipartite graph $H = (V, W, E)$ with $|V| \geq |W|$ has the* strong Hall property *if*

$$\forall X \subset W, X \neq \emptyset : |\Gamma(X)| > |X|.$$

We list some basic and useful facts about matchings in bipartite graphs.

Lemma 2.4 (Hall's Theorem) [31] *Let $H = (V, W, E)$ be a bipartite graph. There is a matching in H of cardinality $|W|$ if and only if H has the Hall property.* □

Corollary 2.1 *Let $H = (V, W, E)$ be a bipartite graph with $|V| = |W|$. Then H has a perfect matching if and only if H has the Hall property.* □

With any directed graph D of n vertices we can associate a bipartite graph H_D as follows: $H_D = (X, Y, E)$, with $|X| = |Y| = n$, and (x_i, y_j) is an edge in E if $\langle v_i, v_j \rangle$ is an arc in D. In addition, all the edges (x_i, y_i) for $1 \leq i \leq n$ are in E.

The following observation, due to Johnson, Dulmage, and Mendelsohn [38], is also proved by Howell [35].

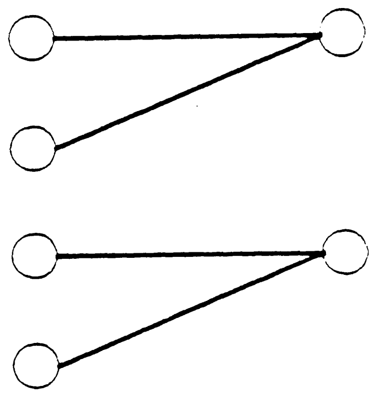


Figure 2.5: Disconnected graph with strong Hall property.

Lemma 2.5 [38] *D is strongly connected if and only if H_D has the strong Hall property.* □

Note that this correspondence between the strong Hall property and strong connectedness is well defined only when the bipartite graph is "square", that is, has the same number of vertices in each part. Indeed, it is possible for a rectangular bipartite graph that has the strong Hall property not to be connected, as in Figure 2.5. However, every bipartite graph has a canonical decomposition into subgraphs with the strong Hall property.

Chapter 3

Computing fill

Many matrix algorithms—e.g., multiplication, Cholesky factorization, inversion—take one or more matrices as input and produce a different matrix as output. Let α denote a matrix algorithm and A a matrix that can serve as input to α. We are mainly interested in what happens to the *structure* of A. Initially we have a structure $\tilde{A}$, and as a result of applying the algorithm α to A we get a new structure $\widetilde{\alpha(A)}$. It is this structure we want to compute directly, without having to apply the numerical algorithm α.

There is an important advantage to a separate computation of the nonzero structure. It allows us to use a static data structure during the numerical phase of the computation, something that makes the implementation simpler, more efficient, and saves space. As George and Heath point out [19], the importance of a static data structure can hardly be overemphasized.

3.1 Structural algorithms

There are of course any number of different algorithms that can be applied to matrix structures. Of a special interest to us, however, are *structural versions* of particular numerical algorithms.

One frequently employed method to compute matrix structures is to carry out a *structural version* of the numerical algorithm. To see how this is done, let us look at a particular representation for the nonzero structure, namely the Boolean matrix.

In brief, we get a structural algorithm from a numerical one by replacing numerical operations in the latter with Boolean operations. A strictly formal definition of how this is done would require us to specify concepts like "algorithm", "numerical operation", "Boolean operation", etc. Rather than go into all that detail, which would probably obscure more than clarify, we trust to the reader's intuition to fill in the gaps.

Definition 3.1 *Let A be an $m \times n$ matrix, $B = B(A)$ the corresponding Boolean matrix, and α a numerical algorithm that can be applied to A. A structural version of α, written $\tilde{\alpha}$, is constructed from α as follows:*

1. *$\tilde{\alpha}$ takes the Boolean matrix B as input.*
2. *Every addition and subtraction in α is replaced by a logical OR.*
3. *Every division with nonzero denominator and multiplication is replaced by a logical AND.*

This way of making structural versions out of numerical algorithms does not work on all numerical algorithms. For example, if there are steps where the magnitude of two values are compared, as in Gaussian elimination with pivoting, then this can in general not be emulated in the structural version of the algorithm. Pivoting, then, cannot be readily modeled. However, we take the following approach: the operations of the algorithm are divided into two separate phases. In phase 1 the rows and columns of the matrix are reordered according to the desired pivoting order. In phase 2, we carry out the numerical operations, now without pivoting.

3.2 Correct prediction of structures

For clarity of presentation we will define most concepts below in terms of algorithms that take a single matrix as input; the generalization to algorithms with several matrices as input is straightforward.

Assume as before that we have a numerical algorithm α which with input A produces a new matrix $R = \alpha(A)$. With the machinery of the preceding section available, we can describe at least two fundamentally different methods for computing the structure of R. The first method carries out the numerical algorithm and then extracts the structure from R, giving $\tilde{R}$. This indeed gives us the structure we are interested in, but it has the disadvantage of requiring a numerical computation.

A second method is to first extract the structure from A, giving $\tilde{A}$, and then use some structural algorithm σ on $\tilde{A}$ giving $\sigma(\tilde{A})$. This way it is not necessary to do any numerical computations; we can compute the structure by doing operations on a graph, for example. But another question arises: *are the two structures equivalent?* The answer to this question is of course that it depends on the algorithms.

Granted that the correctness of the second method outlined above depends on the structural algorithm used, we might want to characterize structural algorithms as being "correct" or "incorrect". But a structural algorithm can not be evaluated in isolation; its "correctness" can only be determined in relation to some numerical algorithm it attempts to emulate. It is therefore natural to look at pairs of algorithms, consisting of one numerical algorithm and one structural algorithm.

We say that a pair of algorithms (α, σ), where α is a numerical algorithm and σ is a structural algorithm, is *an exact pair of algorithms* if σ correctly predicts the structure of R for each matrix A to which α applies. Note that this definition leaves us with the task of defining what it means for σ to correctly predict the structure of R, given some specific matrix A. When it is understood from context which numerical algorithm is involved, we drop the reference to it and say that the structural algorithm σ is *exact* if (α, σ) is an exact pair.

The two structures $\tilde{R}$ and $\sigma(\tilde{A})$ may or may not be equal. In either case, the outcome may be due to "luck" with the numerical values in A rather than inherent equality of structure. To get around this problem in our definition we say that σ *correctly predicts the structure* of R if the following two conditions hold:

1. The structure $\sigma(\tilde{A})$ is large enough to hold all the nonzeros of $\tilde{R}$, and
2. There is some assignment of values to the nonzeros of A, such that the structure $\tilde{R}$ needs all of $\sigma(\tilde{A})$.

A more precise definition is:

Definition 3.2 *Let α be a numerical algorithm and σ a structural algorithm. Given a matrix A the structural algorithm σ* correctly predicts the structure $\widetilde{\alpha(A)}$ *if $\widetilde{\alpha(A)} \subseteq \sigma(\tilde{A})$, and there exists a matrix A' such that $\widetilde{A'} = \tilde{A}$ and $\widetilde{\alpha(A')} \supseteq \sigma(\tilde{A})$.*

The structural version of a numerical algorithm, $\tilde{\alpha}$, always satisfies the first of these conditions but may fail to satisfy the second. This is because the modeling of addition using logical OR is overly pessimistic $(7 + (-7) = 0$ but $1 \vee 1 = 1)$.

An example of a numerical algorithm that can readily be "Booleanized" is "matrix multiplication-with-transpose", $\nu(A) = A^T A$. If we write $C = A^T A$ then the algorithm can be specified as

$$c_{ij} = \sum_{1 \le k \le m} a_{ik} \cdot a_{jk}$$

Setting $\widetilde{D} = \tilde{\nu}(\tilde{A})$ and using Definition 3.1, we see that

$$\tilde{d}_{ij} = \bigvee_{1 \le k \le m} (\tilde{a}_{ik} \wedge \tilde{a}_{jk})$$

As an example of how structural algorithms can be expressed using graph models of the structures, let us show such a version of $\tilde{\nu}$. We assume that A is an $m \times n$ matrix. Its structure will be represented by a bipartite graph $H = (V, W, E)$, where $|V| = m$ and $|W| = n$. The output of the algorithm will be an undirected graph $G = (U, S)$, with $|U| = n$, representing the structure of $\tilde{A}^T \tilde{A}$.

STRUCTURAL MATRIX MULTIPLICATION-WITH-TRANSPOSE:

```
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        if ∃k : ((c_i, r_k) ∈ E and (c_j, r_k) ∈ E) then
            add (i, j) to S fi
    od
od
```

To continue with our example, let us show that $\tilde{\nu}$ is exact.

Lemma 3.1 *Let ν be the numerical algorithm "matrix multiplication-with-transpose", and let $\tilde{\nu}$ be its structural counterpart. Then $(\nu, \tilde{\nu})$ is an exact pair.*

Proof: Let A be an $m \times n$ matrix and let $C = A^T A$. For the first part that we have to prove, assume that $c_{ij} \neq 0$. Then there must be some k such that $a_{ik} \neq 0$ and $a_{jk} \neq 0$. But this implies that $\tilde{d}_{ij} \neq 0$, and hence $\widetilde{\nu(A)} \subseteq \tilde{\nu}(\tilde{A})$.

For the second part of the proof, we form A', a matrix of the same structure as A, by letting $a'_{ij} = |a_{ij}|$. Now, if $\tilde{d}_{ij} \neq 0$, then there is a k such that $\tilde{a}_{ik} \neq 0$ and $\tilde{a}_{jk} \neq 0$, which implies that $a'_{ik} > 0$ and $a'_{jk} > 0$. Since all elements in A' are nonnegative, each term in $\sum a'_{ik} \cdot a'_{jk}$ must be nonnegative. Hence the sum is positive, which means $c'_{ij} > 0$. It follows that $\widetilde{\nu(A')} \supseteq \tilde{\nu}(\tilde{A})$. □

There are numerical algorithms for which no exact structural algorithm exists. Consider, e.g., the algorithm that takes a 2×2 matrix A as input and if $A_{11} > 3$ outputs $R_1 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ and if $A_{11} \leq 3$ outputs $R_2 = \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}$. To correctly predict the structure for $A_1 = \begin{pmatrix} 9 & 0 \\ 0 & 0 \end{pmatrix}$ a structural algorithm should give $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$; and to correctly predict the structure for $A_2 = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}$ a structural algorithm should give $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. But A_1 and A_2 have the same structure, so no single structural algorithm, which takes $\widetilde{A_1} = \widetilde{A_2}$ as input, can correctly predict the structure for both A_1 and A_2.

An alternative, and weaker, definition would be to change the second requirement to:

2b. for each nonzero position (i,j) in $\sigma(\tilde{A})$, there exists a matrix A' such that $\widetilde{A'} = \tilde{A}$ and position (i,j) is nonzero in $\widetilde{\alpha(A)}$.

In our previous example, a structural algorithm that always predicts $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ on all inputs would now be considered exact. The numerical algorithms we will consider below, namely different factoring algorithms without pivoting, do have exact structural algorithms, however. We can see this as follows. In place of each nonzero value A_{ij} put a variable x_{ij}. Next, carry out the computations of the algorithm using symbolic algebra, simplifying expressions as much as possible. The result will be an algebraic expression for each position in the factor $\alpha(A)$ of A. If an expression simplifies to zero, then that position will be zero in $\alpha(A)$, otherwise it will be nonzero. A disadvantage is that there is no guarantee of a polynomial running-time for this algorithm. The size of the algebraic expressions may be exponential in the number of variables in A.

As a candidate for the structural algorithm we are especially interested in the structural version of α, i.e., $\tilde{\alpha}$, as defined in Section 3.1. Many common numerical algorithms do not form exact pairs together with their structural counterpart. For example, Cholesky factorization, Givens factorization, and Householder factorization, all fail to form exact pairs, as is shown below in Figure 3.4 and Section 3.4.1. Rather than looking for an alternative structural algorithm to form an exact pair, it is often satisfactory to show that a structural algorithm is exact on some subclass of matrices. If the subclass contains many of the matrices encountered in practice, then it may still be sufficient to use a structural algorithm that is not exact.

3.3 Structural Gaussian elimination

Structural algorithms on graphs

As we can represent the nonzero structures of matrices with labeled graphs, so we can also model several common algorithms by operations on graphs. Below, we will attempt to model Gaussian elimination on the three graph models discussed in Section 2.3. In the rest of this section it is assumed that the matrix is square.

Let π be the labeling of the vertices of G. To simplify notation we will use i to mean $\pi(i)$, when no confusion can arise.

On a square symmetric matrix with nonzero diagonal, structural Gaussian elimination is modeled on an undirected graph [47].

UNDIRECTED ELIMINATION GAME:

```
1. for v ← 1 to n do
2.     S ← Γ(v)
3.     Add edges to make S a clique
4.     Delete v and its incident edges
   od
```

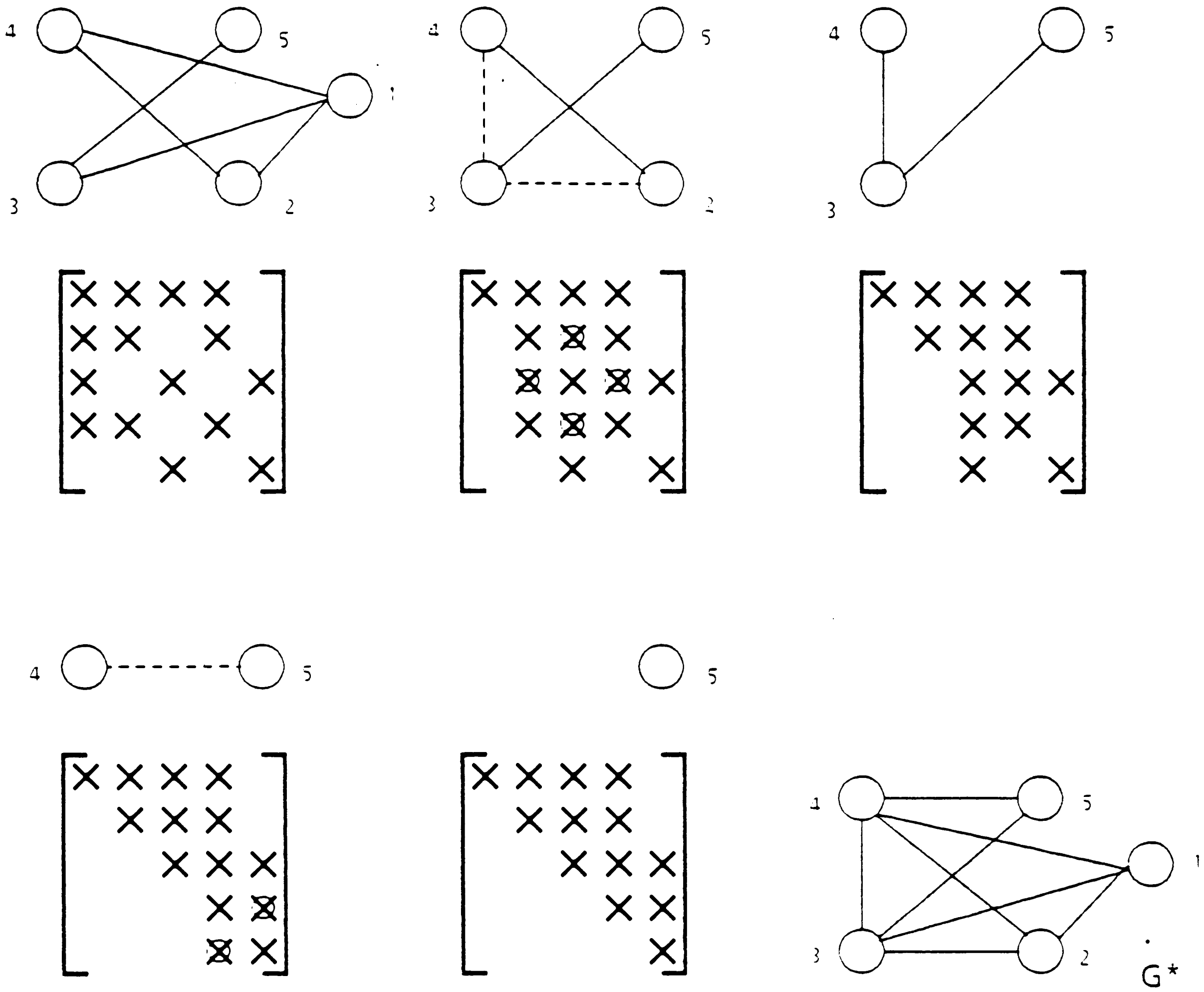
The graph that results from adding to G all the edges added in step 3, is called the *filled graph*, denoted G^*. For an example, see Figure 3.1.

On a square, not necessarily symmetric, matrix with a nonzero diagonal, structural Gaussian elimination is modeled on a directed graph [4,51].

DIRECTED ELIMINATION GAME:

```
1. for v ← 1 to n do
2.     for all pairs u, w such that ⟨u, v⟩, ⟨v, w⟩ ∈ E do
3.         Add the edge ⟨u, w⟩ od
4.     Delete v and its incident edges
   od
```

On a square matrix of arbitrary structure, structural Gaussian elimination is modeled on a bipartite graph [23,28]. The process on the bipartite graph is a bit more complicated; it is also richer in possibilities and can model off-diagonal



Edge that filled in during previous step.

Entry that filled in during previous step.

Figure 3.1: Structural Gaussian elimination on undirected graph.

pivoting. As before, we have to assume that pivoting is carried out by making all the row- and column-permutations first.

If π, τ are the labelings of the vertices let $r_i = \pi(i)$ and $c_i = \tau(i)$. Structural Gaussian elimination then works as follows on the bipartite graph.

BIPARTITE ELIMINATION GAME:

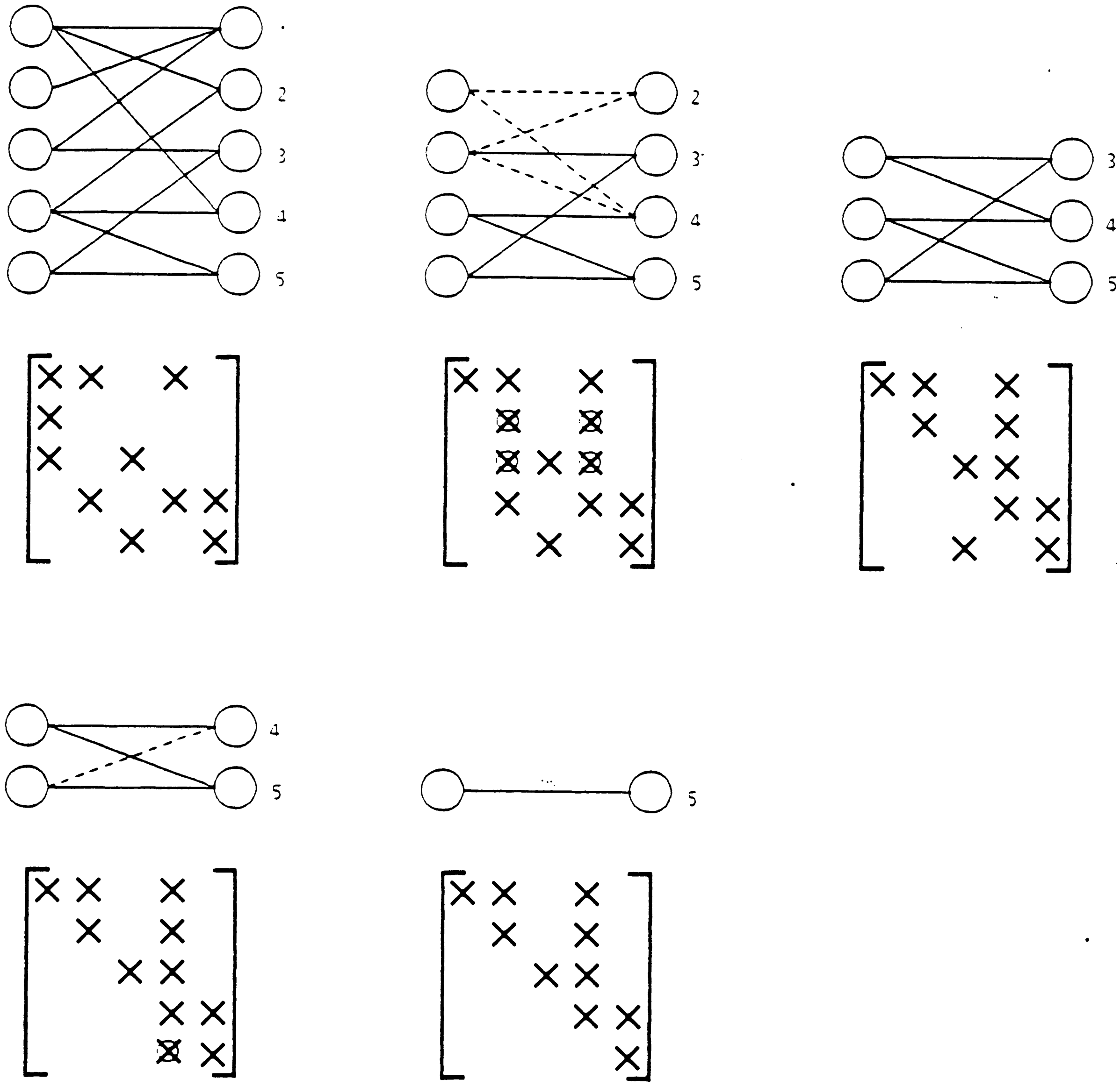
for $i \leftarrow 1$ **to** n **do**
    $S \leftarrow \Gamma(r_i) \setminus \{c_i\}$
    $T \leftarrow \Gamma(c_i) \setminus \{r_i\}$
    Add edges to make $S \cup T$ a complete bipartite graph
    Delete r_i and c_i and their incident edges
od

Figure 3.2 contains an example.

