On the performance of minimum degree and minimum local fill heuristics in circuit simulation

Gunther Reißig

Before you cite this report, please check whether it has already appeared as a journal paper (http://www.reiszig.de/gunther, or e-mail to gr@ieee.org). Thank you.

The author is with Massachusetts Institute of Technology, Room 66-363, Dept. of Chemical Engineering, 77 Massachusetts Avenue, Cambridge, MA 02139, U.S.A.. E-Mail: gr@ieee.org. Work supported by Deutsche Forschungsgemeinschaft. Part of the results of this paper have been obtained while the author was with Infineon Technologies, München, Germany.

Abstract

Data structures underlying local algorithms for obtaining pivoting orders for sparse symmetric matrices are reviewed together with their theoretical background. Recently proposed heuristics as well as improvements to them are discussed and their performance, mainly in terms of the resulting number of factorization operations, is compared with that of the Minimum Degree and the Minimum Local Fill algorithms. It is shown that a combination of Markowitz' algorithm with these symmetric methods applied to the unsymmetric matrices arising in circuit simulation yields orderings significantly better than those obtained from Markowitz' algorithm alone, in some cases at virtually no extra computational cost.

I. Introduction

When the behavior of an electrical circuit is to be simulated, numerical integration techniques are usually applied to its equations of modified nodal analysis. This requires the solution of systems of nonlinear equations, and, in turn, the solution of numerous linear equations of the form

$$Ax = b, \tag{1}$$

where A is a real $n \times n$ matrix, typically nonsingular, and $x, b \in \mathbb{R}^n$ [1–3].

To solve (1), sometimes with A, x, and b complex, is also necessary in other analyses, and the efficiency by which this is done determines the quality of simulation tools as a whole to a great extent.

Although the coefficient matrices of the linear equations to be solved are unsymmetric and indefinite, these equations are solved directly without pivoting for numerical accuracy. In fact, it is assumed that one can, from a numerical point of view, solve the equation

$$(PAP^T)y = Pb \tag{2}$$

by forward elimination and back substitution for any $n \times n$ permutation matrix P and then set $x = P^T y$. That is, the diagonal entries may be chosen as pivots, in any order [3–5]. Nevertheless, it is well known that a deliberate choice of the pivoting order may be crucial for successfully solving equation (1), as we will explain below.

Typically, the coefficient matrices of the linear equations to be solved are very large and extremely sparse, i.e., only few of their entries are nonzero. Exploiting that sparsity will keep the computational complexity low, since the zeros do not need to be stored and

computations on them do not need to be performed. Unfortunately, as the elimination phase progresses, new nonzeros are introduced:

Usually, the coefficient matrix PAP^T is overwritten by the matrix $L + U - \mathrm{id}_n$, where L and U are factors of PAP^T,

$$LU = PAP^T,$$

L being unit lower triangular and U being upper triangular, and id_n is the $n \times n$ identity matrix [3, 6, 7]. While even for a fixed pivoting order, various variants of Gaussian elimination are known [8], we assume here that the kth step of Gaussian elimination, $k \in \{1, \ldots, n-1\}$, creates an $n \times n$ matrix $A^{(k)}$ defined by

$$A^{(k)}_{k,j} = A^{(k-1)}_{k,j}, \tag{3}$$

$$A^{(k)}_{i,k} = A^{(k-1)}_{i,k} / A^{(k-1)}_{k,k}, \tag{4}$$

$$A^{(k)}_{i,j} = A^{(k-1)}_{i,j} - A^{(k)}_{i,k} \cdot A^{(k-1)}_{k,j} \tag{5}$$

for all $i, j \in \{k+1, \ldots, n\}$, where we have set $A^{(0)} = PAP^T$. Thereby, the matrix $A^{(k-1)}$ is overwritten by $A^{(k)}$, which eventually results in the matrix $A^{(n-1)} = L + U - \mathrm{id}_n$ at the place of PAP^T [7].

