A Multi-Elimination Technique for Equilibrium Systems

Christoph L. Fabianek, Wilfried N. Gansterer, Christoph W. Ueberhuber

AURORA TR2002-04

Institute for Applied and Numerical Mathematics
Technical University of Vienna
Wiedner Hauptstrasse 8-10, A-1040 Wien, Austria

E-Mail: fabianek@aurora.anum.tuwien.ac.at,
ganst@aurora.anum.tuwien.ac.at,
christof@aurora.anum.tuwien.ac.at

This work described in this report was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund FWF.

Contents

Chapter 1

Introduction

This report discusses a solution method of the quadratic programming problem described in Gansterer et al. [5]. For a given symmetric matrix $A \in \mathbb{R}^{n \times n}$ with the structure

$$A = \begin{pmatrix} D & X^T \\ X & R \end{pmatrix} \qquad \begin{matrix} D \in \mathbb{R}^{l \times l} \\ X \in \mathbb{R}^{m \times l} \\ R \in \mathbb{R}^{m \times m} \end{matrix} \;,\; l + m = n \tag{1.1}$$

(called *augmented system*), the solution of the linear system

$$A\,x = b$$

is sought. Properly partitioning x and b

$$\begin{pmatrix} D & X^T \\ X & R \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{1.2}$$

one gets the following two equations:

$$D\,x_1 + X^T\,x_2 = b_1 \tag{1.3}$$
$$X\,x_1 + R\,x_2 = b_2. \tag{1.4}$$

Equation (1.3) leads to

$$x_1 = D^{-1}\,b_1 - D^{-1}\,X^T\,x_2$$

and with substitution into (1.4) it results in

$$X\,D^{-1}\,b_1 - X\,D^{-1}\,X^T\,x_2 + R\,x_2 = b_2$$

and thus in

$$\left(R - X\,D^{-1}\,X^T\right)\,x_2 = b_2 - X\,D^{-1}\,b_1. \tag{1.5}$$

The non-singularity of A and D guarantees the non-singularity of the Schur complement $R - X\,D^{-1}\,X^T$.

Therefore the original problem of solving a system of linear equations with dimension n can be reduced to a problem of dimension $m = n - l$ (plus some additional matrix multiplications). If D is diagonal, then D^{-1} can be formed easily in $O(n)$ flops. Therefore, the central idea of the algorithm discussed here is to restore diagonality in the upper left corner of the reduced matrix.

Reordering the matrix $R - X\,D^{-1}\,X^T$ in (1.5) (called the *normal system*) to the augmented system of matrix A (using the permutation matrix P) leads to a *recursive* algorithm for computing x. This scheme is sometimes called *multi-elimination* (Saad [13]) and is formulated in Algorithm 1.

Algorithm 1 Basic Multi-elimination Algorithm.

Input: matrix $A \in \mathbb{R}^{n\times n}$, vector $b \in \mathbb{R}^n$
Output: vector $x \in R^n$
Variables: notation as in (1.1) and (1.2)

1. [Reducing A to normal system and correcting b]
   $A = R - X\,D^{-1}\,X^T$
   $b_2 = b_2 - X\,D^{-1}\,b_1$
2. [Finding a permutation matrix P such that A is an augmented system]
   $P^T A\,P\;P^T x = P^T b_2$
3. [Recursive call of the algorithm with the reduced matrix A and b_2 until the maximum recursion depth is reached, then calculate x_2]
   $x_2 = A^{-1}\,b_2$
4. [Calculating x_1 while unrolling the recursion]
   $x_1 = D^{-1}\,b_1 - D^{-1}\,X^T\,x_2$

The purpose of this report is to present methods for the practical use of this algorithm. I.e., permutation matrices P are sought which decrease the size of matrix A satisfying the following criteria.

- Produce a low fill-in for the matrices A (reduce the computational cost of solving the last system).
- Minimize the computational cost for generating P (i.e., the savings from solving a smaller system should not be neutralized).

Chapter 2

Reordering Strategies

Algorithm 1 shows that the crucial point is to find a good reordering (i. e., a favorable permutation matrix P), which leads to a large diagonal matrix D. Furthermore it is worthwhile to look for a matrix R which is as sparse as possible to avoid fill-in in the new matrix $A = (R - X\,D^{-1}\,X^T)$.

2.1 Standard Reordering

The first reordering algorithm (called *standard reordering*) uses the following idea. Start with a 1×1 diagonal matrix and scan through the matrix A looking for columns with the first nonzero element below the current diagonal matrix. Exchange these columns (and their corresponding rows to maintain symmetry) with the first row/column right after the diagonal matrix.