This procedure always works, and halts in a finite number of steps. But to make sure that it really models structural Gaussian elimination as defined above, one more assumption is necessary. A possible problem is that at the i-th step of the algorithm the edge (r_i, c_i) may not exist in the graph. From the description of the algorithm it is not immediately obvious why this presents a problem, but if we think of what this corresponds to in the matrix we see that it means pivoting on a zero element! Hence, we make the following assumption: the labelings π and τ are chosen so that at the i-th step the edge (r_i, c_i) is in the graph, either as an original edge or as an edge that has been added earlier in the algorithm. Labelings that abide by this rule are called *legal labelings*, and henceforth we assume that all labelings are legal.

Path lemmas

In all three elimination games above the resulting graph G^* is completely determined by the algorithm. In other words, to find out what G^* looks like we can simply carry out the graph algorithm. However, there exist static characterizations of the



--------- Edge that filled in during previous step.

Entry that filled in during previous step.

Figure 3.2: Structural Gaussian elimination on bipartite graph.

structure of G^*. This means that we can predict where the edges in G^* will be without going through the whole algorithm.

Proposition 3.1 [47,50] *Let G be an undirected labeled graph with N vertices and a labeling π, and let G^* be the filled graph. Then (u, v) is an edge in G^* if and only if there exists a path from u to v in G going only through vertices x such that $\pi^{-1}(x) < \pi^{-1}(u)$ and $\pi^{-1}(x) < \pi^{-1}(v)$.* □

Proposition 3.2 [50] *Let D be a directed labeled graph with N vertices and a labeling π, and let D^* be the filled graph. Then $\langle u, v\rangle$ is a directed edge in D^* if and only if there exists a directed path from u to v in G going only through vertices x such that $\pi^{-1}(x) < \pi^{-1}(u)$ and $\pi^{-1}(x) < \pi^{-1}(v)$.* □

The bipartite case is again a little bit more complicated. A path lemma for the bipartite case appears in Gilbert's thesis [23], but contains a flaw. A different version is:

Theorem 3.1 *Let H be a labeled bipartite graph with N row- and column-vertices and labelings π and τ, and let H^* be the filled bipartite graph. Then (u, v) is an edge in H^* if and only if there exists an alternating path from u to v in H' going only through vertices x such that $\pi^{-1}(x) < \pi^{-1}(u)$ and $\pi^{-1}(x) < \pi^{-1}(v)$. The graph H' is H with all edges of the form $(\pi(i), \tau(i))$ added, and the matching that the path alternates over consists of all edges $(\pi(i), \tau(i))$.*

Remark: Since we require all labelings to be legal, it follows that all edges of the form $(\pi(i), \tau(i))$ will appear sooner or later in the graph.

Proof: As before, we let $r_i = \pi(i)$ and $c_i = \tau(i)$. First, assume that a path of the required form exists between r_i and c_j in H'. Proof is by induction on path length. The base case is length one, that is, the edge is present in H. For the inductive step, let (r_k, c_k) be the vertex pair on the path that is eliminated first. The path is of the form $r_i, \ldots, c_q, r_q, c_k, r_k, c_p, r_p, \ldots, c_j$. See Figure 3.3. When (r_k, c_k) is eliminated,

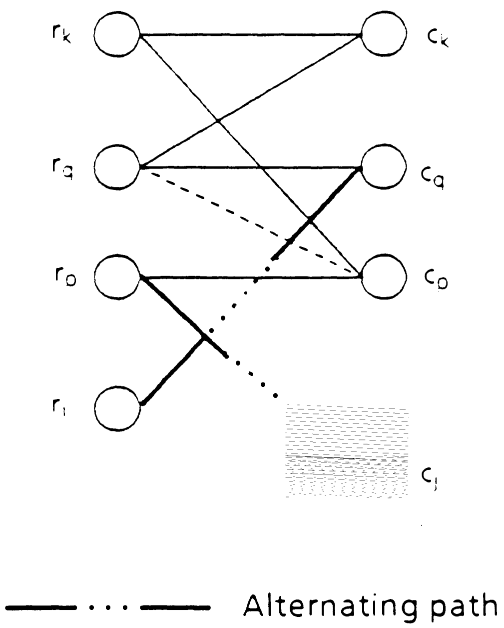
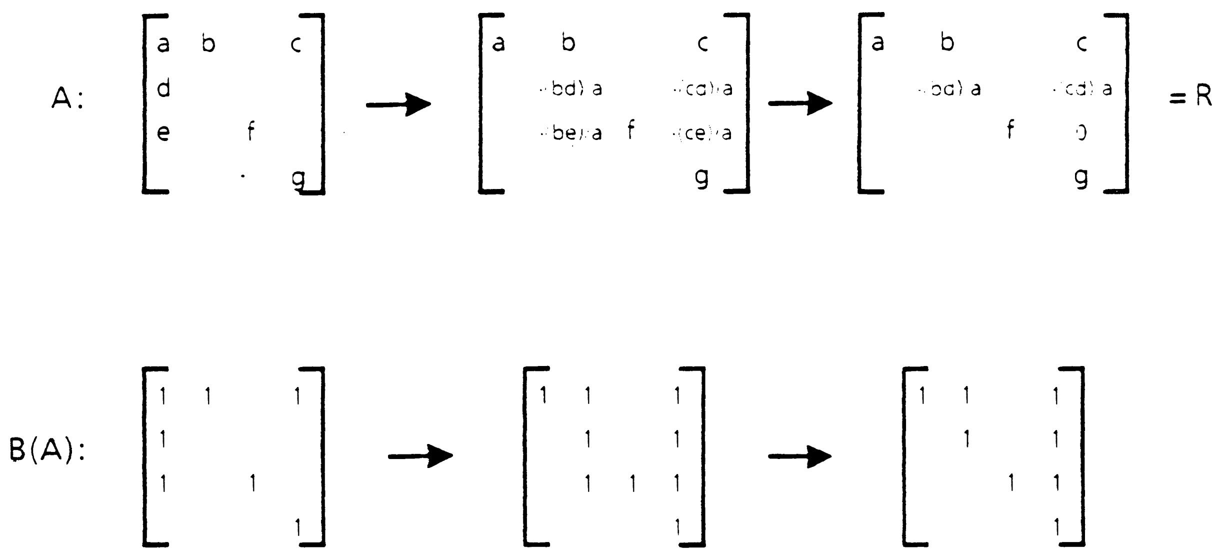


Figure 3.3: The situation when (r_k, c_k) is eliminated.

the edge (r_q, c_p) will fill in. Hence we will get a graph in which there is a path of the prescribed kind from r_i to c_j of length one less than before. By the inductive hypothesis it follows that the edge (r_i, c_j) will fill in in H^*.

For the other direction of the proof, assume that $e = (r_i, c_j)$ is an edge in H^*. The proof is by induction on $\min\{i, j\}$. If e is in H then we are done. If not, assume that e filled in at the k-th stage. At this stage, there were edges (r_i, c_k) and (r_k, c_j) in the graph. Both of these edges were present in H or were added earlier in the process, and so by the inductive hypothesis there are alternating paths from r_i to c_k, and from r_k to c_j. These paths may not be distinct, but since they are alternating paths they can be merged into one alternating path from r_i to c_j. □

Let α denote the numerical algorithm Gaussian elimination. Together with the structural version of this algorithm, $\tilde{\alpha}$, the pair $(\alpha, \tilde{\alpha})$ is not exact [3,47]. This is demonstrated in Figure 3.4. On some matrices, however, structural Gaussian elimination works well. It is possible to characterize the matrices for which $\tilde{\alpha}$ predicts the correct structure. Before we state the theorem we will introduce notation for a



Example of a matrix A where structural Gaussian elimination predicts a structure that is too large: entry (3,4) is predicted nonzero, but will always be zero in R.

Figure 3.4: Structural Gaussian elimination fails.

subgraph that represents a certain submatrix; it is a submatrix that is a principal submatrix except possibly for the last row or column. See Figure 3.5.

Definition 3.3 *Let $H = (V_1, V_2, E)$ be a bipartite graph with $|V_1| = |V_2| = n$. The induced subgraph $H_s = (\{r_1, \ldots, r_k\} \cup \{r_i\}, \{c_1, \ldots, c_k\} \cup \{c_j\}, E)$ where $k = \min\{i, j\} - 1$, is denoted $H_s = \langle H, i, j \rangle$.*

Remark: A theorem similar to our Theorem 3.2 below, but with a quite different proof, was given by Brayton et. al. [3].

Theorem 3.2 *Let A be an $n \times n$ matrix and let $H = H(A)$ be the labeled bipartite graph representing the nonzero structure of A. The bipartite elimination game on H correctly predicts the nonzero structure of $\alpha(A)$ if and only if for every edge (r_i, c_j) in H^*, the induced subgraph $H' = \langle H, i, j \rangle$ has a perfect matching.*

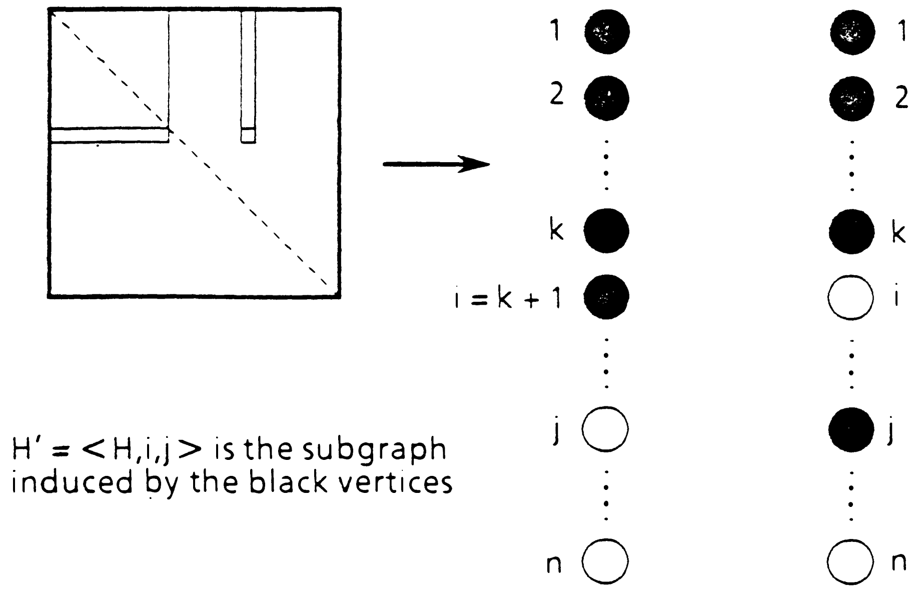


Figure 3.5: A certain submatrix and its graph.

Remark: The conditions of the theorem are not only structural in nature, but also involve the labeling of the bipartite graph (or, equivalently, the ordering of columns and rows of the matrix). Indeed, since any matrix of full rank has a perfect matching (by Lemma 2.2), it also has an ordering that gives a full nonzero diagonal; and hence, as is shown below, the bipartite elimination game correctly predicts the structure on this reordered matrix. In Section 4.1 we will discuss further the strategy of always starting by reordering the matrix to have a full nonzero diagonal.

Proof: Assume that the bipartite elimination game (which is nothing but the graph-version of structural Gaussian elimination) correctly predicts the structure $\widetilde{\alpha(A)} = \tilde{R}$. Let $e = (r_i, c_j)$ be an edge in H^* that is not in H, and assume that $i \leq j$, $k = i - 1$. Form the subgraph $H_s = \langle H, i, j \rangle$. Since the fill prediction is correct we know that there exists a matrix A' of the same structure as A, and such that $R'_{ij} \neq 0$. Hence, the submatrix of R' that corresponds to H_s has a full nonzero diagonal, which by Lemma 2.2 implies that H_s has a perfect matching.

For the other direction of the proof we need the following lemma.

Lemma 3.2 *Let A be a matrix with t nonzeros, where in place of each nonzero we have put a variable, $x_1, x_2, \ldots, x_t$. Let $M_1, M_2, \ldots, M_k$ be submatrices of A. Suppose that for each i with $1 \le i \le k$ there is an assignment of values $\mathbf{a}^{(i)} = (a_1^{(i)}, a_2^{(i)}, \ldots, a_t^{(i)})$ to the nonzero entries of A such that M_i is nonsingular. Then there is a single assignment $\mathbf{a} = (a_1, \ldots, a_t)$ such that every M_i, $1 \le i \le k$, is nonsingular.*

Proof: Each $\det(M_i)$ is a nonzero polynomial $p_i(\mathbf{x})$. The zeros of such a polynomial form a set of measure zero [13]. Since there are only k such polynomials the set $S = \bigcup_i \{\mathbf{x} : p_i(\mathbf{x}) = 0\}$ is also of measure zero. Hence, there must exist a point $\mathbf{a} \notin S$; so $p_i(\mathbf{a}) \neq 0$ for all i. □

To continue with the proof of the theorem, let $e = (r_i, c_j)$ be an edge in H^*. Assume that each submatrix of the prescribed kind has a perfect matching. Then, by Lemma 2.3, for each such submatrix we can find values for its entries that make it nonsingular. By Lemma 3.2 it follows that there is one assignment of values to A's nonzeros that makes all the submatrices nonsingular at once. Call this matrix A'. But if a square submatrix is nonsingular, then its factor is also nonsingular, so the factor has a nonzero diagonal. Hence $R'_{ij} \neq 0$, and the theorem follows. □

One common type of matrix is included in this structural class, as shown in the following lemma:

Corollary 3.1 *For any matrix A with a full nonzero diagonal the bipartite elimination game correctly predicts the structure of $\sigma(A)$.*

Proof: Given a matrix A with a full nonzero diagonal, let $H = H(A)$ be the corresponding bipartite graph. For each edge $e = (r_i, c_j)$ that is in H^* but not in H, we have to show that $H' = \langle H, i, j \rangle$ has a perfect matching. Let F be the matching $\{(r_1, c_1), \ldots, (r_k, c_k)\}$. Under this matching r_i and c_j are exposed vertices in H'; but since e fills in, we know by Lemma 3.1 that there is an alternating path in H' from r_i to c_j going only through vertices in H'. Hence, we can create a new, larger, matching in H' by alternating along this path. This new matching is a perfect matching. □

Corollary 3.2 *For any matrix A that is symmetric and positive definite, the bipartite elimination game correctly predicts the structure of $\sigma(A)$.*

Proof: A symmetric positive definite matrix has a full nonzero diagonal. □

3.4 Structural QR factorization

3.4.1 Structural versions of common numerical algorithms.

Consider solving a system of linear equations

$$A\mathbf{x} \simeq \mathbf{b}$$

where A is an $m \times n$, $m \geq n$, large, sparse matrix. In the rest of this section we assume that A has full column rank. When A is rectangular the system is overdetermined, and we seek the least squares solution—i.e., the solution that minimizes

$$\|A\mathbf{x} - \mathbf{b}\|_2.$$

There are several different methods to find this solution [33]. We will take a look at two of them, *orthogonalization* and the *normal equations* method.

In orthogonalization we factor A into

$$A = Q\begin{pmatrix} R \\ O \end{pmatrix}$$

where Q is an $m \times m$ orthogonal matrix and R is an $n \times n$ upper triangular matrix, and then solve the upper triangular system

$$\begin{pmatrix} R \\ O \end{pmatrix}\mathbf{x} = Q^T\mathbf{b}.$$

Numerically, we can compute Q and R using Householder reflections [27] or Givens rotations [19].

With the normal equations approach, we multiply by A^T and solve the square system

$$A^T A\mathbf{x} = A^T\mathbf{b}.$$

If A is of full rank, then $A^T A$ is symmetric and positive definite, and we can factor $A^T A$ into its Cholesky factors LL^T.

From the following equations, and the uniqueness of L,

$$LL^T = A^T A = (QR)^T(QR) = R^T R,$$

we see that L^T is equal to R except for possible sign differences in some rows. We can identify three different numerical algorithms for computing the factor R.

1. Orthogonalization using Givens rotations. Let this algorithm be denoted α. More specifically, we mean Givens rotations carried out column by column, from left to right, and within each column lower-numbered to higher-numbered rows.

2. Orthogonalization using Householder reflections. Let this algorithm be denoted β. Specifically, Householder reflections carried out column by column, from left to right.

3. The normal equations algorithm, denoted γ.

The structural versions of these three algorithms do not always yield the same result, and they do not always correctly predict the structure $\tilde{R}$. Interestingly enough, though, there is a relation between the structures predicted by the different algorithms.

Theorem 3.3 *Let A be an $m \times n$ matrix with $m \geq n$, and let $R = \alpha(A) = \beta(A) = \gamma(A)$ be the upper triangular factor as computed by any of the three algorithms mentioned above. None of these algorithms form exact pairs with their structural counterparts, but $\tilde{R} \subseteq \tilde{\alpha}(\tilde{A}) \subseteq \tilde{\beta}(\tilde{A}) \subseteq \tilde{\gamma}(\tilde{A})$.*

Remark: In other words, all three algorithms always produce a structure that has room for all the nonzeros of $\tilde{R}$. All of them also may produce a structure that is too large. The Givens algorithm is never worse than the Householder algorithm, which in turn is never worse than the normal equations algorithm. In the following section we will identify a subclass of graphs for which all three algorithms predict the correct structure.

We will prove Theorem 3.3 by dividing it into several lemmas. Before we do so, however, we will give precise definitions of the structural algorithms as carried out on a labeled bipartite graph. Just as before we write $r_i = \pi(i)$ and $c_i = \tau(i)$, and we require the labelings to be legal. As in the case of Gaussian elimination, this means that we require each diagonal element to be nonzero at the time its column is zeroed. This may be considered a more severe restriction in the cases of Givens and Householder factorization, since these numerical methods do not need to perform any explicit pivoting to place nonzeros on the diagonal. However, without this assumption it becomes difficult to model the changes in structure with any accuracy. Also, note that the factor $\boldsymbol{R}$ is independent of row permutations in $\boldsymbol{A}$. Hence, it may be justified to assume that the rows are ordered to achieve a legal labeling.

Here follows now the structural algorithms:

STRUCTURAL GIVENS FACTORIZATION:

for $i \leftarrow 1$ **to** $n-1$ **do**
    $U \leftarrow \Gamma(c_i)$
    for each $r_k \in U$ **let** $\mathcal{W}_k \leftarrow \Gamma(r_k)$
    for each $r_k \in U$ **do**
        $\mathcal{W} \leftarrow \bigcup_{i \le j < k} \mathcal{W}_k$
        Add edges from r_k to each vertex in $\mathcal{W}$
        Add edges from r_i to each vertex in $\mathcal{W}$
    od
    Delete vertices c_i and r_i and incident edges
od

STRUCTURAL HOUSEHOLDER FACTORIZATION:
1. **for** $i \leftarrow 1$ **to** $n-1$ **do**
2. $U \leftarrow \Gamma(c_i)$
3. $W \leftarrow \Gamma(U)$
4. Add enough edges to make the subgraph induced by vertices in U, W a complete bipartite graph.
5. Delete vertices c_i, r_i and incident edges

od

George and Heath [19] suggested using a structural version of normal equations to estimate $\tilde{R}$. Like any other Booleanized numerical algorithm, $\tilde{\gamma}$ will give a structure that has room for all the nonzeros of $\tilde{R}$, but the structure may be too large.