If nonzeros do not cancel out in (5), which we assume throughout this paper, and if $A^{(k)}_{i,j}$ is nonzero, then so are the entries $A^{(k')}_{i,j}$ for all $k' > k$. However, $A^{(k')}_{i,j}$ may be nonzero even if $A^{(k)}_{i,j}$ is not, which results in nonzeros of the matrix $L + U - \mathrm{id}_n$ at positions where the coefficient matrix PAP^T has zeros. Those newly created nonzeros are called *fill-ins*, and their total is called *fill*.

In addition to increasing the storage requirement, fill-ins, once created, enter into the calculation of subsequent elimination steps. As the amount of fill heavily depends on the permutation matrix P in equation (2), that matrix determines the computational complexity of solving equation (1) to a high degree.

In circuit simulation, the coefficient matrices of all linear equations to be solved usually have the same zero-nonzero pattern. Therefore, once a suitable permutation matrix has been determined, a data structure capable of holding the nonzeros of the coefficient matrices and their factors is set up, which, together with the permutation matrix, will

be used for solving all linear equations appearing during the simulation [3, 6, 9]. Hence, choosing a permutation matrix that leads to low fill is extremely important for the overall performance of circuit simulation software.

Various heuristics have been proposed, which aim at acceptable low, rather than minimum, fill [10], as minimizing the fill appears to be harder than solving the linear equations at hand without taking advantage of their sparsity [11, 12].

This paper focuses on so-called *local* algorithms, which mimic Gaussian elimination defined by (3)-(5) based on the zero-nonzero pattern of the coefficient matrix alone, again, under the assumption that nonzeros do not cancel out in (5). These methods determine a pivoting order, or equivalently, a permutation matrix by choosing one pivot per elimination step: First, a pivot is chosen from the diagonal that minimizes some scoring function. Then, the variable corresponding to the pivot column is eliminated using the equation containing the pivot, thereby forming a new zero-nonzero pattern that represents all fill-ins created so far, in addition to the nonzeros of the coefficient matrix itself. Finally, the pivot row and column are removed, and the procedure above is recursively applied to the zero-nonzero pattern thus obtained.

If, for example, the pivots are chosen down the diagonal, the zero-nonzero patterns arising in the above procedure are exactly those of the lower right $(n-k) \times (n-k)$ submatrix of the matrices $A^{(k)}$ defined by (3)-(5).

MARKOWITZ was the first to propose heuristics that aim at the creation of low fill in Gaussian elimination [13]. We present his algorithms in the setting of this paper, pivot selection from the diagonal, though originally, MARKOWITZ did not restrict to that case.

In the *Minimum Local Fill (MF)* algorithm, the score of a diagonal entry is the number of fill-ins that would be created in the next elimination step if that entry was chosen as the next pivot. Although mentioning that strategy, MARKOWITZ recommended using the following scoring function instead, as its values are easier to obtain [13]:

After k elimination steps, let c_i and r_i be the number of off-diagonal nonzeros in the ith column and row, respectively, of the $(n-k) \times (n-k)$ zero-nonzero pattern that evolved from the preceding eliminations. Obviously, the product $c_i r_i$, which is called the *Markowitz product*, is an upper bound on the number of fill-ins introduced when the ith variable in the

current scheme is eliminated next. To chose a diagonal entry with minimum Markowitz product as the next pivot has become known as *Markowitz' algorithm*.

If the coefficient matrix A in (1) is structurally symmetric, i.e., $A_{i,j}$ is nonzero iff $A_{j,i}$ is, $1 \leq i, j \leq n$, then all zero-nonzero patterns arising in any local ordering algorithm are symmetric. In particular, Markowitz' algorithm always selects a diagonal entry in a row with minimum number of nonzeros as the next pivot, which is exactly what TINNEY and WALKER independently proposed for structurally symmetric coefficient matrices [14]. In PARTER's interpretation of Gaussian elimination [15], that strategy corresponds to the selection of a vertex of minimum degree in some graph, hence the name *Minimum Degree (MD)* algorithm [16] for the symmetric case of Markowitz' algorithm.