Fig. 2.1 illustrates this reordering: row/column 11 (the first nonzero element is at 9, i. e., below the current diagonal matrix of size 3) is exchanged with row/column 4, which leads to a new diagonal matrix of size 4.

To apply this reordering strategy efficiently to given matrices it has to be modified in the following way: Firstly the permutation matrix P has to be stored during this process, since this information will be needed when unrolling the recursion to obtain the resulting vector x. It is convenient to store the permutation matrix P as a vector p, where each entry represents the row in which the one is stored (e. g., the initial identity matrix is represented by the vector $[1, 2, \ldots, n]$).

In addition it turned out that it does not make sense to immediately apply the row/column exchange as soon as a suitable column is found. Preferably, the permutation vector p is applied at the end of the reordering algorithm because this leads to a cleaner implementation and the process of generating the matrix E can be easily parallelized when it is done in a separate step at the beginning. (Matrix E contains information about matrix A needed for the reordering algorithm; see below.)

Thus information about the exchanged rows/columns has to be stored. If one row/column is used to increase the size of the diagonal matrix, all nonzero elements of this row/column "disable" the corresponding rows/columns. Therefore

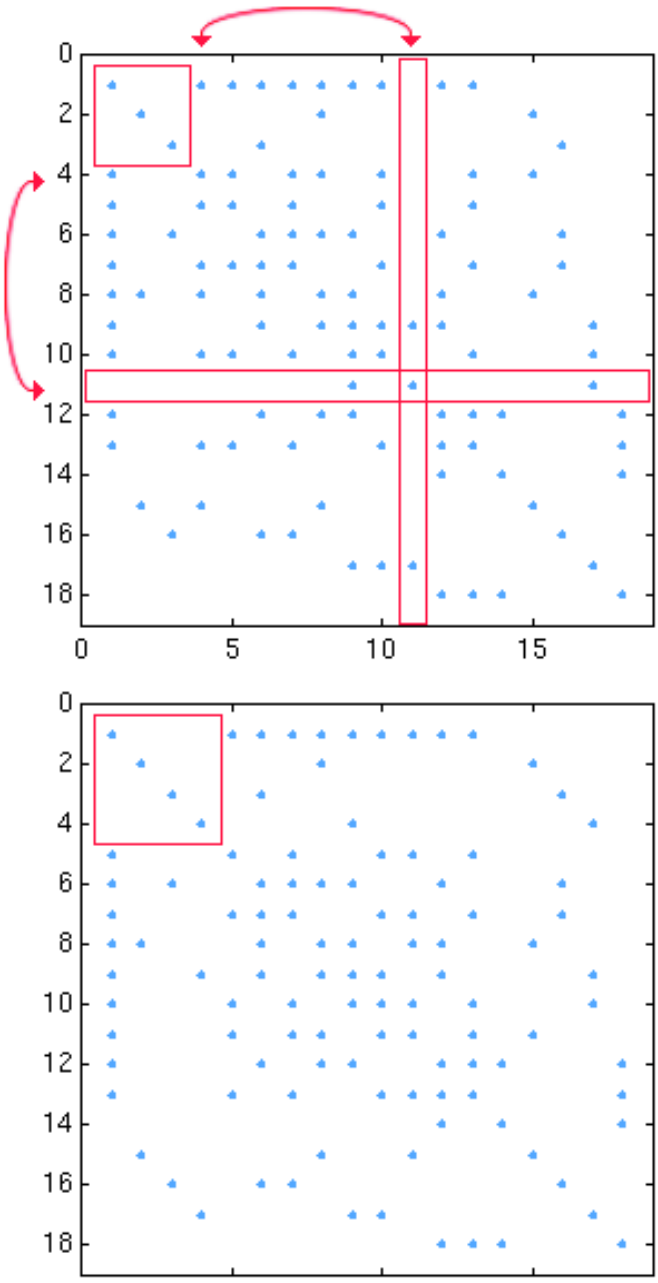

Figure 2.1: Standard reordering.

a vector a of "active" rows/columns is used and each time an active row/column is found (the state is saved in a) the positions in a are disabled where the row/column has nonzero elements; see Algorithm 2.[1]

[1] In the algorithms MATLAB syntax is used for describing sparse matrix manipulations.

Algorithm 2 Standard Reordering.