STRUCTURAL NORMAL EQUATIONS:
1. Using STRUCTURAL MATRIX MULTIPLICATION-WITH-TRANSPOSE (as defined on page 16) form $G \leftarrow \tilde{\nu}(H)$
2. Use STRUCTURAL GAUSSIAN ELIMINATION on G to compute the structure of $\tilde{R}$

Lemma 3.3 *With the notation of Theorem 3.3,* $\tilde{R} \subseteq \tilde{\alpha}(\tilde{A})$, *and there exists a matrix* A' *such that* $\widetilde{R'} \neq \tilde{\alpha}(\widetilde{A'})$.

Proof: That $\tilde{R} \subseteq \tilde{\alpha}(\tilde{A})$ follows from the fact that this is true of any "Booleanized" numerical algorithm. Figure 3.6 shows a matrix A' for which $\widetilde{R'} \neq \tilde{\alpha}(\widetilde{A'})$. □

Lemma 3.4 *With the notation of Theorem 3.3,* $\tilde{\alpha}(\tilde{A}) \subseteq \tilde{\beta}(\tilde{A})$.

Proof: This follows immediately from the definitions above, since the edges added in the Givens factorization are a subset of the edges added in the Householder factorization. □

Lemma 3.5 *With the notation of Theorem 3.3,* $\tilde{\beta}(\tilde{A}) \subseteq \tilde{\gamma}(\tilde{A})$, *and there exists a matrix* A' *such that* $\tilde{\beta}(\widetilde{A'}) \neq \tilde{\gamma}(\widetilde{A'})$.

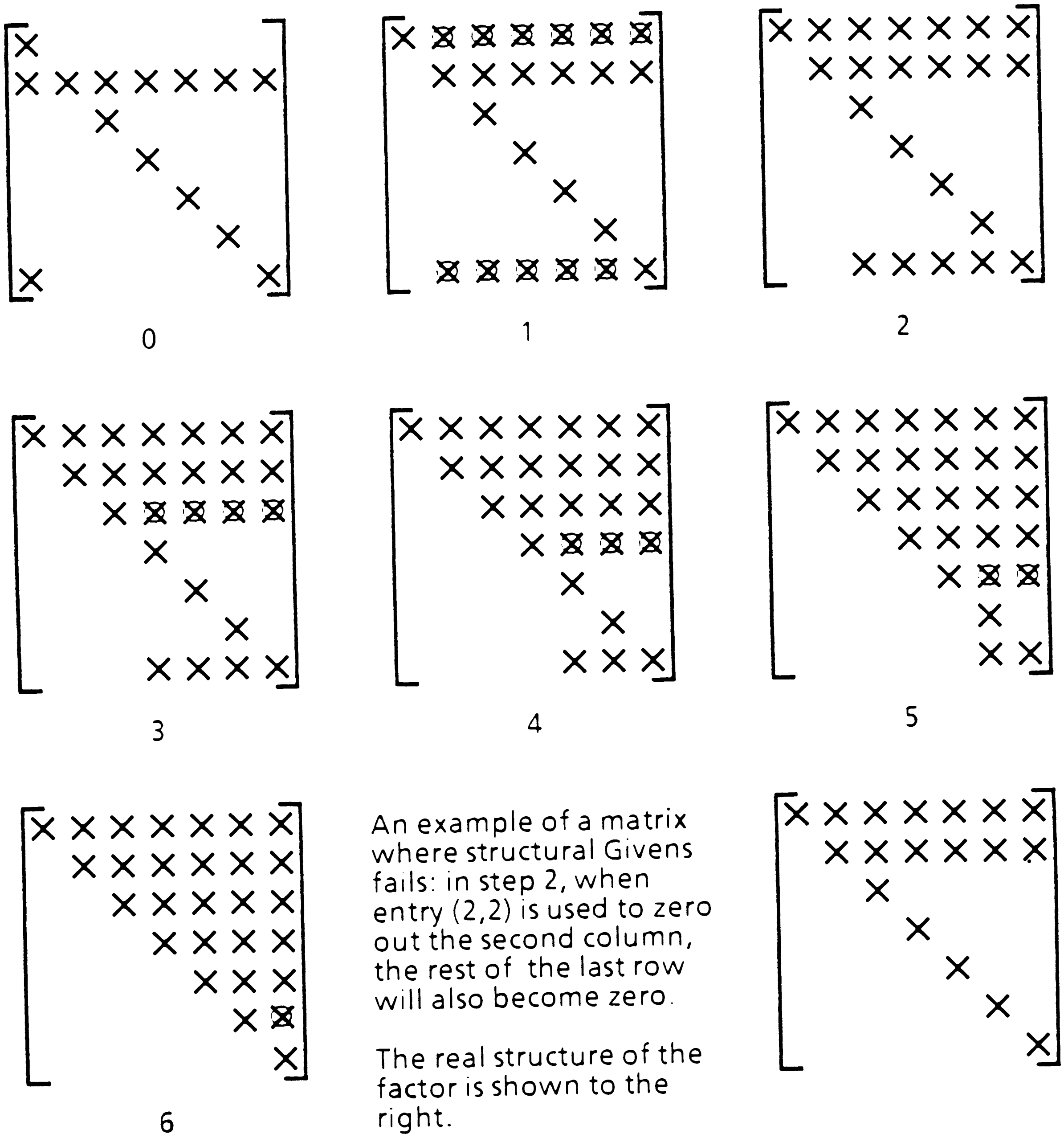


Figure 3.6: Structural Givens factorization fails.

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

Figure 3.7: Structural Householder differs from normal equations.

Proof: Figure 3.7 shows a matrix A' such that $\tilde{\beta}(\widetilde{A'}) \neq \tilde{\gamma}(\widetilde{A'})$.

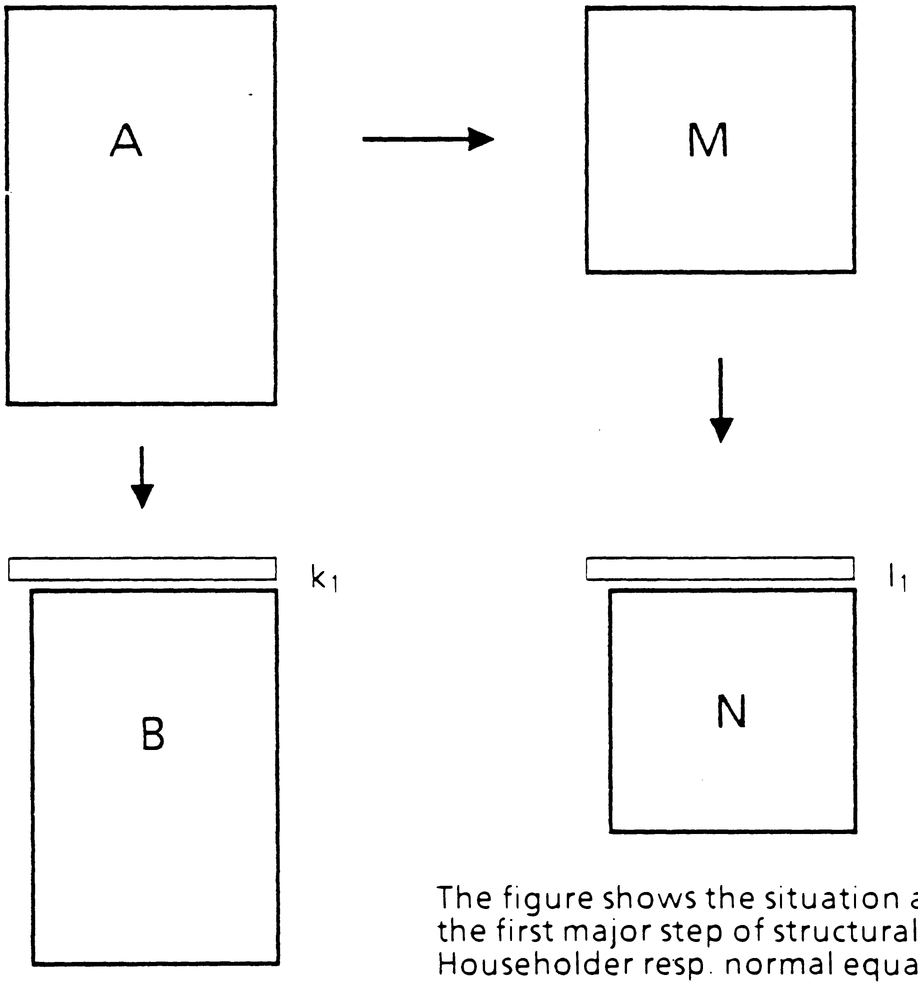
Let $\tilde{A}$ be the structure of the matrix A. If ν is the numerical algorithm "matrix multiplication-with-transpose", then let $\widetilde{M} = \tilde{\nu}(\tilde{A})$. By Lemma 3.1 $(\nu, \tilde{\nu})$ is an exact pair, hence $\widetilde{M}$ is the correct structure.

The proof is by induction on the number of columns in A. By carrying out one major step of the structural Householder algorithm, we zero out the first column, and get one row structure $\widetilde{k_1}$ plus a $(m-1) \times (n-1)$ matrix structure $\tilde{B}$. See Figure 3.8.

Similarly, after zeroing out the first column in $\widetilde{M}$ using structural Gaussian elimination, we get one row structure $\tilde{l}_1$, which is just the first row of $\widetilde{M}$, plus an $(n-1) \times (n-1)$ matrix structure, $\widetilde{N}$. See Figure 3.8.

Now, assume that there is a nonzero in position i in row k_1. By the assumption that the ordering of rows and columns in A is legal, we can conclude that $\tilde{a}_{11} \neq 0$. If $\tilde{a}_{1i}$ is nonzero, then columns 1 and i in $\tilde{A}$ both have nonzeros in row 1. Hence entries $\widetilde{m}_{1i}$ and $\widetilde{m}_{i1}$ are both nonzero. If $\tilde{a}_{1i}$ is zero, then this position filled in during the zeroing of the first column; which means that some row j in $\tilde{A}$, $j > 1$, has a nonzero in positions 1 and i. But this implies that $\widetilde{m}_{1i}$ is nonzero. We conclude that $\widetilde{k_1} \subseteq \tilde{l}_1$.

Next, assume that position (i, j) in $\tilde{B}^T \tilde{B}$ is nonzero. This means that some row r in $\tilde{B}$ has nonzeros in columns i and j; this in turn means that for each of these entries, either they were nonzero in $\tilde{A}$ or row r has a nonzero in column 1 and there



The figure shows the situation after the first major step of structural Householder resp. normal equations.

Figure 3.8: Structural Householder vs. normal equations.

is some row p with nonzeros in columns 1 and i (or some row q with nonzeros in columns 1 and j). This implies that entries $(1,i)$ and $(1,j)$ in $\widetilde{M}$ are nonzero (and, because of symmetry, also entries $(i,1)$ and $(j,1)$). From this it follows that position (i,j) in $\widetilde{N}$ is also nonzero. We conclude that $\tilde{B}^T\tilde{B} \subseteq \widetilde{N}$.

Now, by the inductive hypothesis, if we factor $\tilde{B}$ using structural Householder the result will be a substructure of the structure we get by factoring $\tilde{B}^T\tilde{B}$ using structural Gaussian elimination. And since $\tilde{B}^T\tilde{B}$ is a substructure of $\widetilde{N}$, the structure of the factor of $\tilde{B}^T\tilde{B}$ must be a substructure of $\widetilde{N}$. From this and $\widetilde{k_1} \subseteq \tilde{l}_1$ it follows that $\tilde{\beta}(\tilde{A}) \subseteq \tilde{\gamma}(\tilde{A})$. $\square$

Theorem 3.3 now follows directly from the Lemmas above.

3.4.2 Structural QR factorization on matrices with the Strong Hall property.

The main result of this section is that all three of the algorithms discussed above correctly predict the structure $\tilde{R}$ if $\tilde{A}$ has the strong Hall property. By Theorem 3.3 this would follow immediately if only we knew that normal equations works correctly, i.e., if $\tilde{R} = \tilde{\gamma}(\tilde{A})$.

Special submatrices

The matrix in Figure 3.6 is an example of a matrix on which structural normal equations may overpredict fill. Why does the normal equations algorithm not give the correct answer? Let us analyze the algorithm and try to find the point where things may go wrong.

There are two main steps in the algorithm, forming $\tilde{U} = \tilde{A}^T\tilde{A}$, and factoring $\tilde{U}$ using structural Gaussian elimination. By Lemma 3.1, matrix multiplication-with-transpose is exact, hence it cannot be this step that makes the algorithm fail.

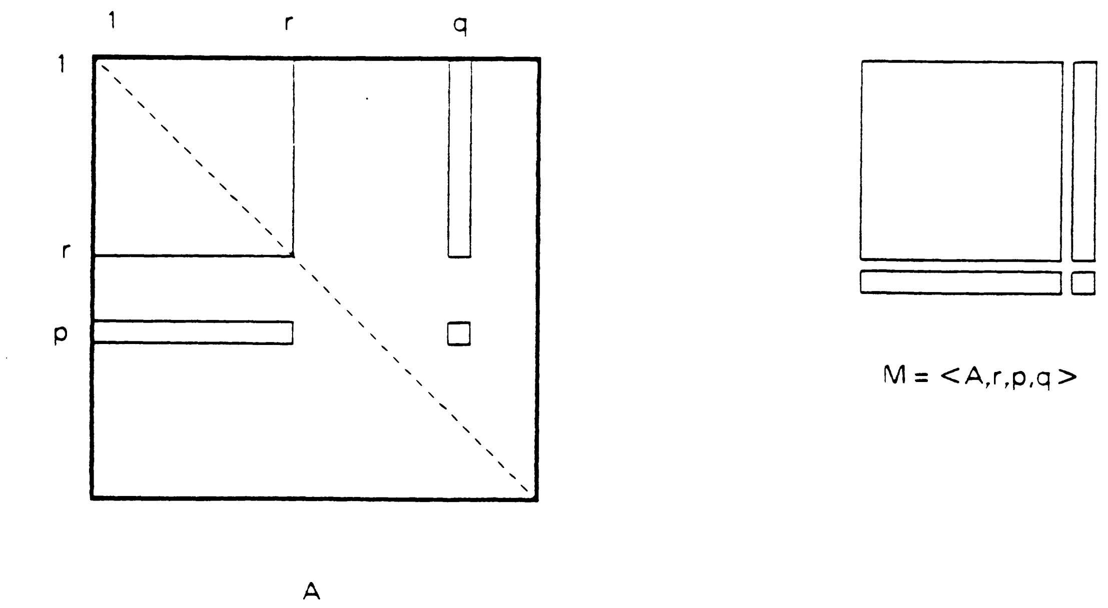


Figure 3.9: A special submatrix.

The second step is a structural Gaussian elimination of $\tilde{U}$. By assumption, A is of full column rank, hence $A^T A$ is symmetric positive definite. Now, Corollary 3.2 states that structural Gaussian elimination works correctly if the matrix is symmetric positive definite. So this step is okay too, or what is going on here?

Well, we were just a little too quick in claiming that the second step is exact. The entries in $A^T A$ are not independent: the structure of A may be such that cancellation will occur when we do Gaussian elimination on $A^T A$, regardless of the values of the nonzeros in A. In general, the composition of two exact structural algorithms may not be exact.

The following lemma shows that if cancellation occurs at position (p, q) in $A^T A$ during Gaussian elimination, then this position is in the lower right-hand corner of a singular submatrix of $A^T A$ of a special form described below.

Consider the square matrix M consisting of the principal $r \times r$ submatrix of $A^T A$, plus the first r entries of row p and column q, plus entry (p, q), where $1 \leq r < p \leq q \leq n$. See Figure 3.9. As a shorthand notation we will write $M = \langle A, r, p, q \rangle$.

Lemma 3.6 *Let A be an $m \times n$, $m \geq n$, matrix with full column rank. If cancellation occurs in A^TA at entry (p,q) during the r-th stage of the Gaussian elimination, when row r is used to zero out entry (p,r), then $M = \langle A, r, p, q\rangle$ has full structural rank and is singular.*

Proof: To show that M has full structural rank, let $C = H(M)$ be the bipartite graph that represents the structure of M, and let $H = (X, Y, S) = H(A^TA)$ be the bipartite graph that represents the structure of A^TA. Clearly, C is a subgraph of H.

Since A^TA is a symmetric matrix with a nonzero diagonal, a perfect matching in H is $F = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. By Lemma 3.1 it then follows that position (p,q) in A^TA fills in if and only if there is an alternating path in H from vertex x_p to vertex y_q going only through lower-numbered vertices. Hence, if we have cancellation at position (p,q), such a path exists, and by interchanging the matching and non-matching edges of this path we can construct a perfect matching in C. Any vertex x_i that does not lie on the alternating path will be matched with y_i.

To see that M is singular, note that performing Gaussian elimination on a square matrix does not change the value of its determinant; and since the resulting matrix is upper triangular and has one diagonal entry equal to zero, it must be singular. □

A structural lemma

In this section we show that if $\tilde{A}$ has the strong Hall property, then we can make any one of the special submatrices of A^TA nonsingular by choosing appropriate values for the nonzeros of A.

Let $H = (V, W, E)$ be the bipartite graph model of $\tilde{A}$. Given any two subsets $W_1, W_2 \subseteq W$ of column-vertices, we can define a new bipartite graph $H' = (X, Y, S)$ where $|X| = |W_1|$, $|Y| = |W_2|$, and $(x_i, y_j) \in S$ if and only if $w_i \in W_1$ and $w_j \in W_2$

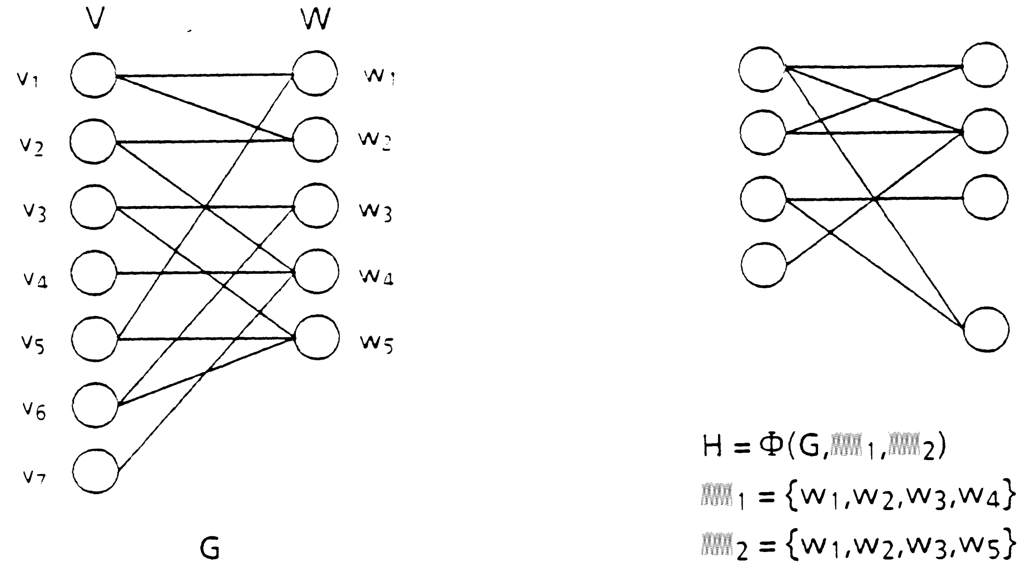


Figure 3.10: A product-bipartite graph.

and $w_j \in \Gamma(\Gamma(w_i))$. We write this as $H' = \Phi(H, \mathcal{W}_1, \mathcal{W}_2)$. See Figure 3.10 for an example.