TINNEY and WALKER further proposed a symmetric version of the MF algorithm [14], which is also called the *Minimum Deficiency* algorithm [16].

A drawback of the MF algorithm is its immense computational cost, which may be two orders of magnitude higher than that of a version of the MD algorithm [17].

On the other hand, it has been found across virtually all application areas that the various extensions of Markowitz' algorithm and the MF algorithm proposed yield only marginally better orderings than the original heuristic of MARKOWITZ and its symmetric descendant [10, 18]. In particular, in circuit simulation, the MF algorithm has been reported to save at most 5% in factorization operations compared with Markowitz' algorithm [3].

Only recently, the MF algorithm as well as newly proposed variants of both the MF and MD algorithms have been found to yield significantly better orderings than the MD algorithm on certain test suites of symmetric matrices [17,19–22], and the running times of some of those algorithms are just in the order of those of the MD algorithm [17, 19, 21, 22].

It has been reported that the savings in fill of the variants of the MF algorithm compared with the MD algorithm heavily depend on the field of application and that these savings increase with increasing problem size [19]. Unfortunately, extensive tests and comparisons of those variants on matrices derived from circuits of a size representing today's simulation tasks are missing.

When applied to structurally symmetric coefficient matrices, ordering methods that take

advantage of symmetry, so-called *symmetric* methods, are considerably more efficient than their competitors. In order to benefit from those savings even if the coefficient matrix A in equation (1) is unsymmetric, it has been proposed to apply symmetric ordering methods to the matrix $|A| + |A|^T$, where X^T is the transpose of X and $|X|$ denotes the matrix obtained from X by substituting each entry by its absolute value, e.g. [23]. That trick allows for the following refinement [24]:

Initially, diagonal entries with Markowitz product zero are chosen as pivots, as many as possible. Since the corresponding elimination steps do not create any fill, the zero-nonzero pattern obtained after those steps is that of a submatrix $\widetilde{A}$ obtained from A by removing the pivot rows and columns. Now, a symmetric ordering method is applied to $|\widetilde{A}| + |\widetilde{A}|^T$, thereby completing the pivoting order for A. That way, much of the unsymmetry of A is removed by the initial steps of Markowitz' algorithm so that $\widetilde{A}$ will be symmetric or nearly so.

The main purpose of this paper is to verify that in circuit simulation, the above combination of Markowitz' algorithm with local symmetric ordering methods is useful and to compare the performance of symmetric methods in this setting.

To this end, we review PARTER's interpretation of Gaussian elimination in terms of graphs [15] in Section II. Further, we discuss data structures underlying local symmetric ordering methods together with their theoretical background and present details of our implementation.

In Section III, recently proposed local symmetric ordering heuristics are reviewed and improvements to them are considered.

In Section IV, we compare the performance of the MD and MF algorithms and the algorithms from Section III on a test set of 33 matrices. 15 of those matrices resulted from the application of initial steps of Markowitz' algorithm and symmetrization as described above to matrices extracted from the circuit simulator TITAN [2].

Our tests show that the above combination of Markowitz' algorithm with local symmetric ordering methods yields orderings that are, in terms of the resulting number of factorization operations, significantly better than those obtained from Markowitz' algorithm alone, in some cases at virtually no extra computational cost.

II. Data structures of local symmetric ordering methods and their implementation

Throughout this section, A is a structurally symmetric $n \times n$ matrix, i.e., the entry $A_{i,j}$ is nonzero iff $A_{j,i}$ is, for all i and j, $1 \le i, j \le n$.