Input: matrix $A \in \mathbb{R}^{n \times n}$
Output: permutation vector $p \in \mathbb{R}^n$
Variables: d – current size of the diagonal matrix,
i – counter for the stepwise proceeding in the matrix,
vector $a \in \mathbb{R}^n$ – active rows / columns,
matrix $E \in \mathbb{R}^{n \times n}$ – contains the positions of the nonzeros in A

1. [Initialization] set $d = 1$; $i = 1$; $p = 1, 2, \ldots, n$ (the original order of the rows/columns); $a =$ `true`; $E(:, x) = find(A(:, x))$
2. [Check column i] if $a(i) =$ `true` and $E(1, i) > d$ then proceed with 3, otherwise increase i and repeat this step until $i = n$
3. [Update] exchange $p(d + 1)$ with $p(i)$; set $a(E(:, i)) =$ `false` (i. e., these rows/columns can not be used anymore); increase i and d; proceed with Step 2

An example using the upper matrix of Fig. 2.1 illustrates the application of Algorithm 2. The diagonal matrix is of size 3 and the first suitable row/column is found at row/column 11 to increase the diagonal matrix. So $u(4)$ with $u(11)$ is exchanged resulting in

$$u = [1\ 2\ 3\ 11\ 5\ \ldots\ 10\ 4\ 12\ \ldots],$$

and $a[9]$, $a[11]$ and $a[17]$ are set to `false`. The next suitable row/column is 14, where these steps are repeated. The next suitable row/column 17 can not be used, since the exchange of row/column 4 with 11 will produce some sort of fill-in. The same applies to the last row/column, because of row/column 14.

2.2 Improved Reordering

Another approach is to order the matrices by the first nonzero element but only column by column, which could be visualized as stairs; see Fig. 2.2. The reordering strategies described here can now achieve a maximum size of the diagonal matrix equal to the number of stairs. (The example shows a 5×5 diagonal matrix.)

In the standard reordering always the first column was chosen for permuting into the leading diagonal matrix. Now additional information of the matrix is used to choose another column. This wouldn't lead to a larger diagonal matrix in the first step, but since different submatrices X and R (notation as in (1.1)) are built, this leads to sparser matrices in the next iteration step.

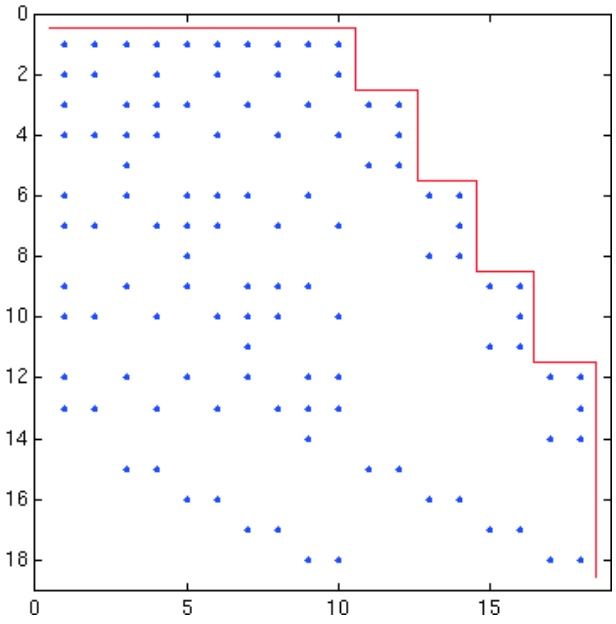

Figure 2.2: Stairs produced by column-wise ordering.

Since the only information that can easily be derived from each column is the number of nonzeros, the following strategies are investigated.

- Most sparse column: Choose the column in each group with the lowest number of nonzero elements—this will lead to a sparse matrix X.
- Most dense column: Use the column with the most nonzeros—leads to a denser matrix X.
- Monte-Carlo: Choose randomly one column in each group—this approach is mainly used to determine the influence of choosing a special column.

It would also be possible to take the arrangement of the nonzeros into account, but this would increase computational cost and would therefore be inefficient.

Algorithm 3 shows the detailed process of this reordering. The main difference to standard reordering is the additional use of matrix F, which is used to store the matrix structure.

Algorithm 3 Improved Reordering.