If A_1 is the matrix containing only the columns of A represented by $\mathcal{W}_1$, and A_2 is the matrix containing the columns of $\mathcal{W}_2$, then $H' = H(A_1^T A_2)$, provided that no two columns that have some nonzeros in the same rows are orthogonal. This is true, for example, if all the entries are nonnegative. In particular, under this assumption $\Phi(H(A), \mathcal{W}, \mathcal{W}) = H(A^T A)$. As before, $H = (V_1, V_2, E)$ is a labeled bipartite graph with the strong Hall property. Let $\mathcal{W}_1 = \{w_1, \ldots, w_r\} \cup \{w_p\}$ and $\mathcal{W}_2 = \{w_1, \ldots, w_r\} \cup \{w_q\}$. Let $C = \Phi(H, \mathcal{W}_1, \mathcal{W}_2)$. We also use the shorthand notation $C = \langle H, r, p, q \rangle$ for this graph.

Now, we are given the structure of a matrix, i.e., a bipartite graph H. We want to show that provided that H has the strong Hall property, we can find a matrix A that fits into the given structure, and is such that for each subgraph $C = \langle H, r, p, q \rangle$ that has a perfect matching, the corresponding submatrix $M = \langle A, r, p, q \rangle$ is nonsingular. If we can show this, then by Lemma 3.6 we can say that there will be no cancellation during Gaussian elimination in A^TA.

We will take on a somewhat easier task, though, namely showing that for any single subgraph $C = \langle H, r, p, q\rangle$ that has a perfect matching, we can find a matrix A that fits into H and has $M = \langle A, r, p, q\rangle$ nonsingular.

Each nonzero term in the determinant of a square matrix corresponds to a different perfect matching in its bipartite graph. Hence, if C has exactly one perfect matching, then the determinant is nonzero—i.e., M is nonsingular. On the other hand, if C has no perfect matching at all, this means that all terms in the determinant of M are zero, and the matrix is singular.

Lemma 3.7 *Let $H = (V, W, E)$ be a bipartite graph with $|V| \geq |W|$, and let W_1 and W_2 be subsets of W such that $|W_1| = |W_2| = k$, and $|W_1 \cap W_2| \geq k - 1$. If H has the strong Hall property, and if $C = \Phi(H, W_1, W_2)$ has a perfect matching, then there is a spanning subgraph $\widehat{H}$ of H such that $\hat{C} = \Phi(\widehat{H}, W_1, W_2)$ has exactly one perfect matching.*

Proof: Let $C = (S, T, U)$, with $S = \{s_1, \ldots, s_k\}$ and $T = \{t_1, \ldots, t_k\}$. It is convenient to distinguish between two cases:

Case 1, $|W_1 \cap W_2| = k$, i.e., $W_1 = W_2$.

Let $H_1 = (V, W_1, E)$. Now H_1 has the Hall property because H does, so there is a matching F_1 in H_1 that covers W_1. Let $\widehat{H} = (V, W, F_1)$. It is easy to see that any vertex s_i in $\hat{C}$ is adjacent only to t_i. Therefore, $\hat{C}$ has precisely one perfect matching, namely $\{(s_1, t_1), \ldots, (s_k, t_k)\}$.

Case 2, $|W_1 \cap W_2| = k - 1$, i.e., there is a vertex $w_p \in W_1$ such that $w_p \notin W_2$, and a vertex $w_q \in W_2$ such that $w_q \notin W_1$. Without loss of generality, we may assume that $W_1 = \{w_1, \ldots, w_{k-1}\} \cup \{w_p\}$, $W_2 = \{w_1, \ldots, w_k - 1\} \cup \{w_q\}$. For the proof of this second case, we need two more lemmas.

The following lemma is due to Ford and Fulkerson.

Lemma 3.8 [14, Corollary 10.9] *Let $H_1 = (V, W_1, E_1)$ and $H_2 = (V, W_2, E_2)$ be two bipartite graphs with one vertex set in common, and $|W_1| = |W_2| = k$. There is a set $P \subseteq V$ of k vertices, with a matching F_1 in H_1 covering P, and a matching F_2 in H_2 also covering P, if and only if:*

$$\forall X \subseteq W_1 \; \forall Y \subseteq W_2 : |X| + |Y| \leq k + |\Gamma(X) \cap \Gamma(Y)|. \tag{3.1}$$

□

Lemma 3.9 *Let $H_1 = (V, W_1, E(V \cup W_1))$ and $H_2 = (V, W_2, E(V \cup W_2))$, with $|W_1| = |W_2| = k$. If $C = \Phi(H, W_1, W_2)$ has a perfect matching, then there is a matching F_1 in H_1 and a matching F_2 in H_2, both of cardinality k and both covering the same set $V' \subseteq V$, where $|V'| = k$.*

Proof: By Lemma 3.8, it is enough to show that Condition (3.1) holds for H_1 and H_2.

Let $X \subseteq W_1$ and $Y \subseteq W_2$. If $|X| + |Y| \leq k$ then the inequality in (3.1) is immediate. Assume that $|X| + |Y| = k + \delta$, where $\delta \geq 1$. We must prove that $|\Gamma(X) \cap \Gamma(Y)| \geq \delta$. Let $Z = X \cap Y$. Clearly, $\Gamma(Z) \subseteq \Gamma(X) \cap \Gamma(Y)$.

Since X and Y are chosen from sets of size k that have $k-1$ elements in common, we can conclude that $|Z| \geq \delta - 1$. We consider separately the cases $Z = \emptyset$ and $Z \neq \emptyset$.

If $Z = \emptyset$ then $\delta = 1$. Let $\Gamma_C(X)$ denote the adjacency set of X in the graph C. By assumption C has a perfect matching, which by Corollary 2.1 implies that $|\Gamma_C(X)| = |\Gamma(\Gamma(X)) \cap W_2| \geq |X|$. Since $\Gamma_C(X) \subseteq W_2$ and

$$|\Gamma_C(X)| + |Y| \geq |X| + |Y| = k + 1,$$

$\Gamma_C(X)$ and Y must have at least one vertex in common. That is, there must be at least one vertex $w \in Y$ such that $w \in \Gamma_C(X)$, and thus

$$|\Gamma(X) \cap \Gamma(Y)| \geq 1 = \delta.$$

Next, assume that $Z \neq \emptyset$. By the strong Hall property $|\Gamma(Z)| > |Z| \geq \delta - 1$. Then

$$|\Gamma(X) \cap \Gamma(Y)| \geq |\Gamma(Z)| \geq \delta.$$

We have thus shown that condition (3.1) is satisfied, and Lemma 3.9 now follows from Lemma 3.8. $\square$

We continue case 2 of the proof of Lemma 3.7.

Among the matchings F_1 in H_1 and F_2 in H_2 that satisfy the conclusion of Lemma 3.9, choose that pair F_1, F_2 that maximizes $|F_1 \cap F_2|$. Let $\widehat{H} = (V, W, F_1 \cup F_2)$ be the graph with the vertices of the original graph H and the edges of the two matchings F_1 and F_2. We will show that $\hat{C} = \Phi(\widehat{H}, W_1, W_2)$ has exactly one perfect matching. In fact, we will see that $\hat{C}$ consists of a path of odd length plus some isolated edges.

First note that every vertex in $\widehat{H}$ has degree at most 2, and that $\deg(w_p) = \deg(w_q) = 1$, since w_p and w_q are only covered by one matching each.

Let s_i be any vertex in $\hat{C}$. If $\deg(w_i) = 1$ and $i \neq p$ and $i \neq q$, then w_i was matched to the same vertex v_j in both F_1 and F_2. This implies that $\deg(v_j) = 1$ in $\widehat{H}$ and that (s_i, t_i) is the only edge incident on s_i and also the only edge incident on t_i, in $\hat{C}$. In other words, s_i can be matched with exactly one other vertex, t_i.

If $\deg(w_i) = 1$ and $i = p$, so that s_i corresponds to w_p, then some edge $e_1 = (w_p, v_\alpha)$ is in F_1. Some other edge $e_2 = (v_\alpha, w_\beta)$ must be in F_2. Thus, s_i in $\hat{C}$ is adjacent to exactly one vertex, namely t_β. It follows that (s_i, t_β) must be a matching edge in $\hat{C}$.

If we delete e_1 and e_2 from $\widehat{H}$, then we can repeat the argument above, beginning at w_β, to find another edge that must be in the matching on $\hat{C}$. This procedure will continue until $\deg(t_\beta) = 1$. Since the only other vertex of degree 1 that we have not dealt with is w_q, this will happen precisely when we reach w_q.

Finally, if $\deg(w_i) = 2$, and if w_i is not on the path between w_p and w_q, then

w_i must lie on a cycle in $\widehat{H}$. The cycle edges alternate between F_1 and F_2. We can increase $|F_1 \cap F_2|$ by putting every other cycle edge into both F_1 and F_2. This contradicts the assumption that the matchings were chosen to maximize $|F_1 \cap F_2|$.

This completes the proof that there is only one perfect matching in $\widehat{C}$. □

Lemma 3.7 above implies that if the graph $\langle H, r, p, q\rangle$ has a perfect matching, then the special submatrix $\langle A, r, p, q\rangle$ of $A^T A$ is made nonsingular by some assignment of values to A. In fact there is a single assignment of values to A that makes all these special submatrices nonsingular at once.

Lemma 3.10 *Let A be a matrix with t nonzeros, where in place of each nonzero we have put a variable, $x_1, x_2, \ldots, x_t$. Let $M_1, M_2, \ldots, M_k$ be submatrices of $A^T A$. Suppose that for each i with $1 \leq i \leq k$ there is an assignment of values $\mathbf{a}^{(i)} = (a_1^{(i)}, a_2^{(i)}, \ldots, a_t^{(i)})$ to the nonzero entries of A such that M_i is nonsingular. Then there is a single assignment $\mathbf{a} = (a_1, \ldots, a_t)$ with $a_i \neq 0$ for $0 \leq i \leq t$, and such that every M_i, $1 \leq i \leq k$, is nonsingular.*

Proof: Each $\det(M_i)$ is a nonzero polynomial $p_i(\mathbf{x})$. The zeros of such a polynomial form a set of measure zero [13]. Since there are only k such polynomials, the set $S = \bigcup_i \{\mathbf{x} : p_i(\mathbf{x}) = 0\}$ is also of measure zero. Furthermore, the set $T = \{(a_1, \ldots, a_t) : \text{at least one } a_i = 0\}$ is also of measure zero. Hence, the union of S and T is of measure zero, so there must exist a point $\mathbf{a} \notin S \cup T$; so $p_i(\mathbf{a}) \neq 0$ for all i, and $a_j \neq 0$ for $1 \leq j \leq t$. □

Main theorem

With all the necessary lemmas in hand, we are ready for the main theorem of this section.

Theorem 3.4 *Let A be an $m \times n$ matrix, $m \geq n$, with an orthogonal factorization $A = QR$, and such that $\tilde{A}$ has the strong Hall property. Then structural normal equations correctly computes $\tilde{R}$.*

Proof: Recall the two conditions of Definition 3.2 that are required for a structural algorithm to be called correct.

The first of these conditions, that $R \subseteq \tilde{\gamma}(\tilde{A})$, was proved in Theorem 3.3.

For the second condition, that there exist a matrix A' of the same structure as A and such that $R \supseteq \tilde{\gamma}(\tilde{A})$, recall that symbolic factorization of $A^T A$ gives an incorrect result when the symbolic Gaussian elimination does not produce the same structure as that from the actual Gaussian elimination. Furthermore, this happens when cancellation occurs during the actual Gaussian elimination. Hence, we want to show that there is a matrix A', with $\widetilde{A'} = \tilde{A}$, in which no cancellation occurs during Gaussian elimination of $A'^T A'$. By Lemma 3.6, it is enough if we can show that there is a matrix A' such that for each subgraph $C = \langle H, r, p, q\rangle$ that has a perfect matching, $M = \langle A', r, p, q\rangle$ is nonsingular.

Let $C = \langle H, r, p, q\rangle$, $1 \leq r < p \leq q \leq n$, be a subgraph of H that has a perfect matching. By Lemma 3.7, for every such C there is a subgraph H' of H, such that $C' = \langle H', r, p, q\rangle$ has precisely one perfect matching.

Form A' by setting $A'_{ij} = 1$ if (w_i, w_j) is an edge in H', and setting $A'_{ij} = 0$ otherwise. Clearly, $\widetilde{A'} \subseteq \tilde{A}$, and $M' = \langle A', r, p, q\rangle$ is nonsingular—since $C = H(M')$, and hence its determinant has exactly one nonzero term.

Thus, for each subgraph $C = \langle H, r, p, q\rangle$ with a perfect matching, there is a matrix A' such that $M' = \langle A', r, p, q\rangle$ is nonsingular. Finally, by Lemma 3.10, we conclude that there is one matrix, A'', such that for all subgraphs $C = \langle H, r, p, q\rangle$ with a perfect matching, $M = \langle A'', r, p, q\rangle$ is nonsingular. For this matrix A'', no essential cancellation will occur during Gaussian elimination in $A''^T A''$, and the theorem follows. □

Theorem 3.5 *Let A be an $m \times n$ matrix, $m \geq n$, with an orthogonal factorization $A = QR$, and such that $\tilde{A}$ has the strong Hall property. If $\tilde{\alpha}$, $\tilde{\beta}$, and $\tilde{\gamma}$ are the structural versions of QR factorization by Givens rotations, Householder reflections,* *and normal equations, respectively, then* $\tilde{R} = \tilde{\alpha}(\tilde{A}) = \tilde{\beta}(\tilde{A}) = \tilde{\gamma}(\tilde{A})$.

Proof: Follows directly from Theorems 3.3 and 3.4. $\square$

We have thus demonstrated that several common structural algorithms correctly predict the structure of R provided A has the strong Hall property. The case when A does not have the strong Hall property is discussed in Section 5.2.

Chapter 4

Minimizing fill in Gaussian elimination

When a matrix is factored using Gaussian elimination some entries that were zero in the original matrix will become nonzero in the factored matrix. This is known as *fill.* Since one objective in dealing with large sparse matrices is to preserve sparsity, to maintain a manageable size of the matrix, it is natural to want to minimize the fill that occurs. In this chapter we will study this problem as posed on the graph model of the nonzero matrix structure.

4.1 Complexity results

Recall how in Chapter 3 we defined the elimination game on different graph models. The elimination procedures always assumed that the vertices had been previously ordered by some permutation. Different choices of permutations will in general lead to different amounts of fill. The problem then, is to find the permutations that give the least fill.

This way of viewing Gaussian elimination, with pivoting to minimize fill, assumes that such reordering can be carried out before the steps of the Gaussian elimination itself. It also assumes *no numerical pivoting.*

Before we can examine in detail the complexity of the algorithms, it is worthwhile to give a precise definition of the problems. We follow Garey and Johnson's notation for problems [16].

Definition 4.1 Minimum Fill on Undirected Graph (MINFILL).
INSTANCE: Undirected graph $G = (V, E)$, integer $K \leq |V|^2$.
QUESTION: Does there exist a labeling π of the vertices of V, such that when the vertices of G are eliminated in this order the filled graph G^* will have at most K edges?

This question was first answered by Yannakakis:

Proposition 4.1 [57] *MINFILL is NP-complete.*

The problem is known to be solvable in polynomial time on chordal graphs, which are just the graphs on which the elimination game can be carried out without any fill, and outerplanar graphs. See page 52.

The directed version of the problem is analogous:

Definition 4.2 Minimum Fill on Directed Graph (MINDIRFILL).
INSTANCE: Directed graph $D = (V, E)$, integer $K \leq |V|^2$.
QUESTION: Does there exist a permutation π of the vertices of V, such that when the vertices of D are eliminated in this order the filled graph D^* will have at most K edges?

Note that the NP-completeness of this problem is an immediate consequence of the undirected case. However, the directed case was proved several years before the undirected case, by Rose and Tarjan (with a correction by Gilbert).

Proposition 4.2 [51,24] *MINDIRFILL is NP-complete.*

The bipartite graph

As before, the bipartite graph model is more complicated. Recall that on the bipartite graph we can model arbitrary matrix structures as well as nonsymmetric pivoting. It is the latter property that seems to add substantially to the difficulty of the analysis of the problem.

In the bipartite case it is necessary to distinguish between two different versions of the problem.

Definition 4.3 Minimum Fill Vertex Ordering (MFVO).
INSTANCE: Bipartite graph $H = (V, W, E)$, $|V| = |W|$, and a positive integer K.
QUESTION: Do there exist permutations π and τ of the vertices of V and W, such that the fill suffered under these permutations is at most K?

Definition 4.4 Minimum Perfect Elimination Graph Completion (MPEGC). INSTANCE: Bipartite graph $H = (V, W, E)$, $|V| = |W|$, and a positive integer K.
QUESTION: Does there exist a set C of edges, $|C| \leq K$, such that by adding C to H we get a perfect elimination bipartite graph?

The two definitions above are easily seen to be equivalent for both the undirected and the directed graph models; and indeed both versions of the problem statement appear in the literature. On the bipartite graph model, though, they are two different problems. In Figure 4.1 is a bigraph whose minimum completion set is just one edge: by adding the edge (a, b) we get a perfect elimination bigraph. But, no matter what edge we start an elimination with, the graph suffers fill of more than one edge.

Because of the above-mentioned difference the complexity results for the undirected and directed graphs do not carry over immediately. As a consequence, neither MFVO nor MPEGC is known to be NP-complete. It is conjectured however, that indeed both problems are NP-complete.

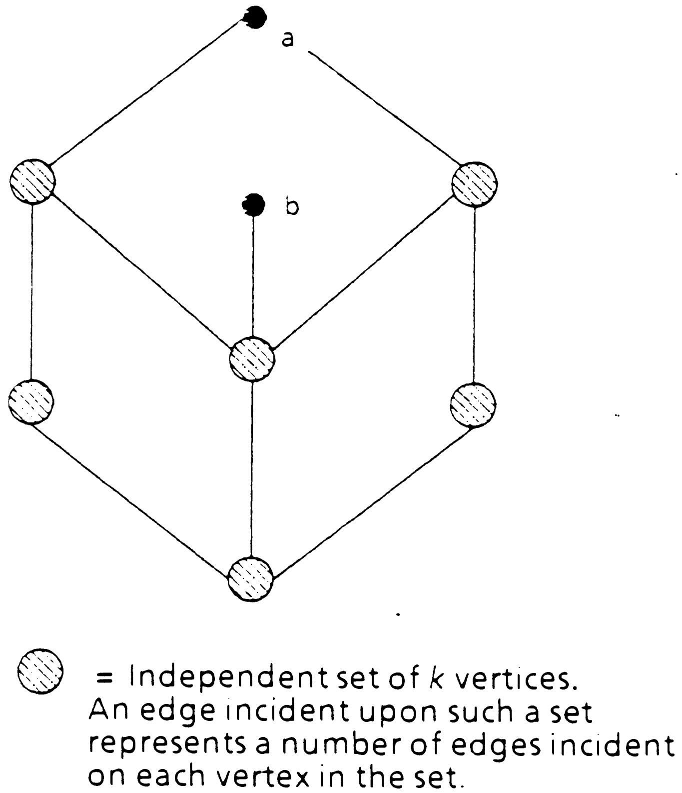


Figure 4.1: A graph on which MFVO≠MPEGC.

A different problem with the bipartite graph model was explained in Section 3.3: the model does not always correctly predict the filled structure. Hence, we may be overly pessimistic in our estimation of fill. By Corollary 3.1 the structural algorithm predicts fill correctly if the original matrix has a full nonzero diagonal. A possible remedy, then, would be to first find a perfect matching in the bipartite graph, and then do the elimination according to a permutation of the matching edges. But this could spoil the possibility of finding the optimal solution, as is shown in Figure 4.2. Here we have a bipartite graph H with a minimum fill of one edge, regardless of whether we use MFVO or MPEGC as our measure. But if we start by finding a perfect matching and then eliminate only matching edges, the smallest possible fill will be $\Theta(n)$ edges.

addition, let ω be an optimal algorithm. To measure the performance of a polynomial time approximation algorithm, then, we compare $\Phi_\alpha(G)$ and $\Phi_\omega(G)$. If we are lucky, maybe we can hope for a good value, i.e., a constant, for the difference: $\delta_\alpha = |\Phi_\alpha(G) - \Phi_\omega(G)|$. Failing to establish such a bound we would next try for a constant bound on the quotient: $\kappa_\alpha = \Phi_\alpha(G)/\Phi_\omega(G)$.

An immediate observation is that there is probably no polynomial time algorithm with a performance that is within an additive constant of the optimal.