A. Gaussian elimination in terms of graphs

A *graph* is an ordered pair (V, E) of a finite set V of *vertices* and a set E of *edges*, $E \subseteq \big\{\{v, w\} \subseteq V \mid v \neq w\big\}$. In order to simplify the notation, we further assume $V \cap \mathcal{P}(V) = \emptyset$, where $\mathcal{P}(V)$ is the power set of V.

Two graphs (V, E) and (V', E') are *isomorphic* if there is a bijection $\pi\colon V \to V'$ such that $E' = \big\{\{\pi(v), \pi(w)\} \mid \{v, w\} \in E\big\}$. For any such bijection π, we denote the mapping that acts on the graphs by π^*, i.e., $\pi^*(V, E) = (V', E')$.

Two vertices $v, w \in V$ are *adjacent* in the graph G, $G = (V, E)$, if $\{v, w\} \in E$. For $w \in V$ and $W \subseteq V$, $\mathrm{adj}_G(w)$ and $\mathrm{adj}_G(W)$ denote the *adjacent set* of w and W, respectively, that is,

$$\begin{aligned} \mathrm{adj}_G(w) &= \big\{u \in V \mid \{u, w\} \in E\big\}, \\ \mathrm{adj}_G(W) &= \bigcup_{u \in W} \mathrm{adj}_G(u) \setminus W. \end{aligned}$$

For $X \in V \cup \mathcal{P}(V)$, we denote the *degree of* X *in* G by $\deg_G(X)$,

$$\deg_G(X) = |\,\mathrm{adj}_G(X)|,$$

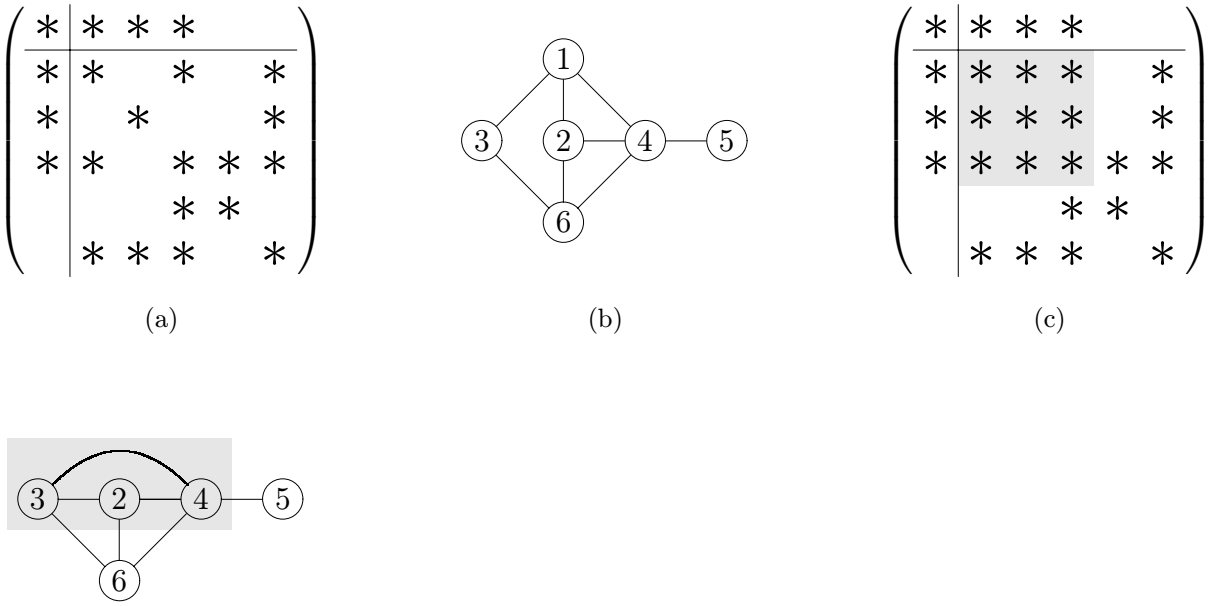
where $|\cdot|$ denotes cardinality.