Input: matrix $A \in \mathbb{R}^{n \times n}$
Output: permutation vector $p \in \mathbb{R}^n$
Variables: d – current size of the diagonal matrix,
i – index of the chosen row/column at each stair,
matrix $E \in \mathbb{R}^{n \times n}$ – contains the positions of the nonzeros in A,
matrix $F \in \mathbb{N}^{3 \times n}$ – data for each row/column

1. [Initialization] $d = 1$, $p = 1, 2, \ldots, n$, $E(:, x) = find(A(:, x))$,
$F(1, :) = 1, 2, \ldots, n$ (the original order of the rows/columns),
$F(2, :) = E(1, :)$ (the index of the first nonzero elements),
$F(3, i) = nnz(A(:, i))$ (the number of nonzeros of each row/column);
$sort(F(2, :))$ – order corresponding to the first nonzero (i. e., produce "stairs")

2. [Find optimal column for each stair] for each stair – as given in $F(2, :)$ – the most sparse/dense or a random row/column is chosen – stored in $F(3, :)$

3. [Swap positions] exchange the elements in $p(d)$ and $F(:, i)$, increase d, again it has to be taken care of disabling rows/columns as in Algorithm 2, this information can also be stored in $F(3, :)$, and such rows/columns (i. e., $E(:, i)$) must be excluded in Step 2

Proceed with Step 2 until all "stairs" are processed

Chapter 3

Preordering

As seen at the improved reordering a high number of "stairs" leads to a corresponding large diagonal matrix. So it is obvious to search for algorithms which generate a reordering with this structure and then perform the improved reordering on this matrix. Such reorderings are classified as reduction algorithms and Goldenberg et al. [8] discusses a comparison of five such algorithms for sparse symmetric matrices.

3.1 Prerequisites from Graph Theory

At first a few basic concepts from graph theory are introduced and their correspondence to matrix concepts is established. Although few results from graph theory have found direct application to the analysis of sparse matrix computations, the notation and concepts are convenient and helpful in describing algorithms and identifying or characterizing matrix structure.

For our purpose, a graph $G = (X, E)$ consists of a finite set of *nodes* or *vertices* together with a set E of *edges*, which are unordered pairs of vertices. An *ordering* {*labeling*} α of $G = (X, E)$ is simply a mapping of $\{1, 2, \ldots, n\}$ onto X, where n denotes the number of nodes of G.

Let A be an n by n symmetric matrix, with entries a_{ij}. The *ordered graph* of A, denoted by $G^A = (X^A, E^A)$ is one for which the n vertices of G^A are numbered from 1 to n, and $\{x_i, x_j\} \in E^A$ if and only if $a_{ij} = a_{ji} \neq 0, i \neq j$. Here x_i denotes the node of X^A with label i. Fig. 3.1 illustrates the structure of a matrix and its labelled graph (the ith diagonal element is highlighted to emphasize its correspondence with node i of the corresponding graph).

For any n by n permutation matrix P, the unlabelled graphs of A and PAP^T are the same but the associated labelings are different. Thus the unlabelled graph of A represents the structure of A without suggesting any particular ordering.

The ith row of A is further defined as

$$f_i(A) = \min\{j \mid a_{ij} \neq 0\}$$

and

$$\beta_i(A) = i - f_i(A).$$

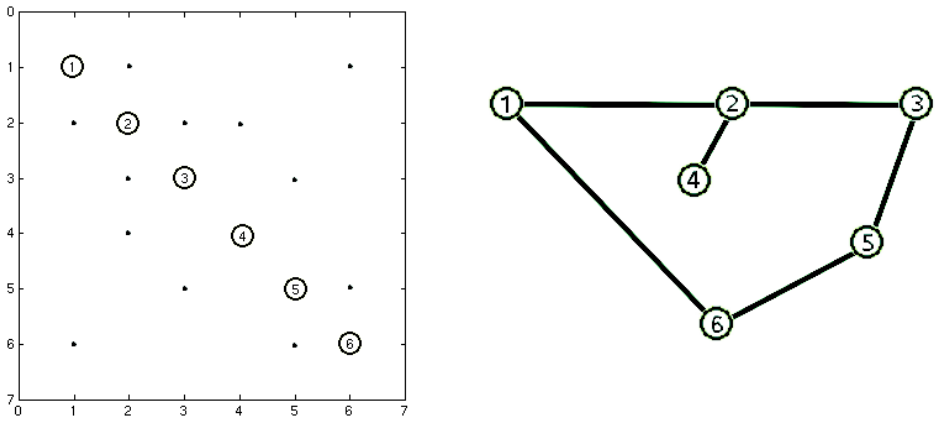

Figure 3.1: A matrix and its labelled graph.