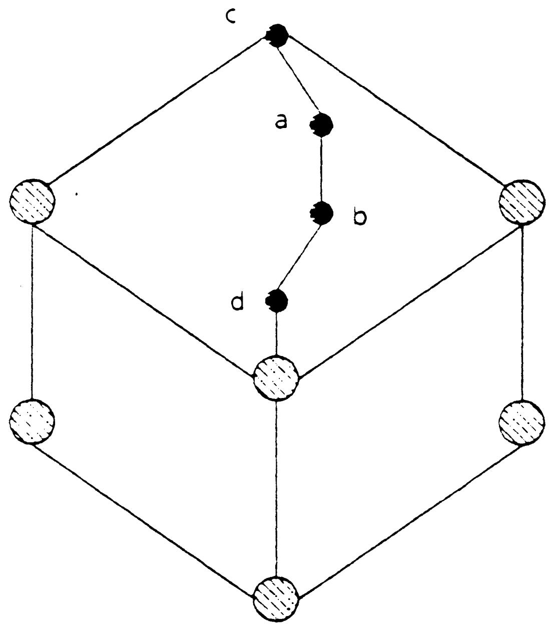
Lemma 4.1 *No polynomial time algorithm α has a constant C such that $\delta_\alpha \leq C$, for all matrices G with $\Phi_\omega(G) \geq N$, unless* P = NP.

Proof: Assume to the contrary that α is such an algorithm, and that P $\neq$ NP. Let G be a graph with $\Phi_\omega(G) = N$. Form a new graph G', consisting of $C + 1$ disjoint copies of G. Then, $\Phi_\omega(G') = N(C + 1)$. By assumption, α can give a fill for G' of $\Phi_\alpha(G') \leq \Phi_\omega(G') + C = (N + 1)C + N$. Hence, at least one the $C + 1$ disjoint copies of G has a fill of only N. This means we could use α to compute the minimum fill in polynomial time, which contradicts the assumption that P $\neq$ NP. $\square$

As for the quotient bound, not much is known. No polynomial time approximation algorithm has been shown to have a good quotient bound, and it has been conjectured that no such algorithm exists [51].

4.2.2 Two common heuristics

Two common heuristics for the MINFILL problem are Minimum Deficiency (MINDEF) [50] and Minimum Degree (MINDEG) [56]. MINDEF means that one always eliminates a vertex of smallest deficiency next, breaking ties arbitrarily. Here the *deficiency* of a vertex v is defined to be the number of edges one has to add to make the neighborhood of v a clique. It is hence a pure "local minimization" tactic.



By eliminating first the edge (a,b) and then the filled in edge (c,d) the total fill will be one edge. If we restrict ourselves to only eliminating original edges of the graph, the fill will be larger.

Figure 4.2: Non-original edges have to be considered.

4.2 Heuristics

Since the general minimum fill problem seems to be difficult, i.e., is NP-complete, the next step will be to look for fast approximation algorithms, or useful heuristics.

4.2.1 Measures of approximation algorithms

A MINFILL algorithm attempts to minimize the number of edges in G^*, the filled graph. The most straightforward way of measuring the performance of an approximation algorithm would be to compare the fill from the approximation algorithm with the optimum value for fill.

The amount of fill suffered in G under the algorithm α is denoted $\Phi_\alpha(G)$. In

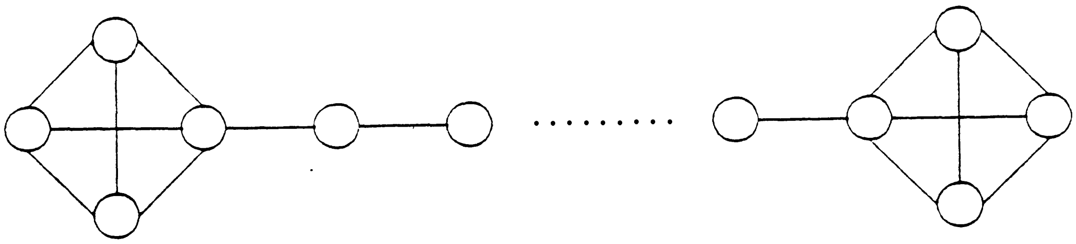


Figure 4.3: A chordal graph on which MINDEG gives $\Theta(n)$ fill.

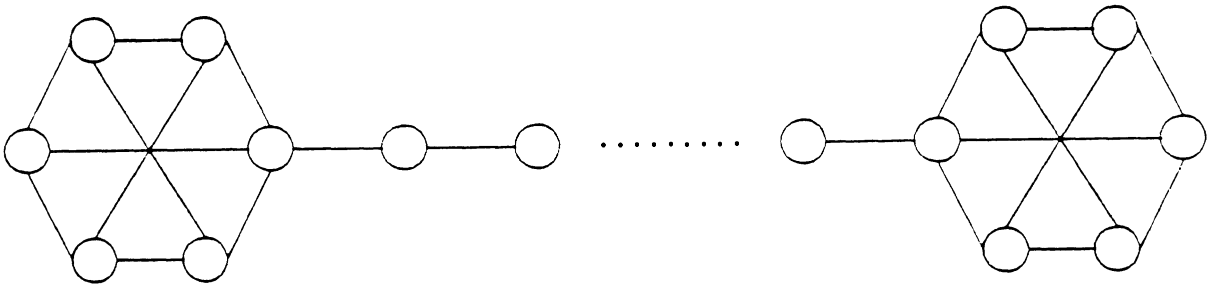


Figure 4.4: A graph with $\Theta(1)$ minimum fill: MINDEF gives $\Theta(n)$ fill.

With MINDEG one always eliminates a vertex of smallest degree next, breaking ties arbitrarily. Of these two heuristics MINDEG is the most widely used; one reson for its popularity is that it can be implemented very efficiently, another is that it performs well in practice.

Neither of the two have good theoretical worst case bounds according to the measure defined above. Figure 4.3 shows a chordal graph (a graph with a minimum fill of zero edges) on which MINDEG always produces $n - 8$ edges of fill. A similar graph, shown in Figure 4.4, has a minimum fill of 6 edges but MINDEF always produces $n - 12$ edges of fill.

It may be argued however, that our measure of performance is unfair. What we really would like to avoid is having a sparse matrix become dense. But if the matrix already is dense, it is not very interesting if an approximation algorithm makes it even denser. This way we can justify a different measure of performance, namely one where we measure not the fill per se, but the the quotient of the number of edges in the filled graphs. Indeed, $\Phi_\alpha(G) + |E|$ is the amount of storage necessary to factor G, so λ_α is a measure of how well α performs in terms of storage.

The filled quotient is denoted

$$\lambda_\alpha(G) = \frac{\Phi_\alpha(G) + |E|}{\Phi_\omega(G) + |E|},$$

where $|E|$ is the number of edges in G. In our previous examples both MINDEG and MINDEF now achieve a constant bound on λ_α.

Two special classes of graphs on which MINDEF performs well are chordal graphs and outerplanar graphs.

Theorem 4.1 *MINDEF gives an optimal fill ordering on chordal graphs and on outerplanar graphs.*

Proof: That MINDEF is an optimal strategy on chordal graphs follows immediately from Lemma 6.2 below. This lemma says that a chordal graph has always at least one vertex of deficiency 0 (called a *simplicial vertex*), and that an ordering without fill can be constructed by repeatedly deleting simplicial vertices.

An outerplanar graph G, is an undirected graph that can be laid out with all vertices on the exterior face and no crossing edges. This is nothing but a triangulated polygon, possibly with some edges missing. See Figure 4.5.

Note that each interior face in an outerplanar graph forms a chordless cycle, and that no edge added to the graph can form a chord in more than one such cycle at a time. Hence, we can bound the minimum fill in the graph from below by summing over all the interior faces. Let ϕ_i denote the number of vertices on interior face number i.

$$\text{MinFill} \geq \Phi_G = \sum_i (\phi_i - 3)$$

We use induction on the number of vertices in G to prove that this is also an upper bound on the fill computed by MINDEF.

The following lemma will be of assistance.

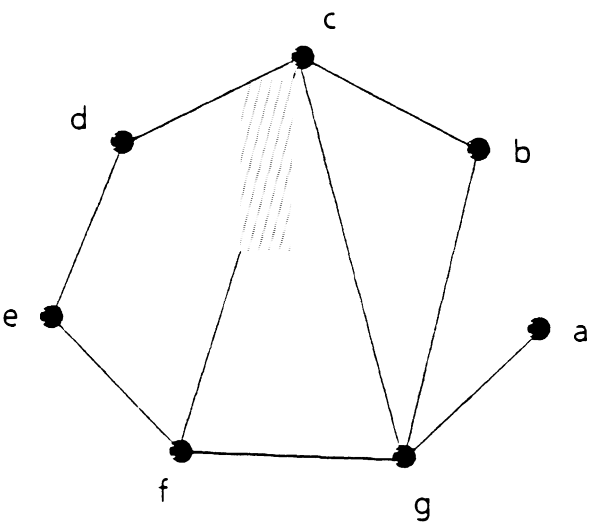


Figure 4.5: An outerplanar graph.

Lemma 4.2 *If G is outerplanar and laid out as described above, then there is always at least one vertex v, such that v lies on at most one interior face.*

Remark: In Figure 4.5 vertex a is on no interior face, vertex b is on one interior face, vertex c is on three interior faces, and so forth.

Proof: For an interior edge (v, w), define the *length* of the edge to be the smallest number of vertices between v and w counted along the "rim" of G. (In Figure 4.5 the length of (c, g) is 2, and the length of (b, g) is 1.)

If no interior edge exists, then the lemma is true for all vertices in G. Otherwise, let (v, w) be an interior edge of minimal length, and let x be a vertex on the shortest rim between v and w. Then, no interior edge can be incident upon x, since such an edge would be strictly shorter than (v, w). Hence, x is on at most one (possibly none) interior face. □

Let G_v be the graph that results from eliminating v from G (in other words, add edges to turn v's neighbors into a clique, and delete v and its incident edges). We now distinguish between two different cases: whether there is a simplicial vertex v, or not.

1. There is a vertex in G of deficiency 0. Clearly, MINDEF will then choose a vertex v of zero deficiency to eliminate first.

   (a) v is on no interior face. In this case, G_v is still outerplanar and has the same interior faces as G. By the inductive hypothesis, the minimum fill on G_v is $\sum_i(\phi_i - 3)$ and is computed by MINDEF. Since this bound is a lower bound for the fill on G, it follows that the minimum fill on G is Φ_G and that MINDEF achieves this bound.

   (b) v is on exactly one interior face, face number 1 say. In this case, the interior face that v is on must be a chordless cycle of size 3. Hence, G_v has one interior face less than G. But, since $\phi_1 = 3$ in G, the sums are still equal: $\Phi_G = \Phi_{G_v}$. Hence, the minimum fill for G is Φ_G, and is achieved by MINDEF.

2. There is no vertex in G of deficiency 0. By Lemma 4.2 there must then be at least one vertex on only one interior face and of deficiency 1. MINDEF must then choose such a vertex v of deficiency 1 to eliminate first.
   If we eliminate v, G_v is still outerplanar. Furthermore, the interior face that v was one has one vertex less than before. Hence, $\Phi_G = \Phi_{G_v} + 1$. By the inductive hypothesis, the minimum fill in G_v is Φ_{G_v} and is achieved by MINDEF. It follows that the minimum fill in G is Φ_G, and is achieved by MINDEF.

□

4.2.3 Divide-and-conquer approaches

As we saw, there are polynomial time algorithms for the MINFILL problem on chordal graphs and on outerplanar graphs. A more important algorithm is *Generalized Nested Dissection* [23,44], which achieves a bound of $O(n \log n)$ fill on planar graphs. The idea of this algorithm is to divide the graph into smaller pieces, such that each piece can be considered independently. Hence, the algorithm can be applied recursively to each piece. To make the desired division, one employs *graph separators*, which are subsets of vertices whose removal disconnects the graph into two or more sections.

An objective then is to find small separators, and to find them quickly. Lipton and Tarjan [45] showed that planar graphs have separators of size $O(\sqrt{n})$ vertices, and that these can be found in polynomial time. This is what gives the bound for GND mentioned above.

Similarly, if for any other class of sparse graphs that is closed under contraction and subgraph, one can demonstrate that it has $\sqrt{n}$-separators that can be found efficiently, then GND will give a bound of $O(n \log n)$ on the fill. Most graphs do not have good separators, however. In fact, most sparse graphs only have separators of size $\Omega(n)$ [44].

An important kind of graph that has especially simple separators is the grid graph (see Figure 4.6). On this type of graph one can use Nested Dissection [18] to compute a fill within a multiplicative constant of the best possible. In practice, however, it is still often preferable to use a simpler algorithm, like MINDEG. This heuristic algorithm, although not asymptotically optimal on even planar graphs, performs well on grid graphs and has even been conjectured to be asymptotically optimal on them. Despite some effort, I have not been able to settle this question.

The simple separator used on the grid graph by an algorithm like nested dissection is simply a row down the middle of the grid, as shown in Figure 4.6.

A different separator for the grid graph, called the *cross separator*, is shown in Figure 4.7. This separator has the advantage of making each vertex be adjacent to only one separator eliminated later than the vertex itself. This in turn makes it a candidate for applications where only part of the matrix can be held in memory at any one time.

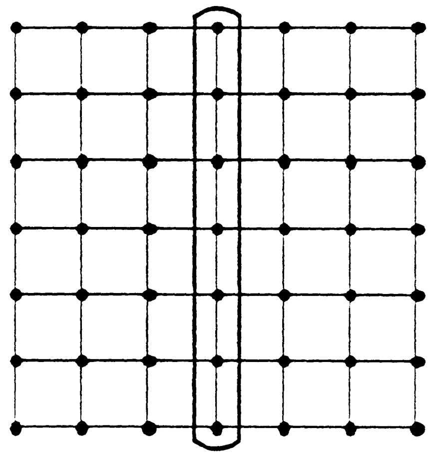


Figure 4.6: A grid graph with a simple separator.

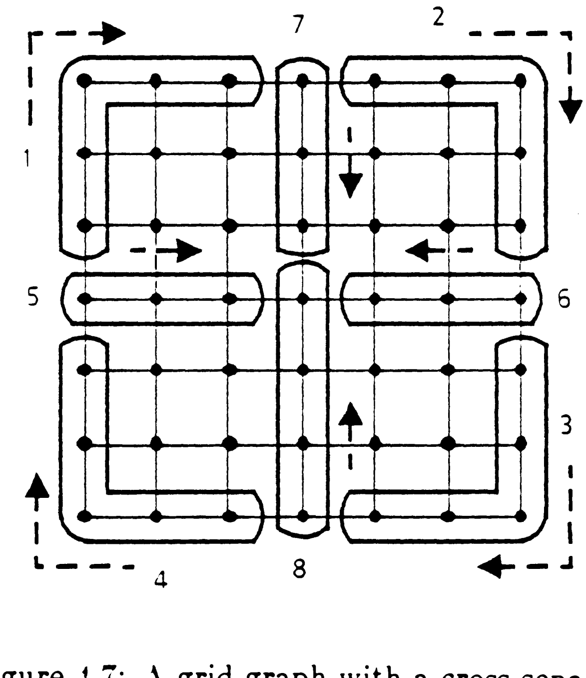


Figure 4.7: A grid graph with a cross separator.

A disadvantage, however, is that the total fill is larger using the cross separator. If the different parts of the separator are eliminated in the order shown in Figure 4.7, then we can compute the fill in the graph precisely. The total fill turns out to be $\frac{119}{8}n^2k + O(n^2)$ edges. This should be compared with the fill using the ordinary row separator on the grid graph, which gives a fill of $\frac{31}{4}n^2k + O(n^2)$ edges. Hence, we get about twice as much fill using the cross separator.

Zmijewski and Gilbert study generalizations of similar separators, and supply experimental results [58].

4.3 More heuristics: Quotient tree partitioning

For certain classes of systems of linear equations $A\mathbf{x} = \mathbf{b}$, a solution can be computed efficiently by partitioning the matrix A into blocks, and solving the system by block factorization. The combinatorial problem we must solve to find a suitable partitioning is equivalent to quotient tree partitioning of undirected graphs. For a detailed description of the background of this problem, see [21].

A *partitioning* of the vertices of G is a division of the vertices into disjoint subsets: $\mathcal{F} = \{V_1, V_2, \ldots, V_p\}$. The *quotient graph* induced by the partitioning, written $G/\mathcal{F}$, is the graph $G' = (\mathcal{F}, \mathcal{E})$ of p vertices where $(V_i, V_j) \in \mathcal{E}$ if and only if there is some edge in E that connects a vertex in V_i with a vertex in V_j. If a quotient graph is a tree we call it a *quotient tree*. In what follows we will refer to vertices of the quotient graph as *nodes*, and reserve the term vertex for vertices of the original graph.

The problem we study is how to find tree partitionings that are "good" in some sense. Just exactly what determines a "good" partitioning depends of course upon our applications. We will suggest three different measures and show that for all three measures the problem of finding a best tree partitioning on an arbitrary undirected graph is NP-complete.

In the following section we will use reductions from the Maximum Independent

Set problem. For our purposes this problem, which is known to be NP-complete [39], is defined as follows:

Definition 4.5 *Maximum Independent Set (MIS).*
INSTANCE: Undirected graph $G = (V, E)$ and an integer $K \leq |V|$.
QUESTION: Does there exist a subset $V' \subseteq V$, such that $|V'| \geq K$ and such that no two vertices in V' are connected by an edge in E?

4.3.1 Complexity of Quotient Tree Partitioning

In this section we investigate the computational complexity of the problem of finding a good quotient tree partitioning. We suggest three different measures and show that for each of them the problem of finding the best quotient tree partitioning is NP-complete.

The first measure is one where we want to maximize the number of nodes in the quotient tree. This is the measure used in an algorithm by George and Liu [20].

Definition 4.6 *Maximum Quotient Tree (MQT).*
INSTANCE: Undirected graph $G = (V, E)$ and an integer $K \leq |V|$.
QUESTION: Does there exist a partitioning $\mathcal{F} = \{V_1, V_2, \ldots, V_p\}$ of V into $p \geq K$ disjoint sets, such that $G/\mathcal{F}$ is a tree?

Theorem 4.2 *Maximum Quotient Tree is NP-complete.*

Proof: By a reduction from Maximum Independent Set.

Given a graph $G = (V, E)$ and an integer K as an instance of MIS, we form an instance of MQT by adding one new vertex r to G and making it adjacent to all the old vertices of G. We claim that this new graph G' has a quotient tree of size $K + 1$ if and only if G has an independent set of size K.

If the size of an independent set in G is K there exists a quotient tree of size $K+1$, since we can take r plus the vertices not in the independent set to be in the root node of the quotient tree, and all the vertices of the independent set will be single-vertex leaves in the quotient tree.

Conversely, suppose there is a quotient tree of $K+1$ nodes in G'. Consider the quotient tree to be rooted at the node that contains r. Since r is adjacent to all other vertices, all the other K nodes in the quotient tree are leaf nodes that are children of the root. Hence, no two vertices that reside in different leaf nodes are adjacent to each other. This means that we can form an independent set of size K by choosing one vertex from each of the K leaf nodes. □

With the maximum number of nodes as our measure of a good quotient tree we may of course find solutions in which a few nodes are very large. This consideration leads to our second measure of a good quotient tree, namely one where we try to minimize the size of each node.

Definition 4.7 *Minimum Nodesize Quotient Tree (MNQT).*
INSTANCE: Undirected graph $G=(V,E)$ and an integer K.
QUESTION: Does there exist a partitioning $\mathcal{F}=(V_1,V_2,\ldots,V_p)$ of V into p disjoint sets, for some $p \le |V|$, such that $|V_i| \le K$, and such that $G/\mathcal{F}$ is a tree?

Theorem 4.3 *Minimum Nodesize Quotient Tree is NP-complete.*

Proof: Again by a reduction from Maximum Independent Set.

Given a graph $G=(V,E)$ and an integer $K \le |V|$ as an instance of MIS we construct an instance of MNQT as follows. First, add a copy of G to get a new graph $G_1=(V_1,E_1)$ that consists of two disjoint copies of G. Clearly, G has an independent set of size K if and only if G_1 has an independent set of size $2K$.

Let $n=|V|$. Now, for each vertex v_i in V_1 construct a clique c_i of size $|c_i| = 4n-1$, and make v_i adjacent to each vertex in its clique. Then, construct a new clique H of size $8n - 1$. Finally, add one more vertex, r, and make this new vertex adjacent to all the other vertices.