A vertex $v \in V$ and an edge $e \in E$ are *incident* if $v \in e$.

The *graph of* A, denoted $\mathcal{G}(A)$, is the graph (V, E) defined by

$$\begin{aligned} V &= \{1, \ldots, n\}, \\ E &= \big\{\{i, j\} \mid A_{i,j} \neq 0, i \neq j\big\}. \end{aligned}$$

Note that the vertices of $\mathcal{G}(A)$ represent diagonal entries or row and column indices, while its edges correspond to off-diagonal nonzeros. Note also that $\mathcal{G}(PAP^T)$ and $\mathcal{G}(A)$ are isomorphic for any $n \times n$ permutation matrix P.



(a) (b) (c)

(d)

Fig. 1. Illustration of Example 1. (a) Zero-nonzero pattern of a 6×6 matrix A. (Nonzeros are denoted by asterisks ($*$).) (b) The graph $\mathcal{G}(A)$ of A. (c) Zero-nonzero pattern after a step of Gaussian elimination using $A_{1,1}$ as pivot. (d) Elimination graph $(\mathcal{G}(A))_1$. (In (c) and (d), the shaded region corresponds to the clique created by the elimination.)

Let $G = (V, E)$ be a graph.

The *elimination graph* G_v, obtained by *eliminating* $v \in V$ *from* G, is defined by $G_v = (V \setminus \{v\}, E')$ and

$$E' = \big\{\{x, y\} \in E \mid v \notin \{x, y\}\big\} \cup \big\{\{x, y\} \subseteq \mathrm{adj}_G(v) \mid x \neq y\big\}.$$

In other words, G_v is obtained from G by removing v and its incident edges from G, and connecting all vertices previously adjacent to v [15], thereby creating a set $\mathrm{fill}_G(v)$ of new edges or *fill-ins*,

$$\mathrm{fill}_G(v) = E' \setminus E.$$

Example 1 Let A be a 6×6 matrix with the zero-nonzero pattern given in Fig. 1(a). The set of vertices of $\mathcal{G}(A)$, the graph of A shown in Fig. 1(b), is $\{1, 2, 3, 4, 5, 6\}$, and the set of its edges is

$$\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\}, \{2, 6\}, \{3, 6\}, \{4, 5\}, \{4, 6\}\}. \tag{6}$$

Obviously, $\mathrm{adj}_{\mathcal{G}(A)}(1) = \{2, 3, 4\}$ and $\mathrm{adj}_{\mathcal{G}(A)}(\{1, 2\}) = \{3, 4, 6\}$.

A step of Gaussian elimination with pivot $A_{1,1}$ creates fill-ins at the positions $(2, 3)$, $(4, 3)$, $(3, 2)$, and $(3, 4)$, see Fig. 1(c). The elimination of vertex 1 from $\mathcal{G}(A)$ creates the new edges $\{2, 3\}$ and $\{3, 4\}$, i.e., $\mathrm{fill}_{\mathcal{G}(A)}(1) = \{\{2, 3\}, \{3, 4\}\}$. The resulting elimination graph $(\mathcal{G}(A))_1$ shown in Fig. 1(d) is isomorphic to the graph of a matrix that has the lower right 5×5 subpattern of the pattern from Fig. 1(c) as its zero-nonzero pattern and has the egde set $\{\{2, 3\}, \{2, 4\}, \{2, 6\}, \{3, 4\}, \{3, 6\}, \{4, 5\}, \{4, 6\}\}$. □

It is easy to show that $(G_v)_w = (G_w)_v$ whenever $v, w \in V$ and $v \neq w$, i.e., the elimination graph resulting from the elimination of a set $\{v_1, \ldots, v_k\} \subseteq V$ of distinct vertices from G does not depend on the order in which these vertices are eliminated. Therefore, we may denote that elimination graph by $G_{\{v_1,\ldots,v_k\}}$,

$$G_{\{v_1,\ldots,v_k\}} = (\cdots(G_{v_1})\cdots)_{v_k}.$$

For simplicity, we further define $G_\emptyset = G$.