The number $f_i(A)$ is simply the column subscript of the first nonzero component in row i of A, whereas $\beta_i(A)$ is the ith bandwidth of A. The bandwidth of A is given by

$$\beta(A) = \max\{\beta_i(A) \mid 1 \leq i \leq n\} = \max\{|i-j| \mid a_{ij} \neq 0\}.$$

Two nodes x and y in G are *adjacent* if $\{x, y\} \in E$. For $Y \subseteq X$, the *adjacent set* of Y, denoted by $Adj(Y)$, is

$$Adj(Y) = \big\{x \in X - Y \mid \{x, y\} \in E \text{ for some } y \in Y\big\}.$$

In words, $Adj(Y)$ is simply the set of nodes in G which are not in Y but are adjacent to at least one node in Y. For $Y \subseteq X$ the *degree* of Y, denoted by $Deg(Y)$, is the number $|Adj(Y)|$.

3.2 The Reverse Cuthill-McKee Ordering

Perhaps the most widely used profile reduction ordering algorithm is a variant of the Cuthill-McKee ordering (Liu and Sherman [9]) which is often chosen because of its simple structure and therefore easy implementation. In 1969, Cuthill and McKee [2] published their algorithm which was primarily designed to reduce the bandwidth of a sparse symmetric matrix (see Fig. 3.2 for an example of the reverse Cuthill-McKee ordering).

The scheme makes use of the following observation. Let y be a labelled node, and z an unlabelled neighbor of y. To minimize the bandwidth of the row associated with z, it is apparent that node z should be ordered as soon as possible after y.

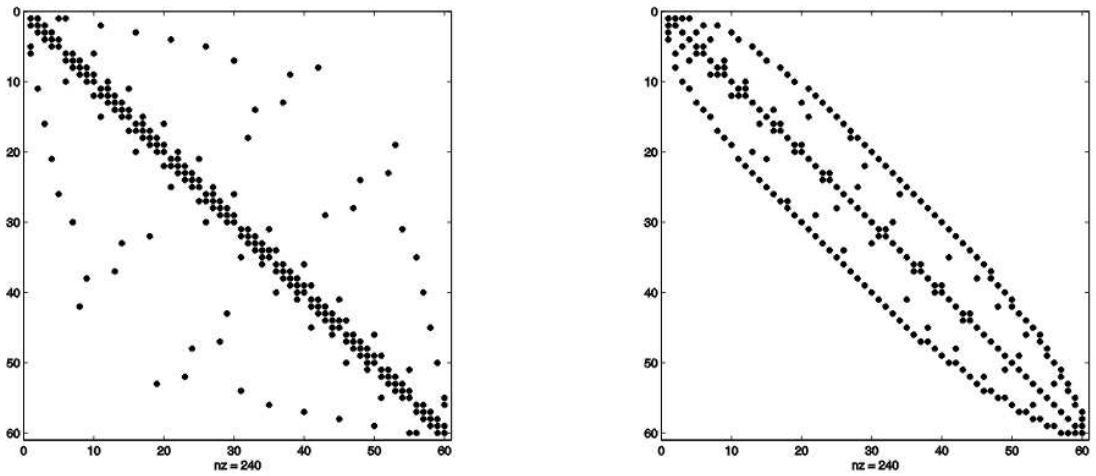

Figure 3.2: Example of the reverse Cuthill-McKee ordering.

The Cuthill-McKee scheme may be regarded to as a method that reduces the bandwidth of a matrix via a local minimization of the β_i's. This suggests that the scheme can be used as a method to reduce the profile of $\sum \beta_i$ of a matrix. George then discovered that the ordering obtained by reversing the Cuthill-McKee ordering often turns out to be much superior to the original ordering in terms of profile reduction, although the bandwidth remains unchanged. He called this the *reverse Cuthill-McKee* ordering (RCM) and is described in Algorithm 4. In case when the graph G^A is disconnected, the algorithm is applied to each connected component of the graph.

Algorithm 4 RCM Algorithm for a Connected Graph.