Call this new graph G'. It has $2n + 2n(4n - 1) + (8n - 1) + 1$ vertices, so its size is polynomial in the size of G. Setting $K' = 5n - K$ we now claim that G' has a quotient tree partitioning with its largest set of size at most K' if and only if G has an independent set of size at least K.

Observation 1: Let $\mathcal{F}$ be any partitioning that makes $G'/\mathcal{F}$ a tree T, and let R be the set that contains r. Consider T to be rooted at R. Then, any other node in T is a child of R.

Observation 2: Let $\mathcal{F}$ be any partitioning and T and R as above. At least one set of $\mathcal{F}$ must have size $\geq 4n$. This is because the clique $H \cup \{r\}$ can be split over at most two subsets.

Observation 3: There exists a partitioning $\mathcal{F}$ of V' such that all sets in $\mathcal{F}$ have size $\leq 5n$. To see this, let each c_i-clique be in a set by itself, and let all vertices of V_1 plus r plus $3n - 1$ of the vertices of H be in one set, and let the $5n$ remaining vertices of H be in the last set. It is easy to verify that this partitioning induces a tree and that no set is larger than $5n$.

Observation 4: If G has an independent set V_d of size d then there is a partitioning $\mathcal{F}$ which has all sets of size $\leq 5n - d$. A partitioning of this form is constructed as follows: For each vertex v_i of V_d take that vertex and its c_i-clique to form a separate set of size $4n$. Let each of the c_i-cliques of the remaining vertices of V_1 form a separate set of size $4n - 1$. Finally, form a root set R containing r, the $2n - 2d$ vertices of V_1 that are not in V_d, and $3n + d - 1$ of the vertices of H. The remaining $5n - d$ vertices of H will be in a separate set. The total size of R is then $2n - 2d + 3n + d - 1 + 1$ which is equal to $5n - d$. Since $d \leq n$ each set of size $4n$ is also $\leq 5n - d$.

Observation 5: Let $\mathcal{F}$ be a Minimum Nodesize partitioning of G'. No set in the partitioning, except R, can contain two or more vertices from V_1. Assume that some set S other than R does contain two vertices from V_1. Then their c_i-cliques have to be split between being in S and being in R. Now, $|S| \leq 5n$ by observation 3 above and the fact that $\mathcal{F}$ is a Minimum Nodesize partitioning. Hence, at least $8n - 5n = 3n$ of the vertices of these cliques must fall in R. But then, since $|H| + 1 + 3n = 11n$ we must have that either R or the part of H outside of R is of size $\lceil 11n/2 \rceil$, which is greater than $5n$. This contradicts the assumption that $\mathcal{F}$ is a Minimum Nodesize partitioning.

Assume that there exists a partitioning of G' which is a MNQT partitioning and in which every set is of size $\leq p$. By observations 2 and 5 we can assume that every vertex from V_1 that is not in R is in a set containing only itself and vertices from its clique, forming a set of size $4n$. Furthermore, the c_i-cliques of all V_1-vertices must be outside R, forming sets of size $4n - 1$. Let S_H be the set formed by the vertices of H that are not in R. By assumption $|S_H| + |R| \leq 2p$ and among the vertices of $S_H \cup R$ are the $8n$ vertices of $H \cup \{r\}$. Thus, the number of vertices of V_1 in R is at most $2p - 8n$. Hence, at least one of the identical copies of the original graph has at most $p - 4n$ vertices in R. This means that for this graph we have an independent set of size at least $n - (p - 4n) = 5n - p$.

It follows from the observation above that G' has a Minimum Nodesize Quotient Tree partitioning with each set $\leq p$ if and only if G has an independent set of size $\geq 5n - p$. $\square$

We come to our third, and last, measure of a good quotient tree. Here we want to minimize the sum of the squares of the set sizes in the partitioning. In the context of matrix block-factorization this is the area of the diagonal blocks; hence, it is an upper bound on the space required for these blocks. Not surprisingly, this measure does not make the problem any easier.

Definition 4.8 *Minimum Sum of Squares Quotient Tree (MSSQT).*
INSTANCE: Undirected graph $G = (V, E)$, integer $K \leq |V|^2$.
QUESTION: Does there exist a partitioning $\mathcal{F} = (V_1, V_2, \ldots, V_p)$ of V into p disjoint sets, for some $p \leq |V|$, such that $G/\mathcal{F}$ is a tree and such that $|V_1|^2 + |V_2|^2 + \cdots + |V_p|^2 \leq K$?

Theorem 4.4 *Minimum Sum Squares Quotient Tree is NP-complete.*

Proof: By a reduction from Maximum Independent Set.

Given a graph $G = (V, E)$ and an integer $K \leq |V|$ as an instance of MIS we construct an instance of MSSQT as follows. Let $n = |V|$ and $s = \lceil \sqrt{n} \rceil$. For each vertex v_i in V make a clique c_i of size s, and make v_i adjacent to each vertex in its clique. Also, add an additional clique r of size s, and connect each vertex in r to all other vertices in the graph. The graph so constructed, $G' = (V', E')$, has $ns + n + s$ vertices—hence its size is polynomial in the size of G.

Setting $K' = K(s+1)^2 + (n-K)s^2 + (s+n-K)^2$, we claim that G' has a quotient tree partitioning with the sum of the squares of the set sizes at most K' if and only if G has an independent set of size K.

Let $\mathcal{F}_0$ be the partitioning $(r \cup V, c_1, \ldots, c_n)$. The cost of $\mathcal{F}_0$ is

$$C_0 = (s+n)^2 + ns^2 = (n+1)s^2 + 2ns + n^2.$$

Keeping this in mind, we can make the following claim:
Claim: In any optimal MSSQT partitioning, r lies completely within one set. Assume that we have a quotient tree partitioning of V' where two sets, S_1 and S_2, both contain vertices of r. Since each vertex of r is adjacent to every other vertex in V', there can be no other sets in the partitioning. Restricted to two sets, we get an optimal partitioning by making S_1 and S_2 equal in size. Hence the cost of this partitioning will be at least $2(\lfloor (ns+n+s)/2 \rfloor)^2$, which is greater than C_0 for $n \geq 2$.

Let $\mathcal{F}$ be any optimal MSSQT partitioning and let R be the set that contains r. If we consider the tree induced by $\mathcal{F}$ as rooted at R, then this tree has height at most 1.

Let S be a set in $\mathcal{F}$, $S \neq R$. If S contains part of a clique c_i, then v_i is either in S or in R. If v_i is in R we can construct a better partitioning by placing the part of c_i that is in S in a separate set. Hence, we can assume that each set S contains vertices v_i and parts of their cliques c_i. For each v_i its c_i-clique is either completely within R, completely within S, or partly within R and partly within S. We can assume without loss of generality that at most one vertex v in S has its clique partly in both R and S.

Assume that some vertex v in S has its clique completely within R. We construct a new partitioning $\mathcal{F}_1$, and claim that $\mathcal{F}_1$ is as least as good as $\mathcal{F}$. Remove the c-clique from R and let it form its own set, and move v from S to R. Clearly, this is still a quotient tree partitioning. We compare the costs by which these sets contribute to the total costs of the partitionings. Let $x = |R| - s$, $y = |S| - 1$, and let δ be the difference in cost between the old and the new partitioning. We get: $\delta = (x+s)^2 + (y+1)^2 - (x+1)^2 - s^2 - y^2 = 2sx + 2y - 2x \geq 0$.

Let v be a vertex in S whose c-clique straddles R and S. Let a be the number of vertices of the clique in R, and let b be the number of vertices of the clique in S. Also, let $x = |R| - a$, $y = |S| - b - 1$. We construct a new partitioning $\mathcal{F}_2$ by moving v from S to R and letting the c-clique be in a separate set. Then, $\delta = (x+a)^2+(y+b+1)^2-(x+1)^2-(a+b)^2-y^2 = 2x(a-1)+2b-2ab+2y(b+1)$. Since R contains all of r, $x \geq s = a+b$ implies $\delta \geq 2(a+b)(a-1)+2b-2ab+2y(b+1) = 2a(a-1)+2y(b+1) \geq 0$.

This means that we now have a partitioning where R does not contain any part of any c-clique. Hence, S contains k vertices of V and all of their c-cliques. Assume $k \geq 2$ and let v be a vertex in S. Let $x = |R|$, $y = |S| - (s+1)$. Construct a new partitioning as follows: move v from S to R and let its c-clique be in a separate set. We have $\delta = x^2 + (y+s+1)^2 - (x+1)^2 - s^2 - y^2 = 2y(s+1) + 2s - 2x$. Since $k \geq 2$, $y \geq (s+1)$. This implies $\delta \geq 2(s+1)^2 + 2s - 2x = 2(s^2 + 3s + 1 - x)$. Furthermore, since R contains only vertices from r and V, $x \leq s + n$; and, by definition, $s^2 \geq n$. So, $\delta \geq 2(s^2 + 3s + 1 - s - n) \geq 2(2s + 1) \geq 0$.

We can finally conclude that the optimal value of a MSSQT partitioning can be achieved through a partitioning of the following kind: we have a set R that contains r and some vertices of V; for each vertex v in R, its c-clique lies in a separate set; for each vertex v not in R, v is in a set that contains precisely v and its c-clique.

Assume that there are d vertices of V not in R. Clearly, from a partitioning of this kind we can find an independent set of size d in G. The cost of this partitioning is $C = (s + n - d)^2 + (n - d)s^2 + d(s + 1)^2$.

Conversely, if G has an independent set of size d, we can easily construct a partitioning of the type and cost described above. □

4.3.2 Maximal Quotient Trees

The NP-completeness of the problems discussed above implies that we probably cannot find polynomial time algorithms for MQT. Instead, we have to turn to approximation algorithms. One possibility then is to look for a maximal quotient tree rather than a maximum quotient tree.

A *maximal quotient tree* is a quotient tree partitioning in which we cannot partition any node further and still have a tree. A maximal quotient tree is not necessarily a maximum quotient tree, and can in fact be arbitrarily far from optimum. George and Liu [20] present an algorithm that will find a maximal quotient tree partitioning for an undirected graph. They offer no estimate of the running time of their algorithm, but an analysis of the implementation in [21] indicates that it has a worst case running time of $\Omega(n^2)$, where n is the number of vertices in the original graph. We present in Figure 4.3.2 an algorithm which is a different implementation of the same idea, and has a worst case running time of $O(m)$, where m is the number of edges in the graph.

The idea is as follows: We start with a level structure of G. Letting each level be a subset in the partitioning will certainly give us a quotient tree. However, to get a maximal quotient tree we refine the partitioning within each level, according to the following rule:

> Partitioning Rule: *Vertices v,w on level k are put in the same node if and only if there is a path from v to w going only through vertices on levels greater than or equal to k.*

George and Liu[20] prove that if we partition the vertices of G according to this rule the result will be a maximal quotient tree.

In the first part of the algorithm we use a breadth first search to find a level structure for G. This part of the algorithm (not explicitly shown in the figure), returns an array of lists of vertices. Furthermore, for each vertex v in the graph we record in the field Level[v] the level of v.

In the second part of the algorithm we process the vertices looking at one level at a time. The first part of the body of the main loop builds an auxiliary graph H, whose vertices are the vertices on the present level plus the nodes of the next higher level.

In building the auxiliary graph we include all the edges that the vertices of H induce. To avoid adding an edge to H more than once, we take two precautions: First, if while processing vertex v on level k, we see an edge (v, w) within the level, we only add it if $w < v$. Second, assume that (v, w) is an interlevel edge, i.e., w is in a node u on level $k + 1$. We add the edge (v, u) to H and set the field Latest[u] to v. Now, if v should be adjacent to any other vertex in node u, we can detect that this edge has already been added simply by checking the value of Latest[u].

MAXIMAL QUOTIENT TREE ALGORITHM:
Input: A connected graph $G = (V, E)$.
Output: A maximal quotient tree partitioning of V as given by the array Node.

```
1.  Use a breadth first search to find a level structure for G.

2.  for i ← 1 to n do Latest[i] ← 0; od

3.  for k ← maxlevel to 1 do
4.      U ← {vertices on level k} ∪ {nodes on level k + 1};
5.      H ← (U, ∅);
6.      for each vertex v on level k do
7.          for each w ∈ Adj[v] do
8.              if Level[w] = Level[v] and w < v then
9.                  H ← H ∪ (w, v);
10.             else if Level[w] > Level[v] then
11.                 p ← Node[w];
12.                 if Latest[p] ≠ v then
13.                     H ← H ∪ (p, v)
14.                     Latest[p] ← v;
                    fi
                fi
            od
        od
15.     Determine the connected components of H,
        using a depth first search.
        The level-k vertices of each connected component of H
        form a new node:
16.         for all vertices v in node p do Node[v] ← p; od
    od
```

Figure 4.8: Algorithm for finding a maximal quotient tree.

Having built H, we find its connected components using a depth first search (not explicitly shown in the figure). Each new node in the quotient tree is made up of the level k vertices of some connected component of H.

Theorem 4.5 *The algorithm in Figure 4.3.2 computes a maximal quotient tree in time $O(m)$.*

Proof: We first claim that vertices v,w on level k are in the same connected component of H if and only if there is a path from v to w going only through vertices of levels greater than or equal to k.

This is obvious for vertices on $\mathit{maxlevel}$. So, assume it is true for vertices on levels $k+1, ..., \mathit{maxlevel}$, and consider level k.

If v and w are in the same component, then obviously there is a path from v to w in H; and the path goes only through vertices of level k and nodes on level $k+1$. If v_i is a node on the path, then v_{i-1} and v_{i+1} must both be vertices on level k. Let (v_{i-1}, u) and (u', v_{i+1}) be the edges in G, where u,u' are in the same node p on level $k+1$. By induction hypothesis there is a path of the desired kind from u to u'. Hence there is a path from v to w going only through vertices on levels greater than or equal to k.

For the other direction, assume that a path $v = v_1, v_2, ..., v_l = w$ exists, where each v_i is on level k or higher. Let v_i be the first vertex on the path that is on level $k+1$. (If no such vertex exists, the path is completely within level k, and we are done.) Let v_{j+1} be the first vertex after v_i that is on level k. By inductive assumption, v_i and v_j are in the same node on level $k+1$, say node p. This means that (v_{i-1}, p) and (p, v_{j+1}) are both edges in H. By repeating this argument we can reduce the path in G from v to w to a path going only through vertices on level k and nodes on level $k+1$. It follows that v and w will be connected in H, and hence will be in the same node on level k.

We have proved that the algorithm makes a partitioning according to the Partitioning Rule above. As proved by George and Liu [20], this implies that the algorithm correctly computes a maximal quotient tree.

For the running time of the algorithm first note that determining the level structure of G can be done in time $O(m)$, since this is the running time of a breadth first search.

In the part where we construct the auxiliary graphs each vertex of G is considered exactly once as v in the loop on line 6. For each adjacent vertex w of v we make a constant number of operations. Hence, the building of all the auxiliary graphs takes at most time $O(m)$.

Finally, it suffices to note that each edge that is added to an auxiliary graph corresponds uniquely to some edge in G. Hence, the total number of edges in all the auxiliary graphs is less than or equal to m. Since depth first search takes time linear in the number of edges, the last part of the algorithm can also be done in time $O(m)$. It follows that the total worst case running time of the algorithm is $O(m)$. □

Chapter 5

Minimizing fill in QR factorization

Let A be an $m \times n$ matrix, with $m \geq n$. As described in Section 3.4, it is of interest to find the factor R in a QR factorization. And, just as with Gaussian elimination, it is important to keep the fill in R to a minimum. We again assume that A has full numeric rank (and hence full structural rank).

5.1 Complexity results

As in Chapter 4 we will study the problem of minimizing fill on a graph model. In this case, since the matrix is in general rectangular, our graph model will be the bipartite graph. Numerically, when the factor is computed using the normal equations method, we note that $A^T A$ is independent of any row permutations in A. Hence, a minimum fill algorithm need only concern itself with looking for a good column permutation of A. And since any numerical algorithm produces the same factor, this is also true for the other numeric algorithms.

Structurally, things are a little bit different. Recall from Chapter 3 that the different structural algorithms do not always produce the same structure for the factor. Structural normal equations, however, does have the same property as its numeric counterpart—that the structure of $\tilde{A}^T\tilde{A}$ is independent of row permutations in $\tilde{A}$. But since the different structural algorithms for QR factorization may produce different results, we have to always make clear which structural algorithm we are talking about.

Finding the optimal column permutation for structural normal equations, amounts to nothing more than finding the optimal symmetric permutation in the square matrix A^TA. Hence, structural normal equations is NP-complete since MINFILL on an undirected graph is.

Theorem 5.1 *Minimum fill for structural normal equations is NP-complete.*

Proof: We need the following lemma:

Lemma 5.1 *For any symmetric matrix structure $\widetilde{M}$ with a nonzero diagonal, there is a structure $\tilde{A}$ such that $\widetilde{M} = \tilde{\nu}(\tilde{A})$.*

Proof: Let $\widetilde{M}$ be an $n \times n$ symmetric matrix structure with a nonzero diagonal. We construct a new matrix structure $\tilde{A}$ as follows. First, set $\tilde{a}_{ii} = 1$ for $1 \leq i \leq n$. These entries will assure that $\widetilde{m}_{ii} \neq 0$.

Next, for each nonzero position $\widetilde{m}_{ij}$ in $\widetilde{M}$, with $i < j$, add a new row to $\tilde{A}$, k say, and set $\tilde{a}_{ki} = \tilde{a}_{kj} = 1$. All other entries in $\tilde{A}$ are set to zero. This way any two columns of $\tilde{A}$ will have at most one row in common, and so $\widetilde{M} = \tilde{A}^T\tilde{A}$. □

Now, if we could find an optimal column ordering permutation for A, then we could find an optimal symmetric ordering for $\widetilde{M}$. Since the latter problem is NP-complete, the former is also. □

What about one of the other structural algorithms? Let us look at structural Givens factorization; the treatment of structural Householder factorization is similar. In fact, if we have a structural algorithm δ that forms an exact pair with the numerical algorithm QR factorization, then what follows below holds also for δ.

With structural Givens factorization it is not necessarily true that the resulting factor is independent of row permutations in $\tilde{A}$. To find the factor with smallest fill we may have to consider both row and column permutations. However, for any given pair of permutations the structure of the factor is always a substructure of the structure computed by structural normal equations—as was shown in Theorem 3.3. Furthermore, as was shown in Theorem 3.5, when $\tilde{A}$ has the strong Hall property the structure computed by structural Givens factorization is equal to the structure computed by structural normal equations. These observations lead to the following theorem.

Theorem 5.2 *Minimum fill for structural Givens factorization (structural Householder, δ) is NP-complete.*

Proof: The proof is by a reduction from minfill for structural normal equations. Given a structure $H = H(A)$ for which we want to minimize fill under normal equations factorization, we can construct a new graph H' which has the strong Hall property and for which the product-bipartite graph $\Phi(H', W', W')$ is equal to that of $\Phi(H, W, W)$. This is done by adding n new row-vertices, one for each column in A, and connecting one such new vertex to each of the column-vertices.

Since no two columns will be adjacent to the same new row-vertex, this does not change the structure of the product-bipartite graph. Hence applying structural normal equations to H' must give the same result as applying it to H. And since H has full structural rank it is easy to see that H' will have the strong Hall property. So by Theorem 3.5 structural Givens factorization on H' also gives the same result as structural normal equations on H, and the theorem follows. □

5.2 An algorithm for QR factorization

The Dulmage-Mendelsohn reordering of a matrix

As we have seen, the problem of minimizing fill in the upper triangular factor R of a QR factorization is NP-complete; and what is more, we do not have any efficient algorithm for predicting the structure of R unless A has the strong Hall property. However, if the structure of A does not have the strong Hall property, we can permute A into parts, each of which has the desired property. And from this reordered matrix A' our structural algorithms can correctly predict the structure R'. This canonical reordering scheme was first studied by Dulmage and Mendelsohn [10,11,12,38].

The reordering we want is what is known as "block upper triangular form" [29]. It is a permutation of the columns and rows of A that leaves square submatrices on the diagonal, and a rectangular submatrix in the lower right-hand corner. See Figure 5.1.

As we pointed out in Section 2.3.3, the strong Hall property is closely related to the strong connectedness of a directed graph. In particular, if A is a square matrix we can do our reordering by finding the strongly connected components in the directed graph that corresponds to A [29,54]:

1. Find a perfect matching in A and, with this matching fixed, construct the directed graph $D(A)$.
2. Find the strongly connected components of D in topological order.

Regardless of which perfect matching we choose in Step 1, Step 2 will give the same row partition and the same column partition.

If A is not square, we can proceed as follows. First find a matching in A that covers all the columns. Then the last, rectangular, component of A consists of every column and row that can be reached from an uncovered row via an alternating path.

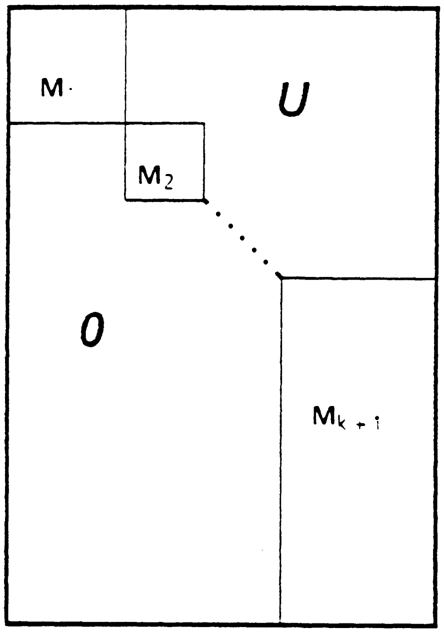


Figure 5.1: The Dulmage-Mendelsohn reordering of a matrix.

The remaining rows and columns are a square submatrix that can be reordered into strongly connected components in topological order, as above. The rectangular component and each of the square strongly connected components have the strong Hall property. The rows and columns in each component are independent of which maximum matching in A we use.

The worst case time complexity of finding a maximum matching in a bipartite graph is bounded by $O(t\sqrt{m+n})$ [34,46], if A is an $m \times n$ rectangular matrix with t nonzeros. The rectangular component can be found by depth first search in $O(t)$ time. The strong components can also be found in $O(t)$ time [1,54]. (In fact, the same depth first search can be used to find the rectangular component and the strong components.) Thus, the worst case time for the reordering is bounded by $O(t\sqrt{m+n})$. Duff's experiments with a heuristic maximum matching algorithm [9] indicate that in practice a maximum matching can often be found in $O(t)$ time; in such cases, the entire reordering takes $O(t)$ time.

Theorem 5.3 *Let A be an $m \times n$ rectangular matrix. Use the algorithm above to reorder the rows of A according to the Dulmage-Mendelsohn order, giving A'. Now, structural Givens factorization on A' correctly predicts the structure of its factor R'.*

Proof: Let M be any of the square submatrices on the diagonal. If there are none, A itself has the strong Hall property, and the theorem follows from Theorem 3.5.

Let $\mathbf{c}_i, \ldots, \mathbf{c}_j$ be the columns that are in M, and let $\mathbf{c}_k$ be any other column in A' to the right of M. If $\mathbf{c}_k$ has only zeros in rows $i, \ldots, j$, then the factoring of M does not affect the structure of $\mathbf{c}_k$ in these rows.

If $\mathbf{c}_k$ has at least one nonzero in rows $i, \ldots, j$, consider the rectangular submatrix, U, made up from columns $\mathbf{c}_i, \ldots, \mathbf{c}_{j-1}, \mathbf{c}_k$. Since column $\mathbf{c}_k$ has a nonzero in row k, U has the strong Hall property. Hence, by Theorem 3.5, structural Givens factorization works on this matrix, and it follows that structural Givens factorization works correctly on the whole of A'. $\square$

Block back substitution

Although the result in Theorem 5.3 is interesting in its own right, it turns out that in practice it is probably better to solve the system by "block back substitution"—and never compute the structure of R. This is done as follows. (Refer to Figure 5.2 for notation.)

BLOCK BACK SUBSTITUTION:
1. Reorder A as shown in Figure 5.2, with square diagonal blocks $M_1, M_2, \ldots, M_k$ and a rectangular diagonal block M_{k+1}.
2. Solve the rectangular least-squares system $M_{k+1}\mathbf{x}_{k+1} \simeq \mathbf{b}_{k+1}$.
3. **for** $i \leftarrow k$ **downto** 1 **do**
4. Update the vector $\mathbf{b}$, by setting $(\mathbf{b}_1, \ldots, \mathbf{b}_i) \leftarrow (\mathbf{b}_1, \ldots, \mathbf{b}_i) - U_{i+1}\mathbf{x}_{i+1}$.
5. Solve the square system $M_i\mathbf{x}_i = \mathbf{b}_i$.

od

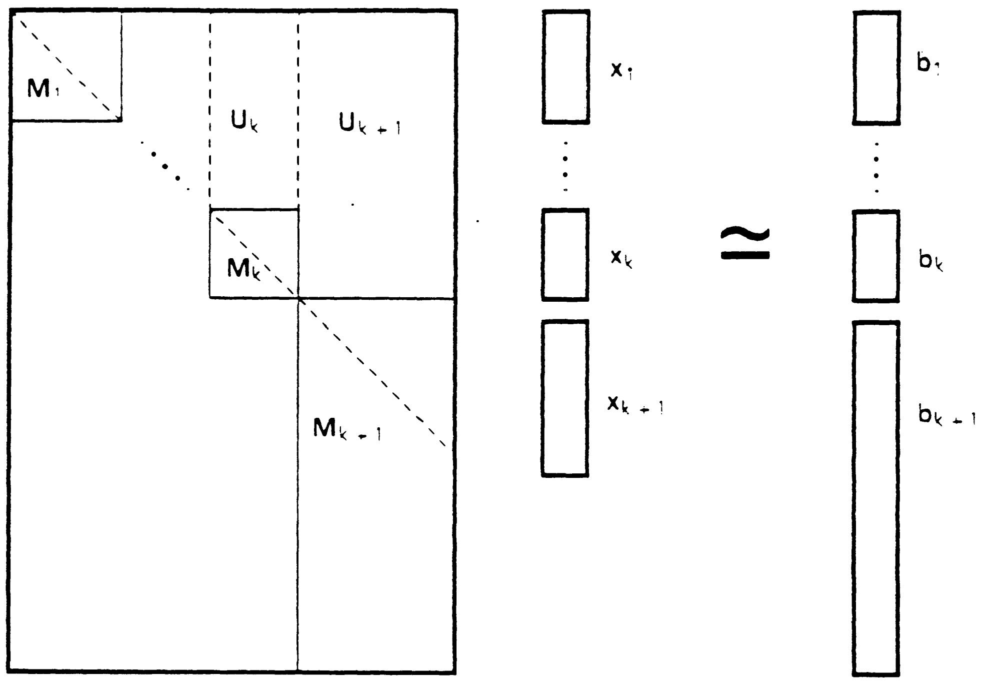


Figure 5.2: Block back-substitution.

This way of solving the system has several advantages. First, in Step (2), we know that M_{k+1} has the strong Hall property, so by Theorem 3.5 we will not use any unnecessary space—whether we use structural normal equations on A^TA or structural Givens factorization. In case we use structural normal equations on A^TA, note that the strong Hall property is preserved under column and row permutations. This means we can use a heuristic, like minimum degree, on the structure of A^TA to find a column ordering that gives small fill, and still be guaranteed that we are not allocating any extra space for R.

Second, all the systems we solve in Step (3) are square, which means that we can use any standard algorithm to solve these, and to find orderings that give little fill. Of course, one way to solve the square system $M_i\mathbf{x}_i = \mathbf{b}_i$ is by a QR factorization of M_i [32]. If we do this, Theorem 3.5 says that structural Gaussian elimination on A^TA correctly predicts the structure of M_i's triangular factor. Structural Gaussian elimination on A^TA can also be useful when solving $M_i\mathbf{x}_i = \mathbf{b}_i$ by Gaussian elimination with partial pivoting, as shown by George and Ng [22].

Third, by updating the vector $\mathbf{b}$ every time we have determined new values for some variables in $\mathbf{x}$, we avoid having to store any fill at all in U, the part of $\boldsymbol{A}$ above the diagonal blocks.

It would be of interest to study how matrices that arise in practice are partitioned, using the algorithm above. One series of experiments indicate that least squares problems from geodetic applications usually have the strong Hall property. Experiments are planned for problems that arise in optimization settings [25].

Chapter 6

Chordal graphs

An undirected graph is said to be *chordal* if every cycle of length at least four has a *chord*, which is an edge joining two vertices that are not adjacent on the cycle. Chordal graphs are perfect; that is, every induced subgraph of a chordal graph has a clique covering and an independent set of the same size [30]. Much of the basic theory of chordal graphs was developed by Dirac [8] and Fulkerson and Gross [15]. Chordal graphs have also been called triangulated graphs, monotone transitive graphs, rigid circuit graphs, and perfect elimination graphs. The latter name derives from the fact that a graph is chordal if and only if the elimination game can be carried out in some order without fill.

Many divide-and-conquer algorithms on graphs are based on finding a small set of vertices or edges whose removal divides the graph roughly in half. An important example is efficient sparse Gaussian elimination [23,44], as we pointed out in Section 4.2.

Most graphs do not have small separators that divide them evenly in half, but some useful ones do. An example is planar graphs [45], as was mentioned in Section 4.2. Other kinds of graphs that can be separated evenly by deleting $o(n)$ vertices are trees ($O(1)$ vertices [36,43]), outerplanar graphs ($O(1)$ vertices [41]), hypercubes

$(O(n/\sqrt{\log n})$ vertices [23]), graphs of genus at most g $(O(\sqrt{gn})$ vertices [26]), and several interconnection graphs for parallel computation [37,40,41]..

Since a complete graph is chordal and has only trivial separators, chordal graphs in general cannot be separated by removing $o(n)$ vertices. But chordal graphs do satisfy a separator theorem in which the size of the separator depends on the density of the graph. A chordal graph with n vertices and m edges has a set of $O(\sqrt{m})$ vertices whose removal leaves no component with more than $n/2$ vertices. (This is immediate at the extremes of density, for complete graphs and for trees.)

A *simplicial vertex* of a graph is a vertex that has a null deficiency, so it can be eliminated without fill; thus, it is a vertex whose neighbors form a clique. A graph G is a *perfect elimination graph* if its vertices can all be eliminated in some order without any fill. Such an order is called a *perfect elimination ordering* of the vertices of G.

Lemma 6.1 [15] *A graph G is chordal if and only if it is a perfect elimination graph.*

A perfect elimination ordering must start with a simplicial vertex. Any simplicial vertex will do, and a choice of simplicial vertices is always available.

Lemma 6.2 [8,42] *If G is chordal and C is any complete proper subgraph then there is a simplicial vertex in $G - C$. Any simplicial vertex can be eliminated first in some perfect elimination ordering.*

In an ordered graph, a path is a *monotone path* if the indices of its vertices are strictly increasing.

Lemma 6.3 [49] *Fix a perfect elimination ordering $v_1, \ldots, v_n$ for a chordal graph G. If $k < h$ and there is a path from v_k to v_h through vertices numbered at most h, then there is a monotone path from v_k to v_h.*

A *separation clique* is a complete subgraph whose removal leaves a disconnected graph.

Lemma 6.4 [2] *If G is chordal and not complete, then G has at least one separation clique.*

6.1 A $\sqrt{m}$-vertex separator theorem

Let G be a chordal graph with n vertices and m edges. Suppose that each vertex of G has a nonnegative *weight*, and that the sum of the weights is n. Then there is a clique that divides the weight roughly in half.

Theorem 6.1 *Let G be a weighted chordal graph as above, with p vertices in its largest clique. Then G contains a clique whose removal leaves no connected component of weight more than $n/2$.*

Proof: The idea of the proof is to start with an arbitrary clique and make it ooze around the graph like an amoeba until it is an $n/2$ separator. It oozes by disgorging vertices that can join or become components of weight less than $n/2$, and by engulfing vertices that are in a component of weight more than $n/2$.

Here are the details. We will not distinguish between a set of vertices of G and the subgraph of G it induces. Unless G is empty, it has at least one clique. Let C be the clique that minimizes the maximum weight of a connected component of $G-C$. In case of ties, minimize the number of vertices in a maximum-weight component of $G-C$. If ties remain, minimize the number of vertices in C. If ties still remain, choose arbitrarily.

Assume for the sake of contradiction that $G-C$ has a component A of weight greater than $n/2$. Then the total weight of $G-A$ is less than $n/2$. We shall state and prove three facts about A and C.

Fact 6.1 *Every vertex of C is adjacent to some vertex of A.*

Proof: If $v \in C$ were not adjacent to any vertex of A, then $C \setminus \{v\}$ would have been chosen in preference to C. □

Fact 6.2 *If B is a nonempty subset of A, then B contains a vertex that is simplicial in $B \cup C$.*

Proof: Immediate from Lemma 6.2. □

Fact 6.3 *Component A contains a vertex v adjacent to every vertex of C.*

Proof: We will take v to be the last vertex of A in a perfect elimination ordering of $A \cup C$ with C ordered last. Thus $v = a_k$ where $\{a_1, \ldots, a_k, c_1, \ldots, c_h\}$ is a perfect elimination ordering of $A \cup C$. Such an ordering exists because by Fact 6.2 we can repeatedly choose simplicial vertices that are not in the clique C.

Let x be a vertex of C. Since A is connected and (by Fact 6.1) x is adjacent to a vertex of A, there is a path from x to a_k in $A \cup C$ that uses only vertices of $A - \{a_k\} = \{a_1, \ldots, a_{k-1}\}$. Lemma 3.1 says that if $\{x, a_k\}$ is not an edge of $A \cup C$ then it is a fill edge. But a perfect elimination ordering has no fill, so x is adjacent to a_k in $A \cup C$ and in G. Thus a_k is adjacent to every vertex of C, so we can take $v = a_k$.

Fact 6.3 leads to a contradiction: $C \cup \{v\}$ is a clique, and it should have been chosen in preference to C. Thus C is the desired $n/2$ separator. □

□

Corollary 6.1 *Let G be a chordal graph with n vertices and m edges. Suppose G's vertices have nonnegative weights that add up to n. Then G has a set of $O(\sqrt{m})$ vertices whose removal leaves no connected component of weight more than $n/2$.*

Proof: Theorem 6.1 says that G has a clique that separates the graph as required. This clique has at most m edges and hence only $O(\sqrt{m})$ vertices. □

Corollary 6.2 *Let G be a chordal graph with n vertices and m edges. Then G has a set of $O(\sqrt{m})$ vertices whose removal leaves no connected component with more than $n/2$ vertices.*

6.2 An $O(m)$ algorithm

Throughout this section G will be a connected chordal graph with n vertices having nonnegative weights that add up to n, and with m edges. We shall present an algorithm to find the separator of Theorem 6.1 in $O(m)$ time.

The proof of Theorem 6.1 leads directly to the following algorithm.

```
procedure SLOW CHORDAL SEPARATOR (graph G);
    begin
        C ← {};
        while some component A of G \ C has weight more than n/2 do
            while some vertex x of C is adjacent to no vertex of A do
                C ← C − {x}
            od;
            v ← some vertex of A adjacent to every vertex of C;
            C ← C ∪ {v}
        od;
        return C
    end
```

Since a vertex is added to C at most once, the main loop is executed at most n times. The whole algorithm is easily implemented to run in $O(mn)$ time. To speed it up, we need to make four observations.

First, a close examination of the proof of Theorem 6.1 reveals that vertices are added to C in the reverse of a perfect elimination order. We can save searching by precomputing this order.

Second, we don't really need to delete vertices from C as we go along. We can just keep adding vertices to C until no component of $G - C$ weighs more than $n/2$; it then turns out that the desired separator is the last vertex added to C, plus all vertices of C adjacent to that vertex. This observation is formalized and proved in Lemma 6.6 below.

Third, we can find "the last vertex added to C" by looking at the vertices in the opposite order, that is, in perfect elimination order. We simply start with an empty graph H and add vertices of G to it in perfect elimination order until some connected component of H weighs more than $n/2$. Then the last (highest-numbered) vertex added to H is the same as the last (lowest-numbered) vertex added to C by the slow algorithm.

Finally, we don't have to represent the connected components of H explicitly. The fast algorithm maintains an array $w[1..n]$ indexed by vertex number. When the i-th vertex is added to H, the array entry $w[i]$ is the weight of the connected component containing that vertex. Because G is chordal, it turns out to be possible to maintain w by doing only one operation per vertex. This observation is formalized and proved in Lemma 6.5 below.

```
procedure FAST CHORDAL SEPARATOR (graph G);
  begin
    real array w[1..n];
    find a perfect elimination ordering {v1, ..., vn} of G;
    for i ← 1 to n do w[i] ← weight of vi od;
    i ← 1;
    while w[i] ≤ n/2 do
      comment w[i] is the weight of the connected component
        of {v1, ..., vi} that contains vi;
      vk ←lowest-numbered neighbor of vi with k > i;
      w[k] ← w[k] + w[i];
      i ← i + 1
    od;
    comment i is minimum such that some component of
      {v1, ..., vi} weighs more than n/2;
    C ← vi plus all of vi+1, ..., vn adjacent to vi;
    return C
  end
```

We will now prove that this algorithm correctly finds the separator C mentioned in Theorem 6.1. This requires two lemmas.

Lemma 6.5 *Consider the subgraph of G induced by $v_1, \ldots, v_i$. At the beginning of the i-th iteration of the* **while** *loop, $w[i]$ is the weight of the connected component of this subgraph that includes v_i. (That is, the comment in the* **while** *loop is correct.)*

Proof: The lemma is immediate for $i = 1$. Suppose it is true for $1, 2, \ldots, i-1$, and consider the situation at the beginning of the i-th iteration of the loop.

Originally $w[i]$ was the weight of v_i, and something was added to $w[i]$ during the k-th iteration if and only if $k < i$, v_k is adjacent to v_i, and v_k is adjacent to no v_h with $k < h < i$. We will show that each connected component of $\{v_1, \ldots, v_{i-1}\}$ containing a vertex adjacent to v_i had its weight added to $w[i]$ exactly once.

Let A be a connected component of $\{v_1, \ldots, v_{i-1}\}$. If A has no vertex adjacent to v_i, nothing was added to $w[i]$ during an iteration for any vertex in A.

Suppose A has a vertex v_k adjacent to v_i, and suppose v_h is the highest-numbered vertex of A. Thus $k \leq h < i$. There is a path from v_h to v_i through vertices numbered no higher than i, so by Lemma 6.3 there is a monotone path from v_h to v_i. Since v_h is the highest-numbered vertex in A, the monotone path must be a single edge. Therefore v_h is adjacent to v_i and, by the inductive hypothesis, the weight of A was added to $w[i]$ during the h-th iteration. If $k < h$ then, again by Lemma 6.3, there is a monotone path from v_k to v_h. This implies that v_k is adjacent to a vertex numbered higher than k but lower than i, so nothing was added to $w[i]$ during the k-th iteration.

This shows that each connected component of $\{v_1, \ldots, v_{i-1}\}$ adjacent to v_i has its weight added to $w[i]$, and nothing else is ever added to $w[i]$. Thus $w[i]$ is the weight of the connected component of $\{v_1, \ldots, v_i\}$ that contains v_i. □

Lemma 6.6 *Consider the smallest i such that some component of $\{v_1, \ldots, v_i\}$ weighs more than $n/2$. Then $C = \{v_i\} \cup \{v_k : k > i$ and v_k is adjacent to $v_i\}$ is a clique whose removal from G leaves no component that weighs more than $n/2$.*

Proof: Define the *boundary* of a set A of vertices, written ∂A, to be the set of vertices of $G - A$ that are adjacent to vertices of A. Let i be as in the statement of the lemma, and let A_i be the heaviest component of $\{v_1, \ldots, v_i\}$.

First, C is a clique, because v_i is simplicial in $\{v_{i+1}, \ldots, v_n\}$ by the definition of a perfect elimination ordering.

Second, v_i must be in A_i, because A_i weighs more than $n/2$ but no component of $\{v_1, \ldots, v_{i-1}\}$ weighs more than $n/2$. Then v_i is the last vertex of A_i in the perfect elimination ordering, so the same argument as the proof of Fact 6.3 in Theorem 6.1 shows that v_i is adjacent to every vertex in ∂A_i. Therefore C is $\partial A_i \cup \{v_i\}$. Since A_i is the only component of $G - \partial A_i$ that weighs more than $n/2$, $\partial A_i \cup \{v_i\}$ is a set whose deletion from G leaves no component that weighs more than $n/2$. □

The correctness of the algorithm follows immediately from these two lemmas: Lemma 6.5 implies that the **while** loop finds the smallest i such that some component of $\{v_1, \ldots, v_i\}$ weighs more than $n/2$, and then Lemma 6.6 says that the algorithm finds the desired separator.