Input: matrix A
Output: vector y

1. [Determine starting node] determine a starting node r and assign $x_1 = r$
2. [Main loop] for $i = 1, 2, \ldots, n$, find all the unnumbered neighbors of the node x_i and number them in increasing order of degree
3. [Reverse ordering] the reverse Cuthill-McKee ordering is given by $y_1, y_2, \ldots, y_n$ where $y_i = x_{n-i+1}$ for $i = 1, 2, \ldots, n$

Now the problem of finding a starting node for the RCM algorithm is tackled. A path of length k from node x_0 to x_k is an ordered set of distinct vertices $(x_0, x_1, \ldots, x_k)$, where $x_i \in Adj(x_{i+1})$ for $i = 0, 1, \ldots, k-1$. The *distance* $d(x, y)$ between two nodes x and y in the connected graph $G = (X, E)$ is the length of

a shortest path joining nodes x and y. The *eccentricity* of a node x is defined as the quantity

$$l(x) = \max\{d(x,y) \mid y \in X\}.$$

The *diameter* of G is then given by

$$\delta(G) = \max\{l(x) \mid x \in X\}.$$

A node $x \in X$ is said to be a *peripheral node* if its eccentricity is equal to the diameter of the graph, that is, if $l(x) = \delta(G)$. Fig. 3.3 shows a graph having 8 nodes, with a diameter of 5. The nodes x_2, x_5 and x_7 are peripheral nodes.

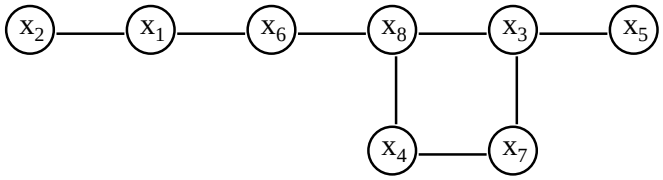

Figure 3.3: An 8-node graph G with $\delta(G) = 5$.

Using this terminology the objective of this section is to describe an efficient heuristic algorithm for finding nodes of high eccentricity. Although the algorithm does not necessarily find a peripheral node, or even one that is close to being peripheral, the nodes found usually do have high eccentricity, and are good starting nodes. Secondly, in most situations it would be too expensive to find peripheral nodes. Therefore the nodes produced by this algorithm are refered to as *pseudo-peripheral nodes.*

Finally, some notation and terminology which is useful in describing the algorithm is introduced. A key construct in the algorithm is the *rooted level structure* as first described in Arany et al. [1]. Given a node $x \in X$, the level structure rooted at x is the *partitioning* $\mathcal{L}(x)$ of X satisfying

$$\mathcal{L}(x) = \big\{L_0(x), L_1(x), \ldots, L_{l(x)}(x)\big\},$$

where

$$L_0(x) = \{x\}, \qquad L_1(x) = Adj\big(L_0(x)\big),$$
$$L_i(x) = Adj\big(L_{i-i}(x)\big) - L_{i-2}(x), \quad i = 2, 3, \ldots, l(x).$$

The eccentricity $l(x)$ of x is called the length of $\mathcal{L}(x)$ and the *width* $w(x)$ of $\mathcal{L}(x)$ is defined by

$$w(x) = \max\big\{|L_i(x)| \mid 0 \le i \le l(x)\big\}.$$

In Fig. 3.4 a rooted level structure of the graph of Fig. 3.3 is shown, rooted at the node x_6. Note that $l(x_6) = 3$ and $w(x_6) = 3$.

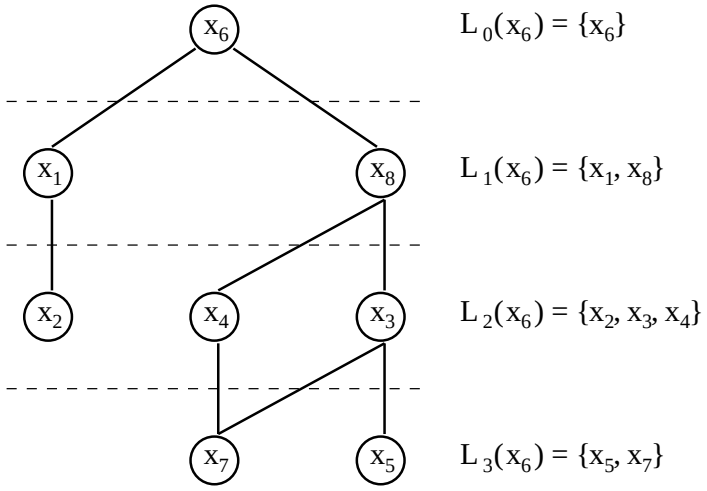

Figure 3.4: An 8-node graph G with $\delta(G) = 5$.

The pseudo-peripheral node finding algorithm is described in the following. This algorithm is essentially a modification of an algorithm due to Gibbs et al. [7]. Using the level structure notation just introduced, the complete procedure is given in Algorithm 5.

Algorithm 5 Pseudo-Peripheral Node Finding.

Input: graph $G = (X, E)$
Output: node x

1. [Initialization] choose an arbitrary node r in X
2. [Generate a level structure] construct the level structure rooted at r:
   $\mathcal{L}(r) = \{L_0(r), L_1(r), \ldots, L_{l(r)}(r)\}$
3. [Shrink last level] choose a node x in $L_{l(r)}(r)$ of minimum degree
4. [Generate a level structure]
   (a) construct the level structure rooted at x:
   $$\mathcal{L}(x) = \{L_0(x), L_1(x), \ldots, L_{l(x)}(x)\}$$
   (b) if $l(x) > l(r)$ set $r = x$ and go to Step 3
5. [Finished] the node x is a pseudo-peripheral node

3.3 The Minimum Degree Ordering

The minimum degree ordering (Tinney [14]) is another popular fill-in-reducing scheme, which corresponds to the Markowitz scheme (Markowitz [10]) for unsymmetric matrices (see Fig. 3.5 for an example of the minimum degree ordering).

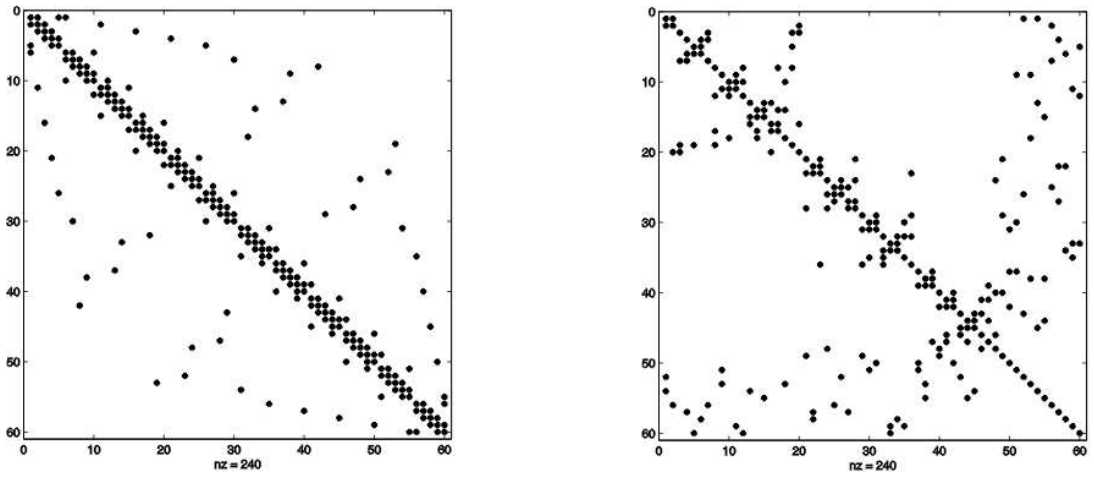

Figure 3.5: Example of a minimum degree ordering.

Let A be a given symmetric matrix and let P be a permutation matrix. Although the nonzero structures of A and PAP^T are different, their number of nonzeros are the same:

$$nnz(A) = nnz(PAP^T).$$

However, the crucial point is that there may be a dramatic difference between $nnz\big(f(A)\big)$ and $nnz\big(f(PAP^T)\big)$ for some permutation P (f is an arbitrary operation on the matrices).

Ideally a permutation P^* is sought that minimizes the number of nonzeros of the permuted matrix. So far, there is no efficient algorithm for getting such an optimal P^* for a general symmetric matrix. Indeed, this problem for A being unsymmetric has been shown to be NP complete (Rose and Tarjan [12]). Thus, one has to rely on heuristics to produce an ordering P with an acceptably small but not necessarily minimum $nnz\big(f(PAP^T)\big)$.

The minimum degree ordering scheme is based on the following observation: Suppose labeled nodes $x_1, x_2, \ldots, x_{i-1}$; the number of nonzeros in the graph for these columns is fixed. In order to reduce the number of nonzeros in the ith column, it is apparent that in the submatrix remaining to be factored, the column with

Algorithm 6 Basic Minimum Degree Ordering.

Input: symmetric graph $G_0 = (X, E)$
Output: ordered graph $G_{|X|}$

1. [Initialization] $i = 1$
2. [Minimum degree selection] in the elimination graph $G_{i-1} = (X_{i-1}, E_{i-1})$, choose a node x_i of minimum degree in G_{i-1}
3. [Graph transformation] form the new elimination graph $G_i = (X_i, E_i)$ by eliminating the node x_i from G_{i-1}
4. [Loop or stop] $i = i + 1$; if $i > |X|$ stop, otherwise go to Step 2

the fewest nonzeros should be moved to become column i. Algorithm 6 describes the above scheme.