It remains only to analyze the algorithm's running time. Finding a perfect elimination ordering takes $O(m)$ time by an algorithm of Rose, Tarjan, and Lueker [52]. The **while** loop is executed at most n times. Finding the lowest-numbered neighbor of v_i means looking at all its adjacent vertices, for a total of $O(m)$ over all vertices. Finally, the computation of C takes $O(m)$ time. Thus the total running time is $O(m)$.

6.3 Fast parallel chordal graph recognition

Chordal graphs can be recognized in sequential time $O(m)$ using either Lexicographic Breadth First Search (LBFS), a variant on breadth first search due to Rose,Tarjan, and Lueker [52], or Maximum Cardinality Search (MCS), which is defined as follows in an article by Tarjan and Yannakakis [55]:

> Vertices are numbered from 1 to n. The next vertex to be numbered is always one which is adjacent to the most numbered vertices, ties being broken arbitrarily.

The fast parallel algorithm we will exhibit runs in time $O(\log n)$ and uses $O(n^4)$ processors. Hence it is in the class commonly known as NC [7,48]. The model for parallel computation used here is the shared memory computer, in which a number of processors work together synchronously and access a common main memory. Different processors can read from the same memory location simultaneously; they can write to different memory locations simultaneously, but no two processors may attempt to write to the same location at the same time.

There seems to be no obvious way to implement either LBFS or MCS efficiently in parallel. Like depth first search MCS appears to be sequential in nature. Instead, we use a different characterization of chordal graphs to construct a fast parallel recognition algorithm. A *v, w separator* is a set of vertices that cuts every path from v to w.

Lemma 6.7 [8] *A graph G is chordal if and only if for all vertices v and w, every minimal v, w separator in G is a clique.*

At first, this does not seem to help, since for a given pair of vertices v, w there may be exponentially many separators. Hence we can't just check all of them. By changing the characterization from Lemma 6.7 slightly our task becomes easier. We will only have to look at separators of a certain kind.

Definition 6.1 *A* close v, w vertex separator *(or* close separator *for short) is a minimal v, w vertex separator M, such that $M \subseteq \Gamma(v)$.*

Note that with this definition a close separator is unique, hence we can talk about *the* close v, w vertex separator.

Lemma 6.8 *A graph G is chordal if and only if for each pair of vertices v, w in G, the close v, w vertex separator is a clique.*

```
PARALLEL CHORDAL GRAPH RECOGNITION:
for every vertex pair v, w do (in parallel):
    A ← Γ(v)
    if w ∈ A then return(YES).
    M ← ∅
    for each u ∈ A do (in parallel):
        G_u ← (G \ A \ {v}) ∪ {u}
        if there is a path in G_u from u to w then M ← M ∪ {u}
    od
    if M is a clique then return(YES) else return(NO).
od
if for each v, w pair the answer is "YES" then G is chordal.
```

Figure 6.1: Parallel algorithm for recognizing chordal graphs.

Proof: Assume G is chordal. By Lemma 6.7, *any* minimal vertex separator is a clique. Hence, in particular, any close v, w vertex separator is a clique.

For the proof in the other direction, assume that every close v, w vertex separator is a clique, and let $C = (u_0, u_1, \ldots, u_k = u_0)$ be a cycle in G of length ≥ 4. Let $v = u_i$ and $w = u_j$ be two nonadjacent vertices on C. If no two such vertices exist, then C must have a chord, and we are done. Otherwise, let M be a close v, w vertex separator. Since there are vertex-disjoint paths going from u_{i-1} and u_{i+1} to w, both u_{i-1} and u_{i+1} must be in M. By assumption, M is a clique, so (u_{i-1}, u_{i+1}) is an edge in G, and hence a chord in C.

Since any cycle of length at least 4 has a chord, G is chordal. □

It is now possible to have an algorithm look at all close separators. The algorithm is shown in Figure 6.1.

Lemma 6.9 *The set M constructed in the algorithm is a close v, w vertex separator.*

Proof: Obviously, $M \subseteq \Gamma(v)$. If M is not a close v, w separator, then there is a path $v = u_0, u_1, \ldots, u_k = w$ in $G \setminus M$. Let u_i be the last vertex on this path that is in $\Gamma(v)$. Such a vertex exists since $u_1 \in \Gamma(v)$. Hence, there is a path from u_i to w that does not go through any of the vertices of $\Gamma(v)$. But this implies that $u_i \in M$, a contradiction.

Clearly, M is minimal, since for each $u \in M$ there is a path from u to w in $G \setminus M \cup \{u\}$; removing u from M would give us a path from v to w in $G \setminus M$.

We conclude that M is a close v, w vertex separator. □

Theorem 6.2 *The algorithm in Figure 6.1 correctly tests a graph for chordality in time $O(\log n)$ time using $O(n^4 + n^3 m)$ processors.*

Proof: The correctness of the algorithm follows immediately from Lemma 6.8 and Lemma 6.9.

As for the running time, the only step that is not obviously implementable in logarithmic time is Step 7. But this step can be carried out using an algorithm that computes connected components. The fastest parallel algorithm for this task runs in time $O(\log n)$ [53] and uses $O(n+m)$ processors. Total running time is then $O(\log n)$.

To count the number of processors used, we first note that there are $\binom{n}{2}$ pairs of vertices in G. For each adjacent vertex of each such pair we do a path testing which requires $O(n+m)$ processors. Hence, we need at most a total of $O(n^2 \cdot n \cdot (n+m)) = O(n^4 + n^3 m)$ processors. □

It is interesting to note that the algorithm above recognizes chordal graphs without computing a perfect elimination order, in contrast to LBFS and MCS. This, of course, makes it less useful than either of the sequential algorithms. At present it is not known whether the problem of computing a perfect elimination order is in NC.

Chapter 7

Conclusion

Below is a brief summary of the problems disseminated in this thesis. In connection with each subject are discussed open problems and possible directions for future research.

Our emphasis was on the nonzero structure of matrices. We showed how a structure could be given different representations: Boolean matrix, undirected graph, directed graph, or bipartite graph. With the help of the representation of the matrix structures, we attempted to model the structural effects of numerical algorithms. The problem of accurately modeling an algorithm is complex, and for many numerical algorithms we do not have efficient structural versions. In the case of Gaussian elimination we could show that the structural version, although not exact, is correct on subclasses of matrix structures that appear often in practice. We also gave precise conditions on the matrix structure for structural Gaussian elimination to be correct.

With QR factorization, we isolated a class of structures (namely, those with the strong Hall property) for which a structural version of the normal equations algorithm is correct. Similarly, we showed that structural Givens and Householder factorization are correct on structures with the strong Hall property. However, our definition of structural Givens and Householder factorization is somewhat restrictive. We also showed that the structures computed by these different structural algorithms can be linearly ordered by substructure.

An obvious open problem here is to find an efficient exact structural QR factorization algorithm. Failing that, it would also be interesting to find structural algorithms that model Givens and Householder factorization without the restrictions that the matrix be given what we called a "legal ordering". Another unanswered question is whether structural Givens and structural Householder (as defined in Chapter 3) always compute the same structure.

The next problem we looked at was that of finding the best ordering in which to perform factoring algorithms. By "best" we mean the ordering that causes the least fill in the factored matrix. These combinatorial problems were studied on the different graph models. For the undirected and directed graph models the problem of minimizing fill was known to be NP-complete. On the bipartite graph model, the problem remains open. We showed how it is really two different problems on the bipartite graph (MFVO and MPEGC). An interesting question is whether these two problems can be shown equivalent on some restricted class of graphs. It is known that they differ on a bipartite graph representing a nonsymmetric matrix with a full nonzero diagonal. However, it is not known whether the problems are equivalent on a bipartite graph representing a symmetric matrix with a full nonzero diagonal.

Since the general problems of minimizing fill are NP-complete it is important to study approximation algorithms. We looked at two common heuristics, MINDEF and MINDEG, and saw that their performance is not always within a multiplicative constant of the best (in our terminology, they did not have a constant bound on κ). However, we do not know of any examples that show MINDEF/MINDEG to have a non-constant lower bound on λ. Either an example to show such a non-constant lower bound for MINDEF/MINDEG, or a proof that they have a constant bound on λ would be interesting.

It would of course be of great interest to find an efficient approximation algorithm with a guaranteed bound on its performance: preferably, a multiplicative constant. Since the average numerical problem is not a random matrix, but rather has a lot of "regularity" to its structure, it is also important to ask whether MINDEF/MINDEG, or any other efficient heuristics, have theoretical bounds on their performance on some special class of graphs. We showed that MINDEF gives optimal orderings on chordal and outerplanar graphs. More interesting, though, would be to show theoretical bounds for either heuristic on grid graphs, or other finite element graphs.

Another heuristic that is in use is quotient tree partitioning. We showed that the problem of finding the best partitioning, using three different measures for what is best, is NP-complete. We also gave a new algorithm for finding a maximal quotient tree, and proved it to have linear running time.

As for minimizing fill in QR factorization, we showed that it essentially reduces to the problem of minimizing fill in Gaussian elimination. Hence, the same limits on the complexity apply, and the same approximation algorithms can be used. We showed that the problem of minimizing fill when computing the structure using structural normal equations (or any other structural algorithm) is NP-complete.

We had showed that structural normal equations, Givens, and Householder all were correct on matrix structures with the strong Hall property. We generalized this to matrices without this property, by showing how the rows and columns of an arbitrary matrix can be reordered into the Dulmage-Mendelsohn order. With this reordering, structural Givens is again correct. However, we also described how the least squares problem can be solved using block back substitution on the reordered matrix.

Finally, we studied chordal graphs, and showed how it is possible to find a

$O(\sqrt{m})$ separator in a chordal graph in linear time. In addition, we showed that chordal graphs can be recognized in parallel logarithmic time on a polynomial number of processors. Unlike its sequential versions, the parallel algorithm did not construct a perfect elimination order. It is still an open question whether it is possible to find a perfect elimination order for a chordal graph in parallel logarithmic time. Also, a desired improvement on the algorithm presented here is to bring down the number of processors used.

Bibliography

[1] Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, Reading, Mass., 1974.

[2] C. Berge.
"Some classes of perfect graphs."
in *Graph theory and theoretical physics*, Frank Harary, Ed.
Academic Press, New York, 1967.

[3] Robert K. Brayton, Fred G. Gustavson and Ralph A. Willoughby.
"Some results on sparse matrices."
Mathematics of Computation **24**: 937–954, 1970.

[4] James R. Bunch and Donald J. Rose.
"Partitioning, tearing and modification of sparse linear systems."
Journal of Mathematical Analysis and Applications **48**: 574–593, 1974.

[5] James R. Bunch and Donald J. Rose (Eds.).
Sparse Matrix Computations.
Academic Press, New York, 1976.

[6] Thomas F. Coleman, Anders Edenbrandt and John R. Gilbert.
Predicting fill for sparse orthogonal factorization.
To appear in *Journal of the ACM.*

[7] Stephen A. Cook.
"The classification of problems which have fast parallel algorithms."
Proceedings of the 1983 International Foundations of Computation Theory Conference, Lecture Notes in Computer Science, volume 158, Springer-Verlag, 1983.

[8] G. A. DIRAC.
"On rigid circuit graphs."
Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg **25**: 71–76, 1961.

[9] IAIN S. DUFF.
"On algorithms for obtaining a maximum transversal."
ACM Transactions on Mathematical Software **7**: 315–330, 1981.

[10] A.L. DULMAGE AND N.S. MENDELSOHN.
"Coverings of bipartite graphs."
Canadian Journal of Mathematics **10**: 517–534, 1958.

[11] A.L. DULMAGE AND N.S. MENDELSOHN.
"A structure theory of bipartite graphs of finite exterior dimension."
Transactions of the Royal Society of Canada **53**: Sect III, 1–13, 1959.

[12] A.L. DULMAGE AND N.S. MENDELSOHN.
"Two algorithms for bipartite graphs. "
Journal of the Society for Industrial and Applied Mathematics **11**: 183–194, 1963.

[13] HERBERT FEDERER.
Geometric Measure Theory.
Springer-Verlag, Berlin, 1969.

[14] L.R. FORD AND D.R. FULKERSON.
Flows in Networks.
Princeton University Press, Princeton, N.J., 1962.

[15] D. R. FULKERSON AND O. A. GROSS.
"Incidence matrices and interval graphs."
Pacific Journal of Mathematics **15**: 835–855, 1965.

[16] MICHAEL R. GAREY AND DAVID S. JOHNSON.
Computers and Intractability.
W.H. Freeman and Company, San Francisco, 1979.

[17] W. MORVEN GENTLEMAN.
"Row elimination for solving sparse linear systems and least squares problems."
Proceedings Dundee Conference on Numerical Analysis, Lecture Notes in Mathematics, volume 506, G.A. Watson, (Ed.), 1975.

[18] ALAN GEORGE.
"Nested dissection of a regular finite element mesh."
SIAM Journal on Numerical Analysis **10**: 345–363, 1973.

[19] ALAN GEORGE AND MICHAEL T. HEATH.
"Solution of sparse linear least squares problems using Givens rotations."
Linear Algebra and its Applications **34**: 69–83, 1980.

[20] ALAN GEORGE AND JOSEPH W.H. LIU.
"Algorithms for matrix partitioning and the numerical solution of finite element systems."
SIAM Journal on Numerical Analysis **15**: 297–327, 1978.

[21] ALAN GEORGE AND JOSEPH W. LIU.
Computer Solution of Large Sparse Positive Definite Systems.
Prentice-Hall, Englewood Cliffs, N.J., 1981.

[22] ALAN GEORGE AND ESMOND NG.
"An implementation of Gaussian elimination with partial pivoting for sparse systems."
SIAM Journal on Scientific and Statistical Computing **6**: 390–409, 1985.

[23] JOHN RUSSELL GILBERT.
Graph Separator Theorems and Sparse Gaussian Elimination.
Ph.D. Thesis, Stanford University, 1980.

[24] JOHN R. GILBERT.
"A note on the NP-completeness of vertex elimination on directed graphs."
SIAM Journal on Algorithmic and Discrete Methods **1**: 292–294, 1980.

[25] JOHN R. GILBERT.
Personal communication.

[26] JOHN R. GILBERT, JOAN P. HUTCHINSON, AND ROBERT ENDRE TARJAN.
"A separator theorem for graphs of bounded genus."
Journal of Algorithms **5**: 391–407, 1984.

[27] GENE H. GOLUB AND PETER BUSINGER.
"Linear least squares solutions by Householder transformations."
Numerische Mathematik **7**, Hand Book Series Linear Algebra, 269–276, 1965.

[28] MARTIN CHARLES GOLUMBIC.
Algorithmic graph theory and perfect graphs.
Academic Press, New York, 1980.

[29] FRED GUSTAVSON.
"Finding the block lower triangular form of a sparse matrix."
In [5], 275–289.

[30] ANDRÁS HAJNAL AND JÁNOS SURÁNYI.
"Über die Auflösung von Graphen in vollständige Teilgraphen."
Annales Universitatis Scientarium Budapestinensis, Sectio Mathematica **1**: 113–121, 1958.

[31] P. HALL.
"On representatives of subsets."
Journal of the London Mathematical Society **10**: 26–30, 1935.

[32] MICHAEL T. HEATH.
"Some extensions of an algorithm for sparse linear least squares problems."
SIAM Journal on Scientific and Statistical Computing **3**: 223–237, 1982.

[33] MICHAEL T. HEATH.
"Numerical methods for large sparse linear least squares problems."
SIAM Journal on Scientific and Statistical Computing **5**: 497–513, 1984.

[34] JOHN E. HOPCROFT AND RICHARD M. KARP.
"An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs."
SIAM Journal on Computing **2**: 225–231, 1973.

[35] THOMAS D. HOWELL.
"Partitioning using PAQ."
In [5], 23–37.

[36] CAMILLE JORDAN.
"Sur les assemblages de lignes."
Journal für die reine und angewandte Mathematik **70**: 185–190, 1869.

[37] DAN HOEY AND CHARLES E. LEISERSON.
A layout for the shuffle-exchange network.
Carnegie-Mellon University Department of Computer Science, Technical report CMU-CS-80-139, 1980.

[38] DIANE M. JOHNSON, A.L. DULMAGE AND N.S. MENDELSOHN.
"Connectivity and reducibility of graphs."
Canadian Journal of Mathematics **14**: 529–539, 1962.

[39] RICHARD M. KARP.
"Reducibility among combinatorial problems."
in R.E. Miller and J.W. Thatcher (Eds.), *Complexity of Computer Computations*, Plenum Press, New York, 1972, 85–103.

[40] FRANK THOMSON LEIGHTON.
"New lower bound techniques for VLSI."
Proceedings of the 22nd Annual Symposium on Foundations of Computer Science, 1–12, 1981.

[41] CHARLES E. LEISERSON.
"Area-efficient graph layouts (for VLSI)."
Proceedings of the 21st Annual Symposium on Foundations of Computer Science, 270–281, 1980.

[42] C. G. LEKKERKERKER AND J. CH. BOLAND.
"Representations of a finite graph by a set of intervals on the real line."
Fundamenta Mathematicae Polska Akademia Nauk **51**: 45–64, 1962.

[43] P. M. LEWIS II, R. E. STEARNS AND JURIS HARTMANIS.
"Memory bounds for the recognition of context-free and context-sensitive languages."
IEEE Conference Record on Switching Theory and Logical Design, 191–202, 1965.

[44] RICHARD J. LIPTON, DONALD J. ROSE AND ROBERT ENDRE TARJAN.
"Generalized nested dissection."
SIAM Journal on Numerical Analysis **16**: 346–358, 1979.

[45] RICHARD J. LIPTON AND ROBERT ENDRE TARJAN.
"A separator theorem for planar graphs."
SIAM Journal on Applied Mathematics **36**: 177–189, 1979.

[46] CHRISTOS H. PAPADIMITRIOU AND KENNETH STEIGLITZ.
Combinatorial Optimization.
Prentice Hall, Englewood Cliffs, N.J., 1982.

[47] S. PARTER.
"The use of linear graphs in Gauss elimination."
SIAM Review **3**: 119–130, 1961.

[48] NICHOLAS PIPPENGER.
"On simultaneous resource bounds."
Proceedings 20th IEEE FOCS, 307–311, 1979.

[49] DONALD J. ROSE.
"Triangulated graphs and the elimination process."
Journal of Mathematical Analysis and Applications **32**: 597–609, 1970.

[50] DONALD J. ROSE.
"A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations."
In *Graph Theory and Computing* (Ronald C. Read, editor), 183–217, Academic Press, 1972.

[51] DONALD J. ROSE AND ROBERT ENDRE TARJAN.
"Algorithmic aspects of vertex elimination on directed graphs."
SIAM Journal on Applied Mathematics **34**: 176–197, 1978.

[52] DONALD J. ROSE, R. ENDRE TARJAN AND GEORGE S. LUEKER.
"Algorithmic aspects of vertex elimination on graphs."
SIAM Journal on Computing **5**: 266–283, 1976.

[53] YOSSI SHILOACH AND UZI VISHKIN.
"An $O(\log n)$ parallel connectivity algorithm."
Journal of Algorithms **3**: 57–67, 1982.

[54] ROBERT TARJAN.
"Depth-first search and linear graph algorithms."
SIAM Journal on Computing **1**: 146–160, 1972.

[55] ROBERT ENDRE TARJAN AND MIHALIS YANNAKAKIS.
"Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs."
SIAM Journal on Computing **13**: 566–579, 1984.

[56] W. F. TINNEY.
"Comments on using sparsity techniques for power system problems."
in *Sparse Matrix Proceedings*, IBM Research report RAI 3-12-69, 1969.

[57] MIHALIS YANNAKAKIS.
"Computing the minimum fill-in is NP-complete."
SIAM Journal on Algebraic and Discrete Methods **2**: 77–79, 1981.

[58] EARL ZMIJEWSKI AND JOHN R. GILBERT.
Wide quotient trees for finite element problems.
Cornell University, Department of Computer Science Technical Report 85-673, 1985.