Notice that there can be more than one node with the minimum degree at a particular step, which leads to different versions of the algorithm for selecting a special node. Each step of the algorithm involves a graph transformation, which is the most expensive part of the algorithm in terms of implementation. These and some other enhancements are discussed in-depth in George and Liu [6].

The minimum degree ordering is especially interesting because the most promising columns (i. e., columns with the first nonzero at a high index) are ordered at the beginning. So it is sufficient to use the standard reordering only at the first part of the preordered matrix and get almost the same results as using the improved reordering (variant sparse vectors) with the RCM on the whole matrix, but with the great advantage of lower computational cost.

Chapter 4

Summary

This report presents a multi-elimination method to recursively solve large sparse linear systems. Through special reordering and elimination techniques a system of linear equations is reduced iteratively and then solved conventionally (by using LU factorization) in the last step. The solution vector is then built up step-by-step when unrolling the recursion. The benefit of this method is the lower memory consumption when solving the system, and less computational cost if an efficient reordering strategie is used.

Several reordering strategies are introduced. The reorderings use different data of the matrix and vary on the computational cost. Also they result in different structures of the reordered matrix, which influences the sparsity in the following recursion. It seems also favorable to preprocess the matrix with a bandwidth or fill-reducing ordering and apply the reordering on this preordered matrix. Interesting in this context are the reverse Cuthill-McKee and the minimum degree ordering.

Fabianek et al. [4] discusses the implementation and numerical experiments of these algorithms based on data of the AURORA project (Group 6) which deals with large sparse linear systems arising in stochastic optimization (Pflug and Swietanowski [11]).

Bibliography

[1] I. Arany, W.F. Smyth, L. Szoda: *An improved method for reducing the bandwidth of sparse symmetric matrices.* In *Information Processing 71*, Proc. of IFIP Congress, Amsterdam, 1972.

[2] E. Cuthill, J. McKee: *Reducing the bandwidth of sparse matrices.* Proc. 24th Nat. Conf. Assoc. Comput. Mach., ACM Publ. (1969), pp. 157–172.

[3] C.L. Fabianek: *Multistep Elimination Techniques for Sparse Linear Systems Arising in Stochastic Optimization.* Master's thesis, Vienna University of Technology, 2001.

[4] C.L. Fabianek, W.N. Gansterer, C.W. Ueberhuber: *Experiments with a Multi-Elimination Technique for Equilibrium Systems.* Technical Report AURORA TR2002-05, Vienna University of Technology, 2002.

[5] W.N. Gansterer, J. Schneid, C.W. Ueberhuber: *Properties of Equilibrium Systems.* Technical Report AURORA TR2002-03, Vienna University of Technology, 2002.

[6] A. George, J.W. Liu: *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, New Jersey, 1981.

[7] N.E. Gibbs, W.G. Poole, P.K. Stockmeyer: *An algorithm for reducing the bandwidth and profile of a sparse matrix.* SIAM J. Numer. Anal. 13 (1976), pp. 236–250.

[8] P. Goldenberg, S.R.P. Medeiros, P.M. Pimenta: *An Algorithm for Profile and Wavefront Reduction of Sparse Matrices with a Symmetric Structure.* Engineering Computations 10 (1993), pp. 257–266.

[9] W.H. Liu, A.H. Sherman: *Comparative Analysis of the Cutthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices.* SIAM J. Math. Anal. 13 (1976)(9), pp. 198–213.

[10] H.M. Markowitz: *The elimination form of the inverse and its application to linear programming.* Management Science 3 (1957), pp. 255–269.

[11] G.C. Pflug, A. Swietanowski: *Parallel Decision Support for Financial Management Under Uncertainty.* Technical Report AURORA TR1998-07, University of Vienna, 1998.

[12] D. J. Rose, R. E. Tarjan: *Algorithmic aspects of vertex elimination*. In *Proc. 7th Annual Symposium on the Theory of Computing*, 1975, pp. 245–254.

[13] Y. Saad: *Iterative methods for sparse linear systems*. PWS Publishing Companies, Boston, Mass., 1996.

[14] W. F. Tinney: *Comments on using sparsity techniques for power system problems*. In *Sparse Matrix Proceedings*, 1969, IBM Research Report.