Choosing pivots down the diagonal in PAP^T for some $n \times n$ permutation matrix P corresponds to selecting vertices of $\mathcal{G}(A)$ in the order $\pi(1)$, $\pi(2)$, ..., $\pi(n)$ for some bijection $\pi : V \to V$, which we call the *pivoting order*. Thus, local algorithms that mimic Gaussian elimination to select pivots are equivalent to the selection of vertices as follows:

Input:	Graph $G = (V, E)$, scoring function s.
Step 1:	$S := \emptyset$.
Step 2:	Pick $v \in V \setminus S$ with $s(v, G) = \min_{w \in V \setminus S} s(w, G)$.
Step 3:	$S := S \cup \{v\}$, $\pi(\lvert S\rvert) := v$, $G := G_v$.
Step 4:	If $S \neq V$, goto Step 2.
Output:	Pivoting order π.

In particular, if the input graph G equals $\mathcal{G}(A)$, the above algorithm is the MD and the MF algorithm for $s(v, G) = \deg_G(v)$ and $s(v, G) = |\,\mathrm{fill}_G(v)|$, respectively.

B. Bookkeeping scores

In this and the remaining paragraphs of this section, let G be a graph, $G = (V, E)$.

Usually, when a vertex is eliminated, the scores of only few vertices change. Indeed, for all scoring functions considered in this paper, eliminating $v \in V$ from G does not change the score of $w \in V$ if $w \neq v$ and $w \notin \mathrm{adj}_G(\mathrm{adj}_G(v)) \cup \mathrm{adj}_G(v)$. In particular, $\deg_G(w) = \deg_{G_v}(w)$ if $w \notin \mathrm{adj}_G(v)$, and $\mathrm{fill}_G(w) = \mathrm{fill}_{G_v}(w)$ if $w \notin \mathrm{adj}_G(\mathrm{adj}_G(v)) \cup \mathrm{adj}_G(v)$. Hence, it is important for the performance of ordering algorithms to update scores and retrieve vertices of minimum score efficiently.

A *heap* is a data object that is particularly well suited for that purpose, as it is capable of holding any subset of n vertices together with their scores such that inserting and deleting a vertex requires at most $O(\log n)$ operations, and retrieving a vertex with minimum score requires $O(1)$ operations [25].

The heap may be initialized with the scores of the vertices of $\mathcal{G}(A)$. Those scores may then be updated later at low cost.

C. Clique representations of elimination graphs

A *clique* is a graph in which any two distinct vertices are adjacent [26]. The vertex set of a clique will also be called a clique wherever that is unambiguous.

Note that cliques are isomorphic to graphs of *full* matrices, i.e., matrices that do not have zero entries.

A set $\mathcal{C} \subseteq \mathcal{P}(V)$ is called a *clique representation* of G if $E = \big\{\{v, w\} \subseteq C \mid C \in \mathcal{C}, v \neq w\big\}$. Hence, clique representations of $\mathcal{G}(A)$ correspond to coverings of the nonzeros of A by full symmetric minors, and the edge set of any graph is a clique representation of that graph.

If $\mathcal{C}$ is a clique representation of G and $v \in V$, then

$$\mathrm{adj}_G(v) = \bigcup_{\substack{C \in \mathcal{C} \\ v \in C}} C \setminus \{v\}, \tag{7}$$

and the set $\mathcal{C}'$ defined by

$$\mathcal{C}' = \{C \in \mathcal{C} \mid v \notin C\} \cup \{\mathrm{adj}_G(v)\} \tag{8}$$

is a clique representation of G_v. Hence, clique representations of elimination graphs are obtained easily. Moreover, as the number of elements in the newly created clique $\mathrm{adj}_G(v)$ is less than the sum of the number of elements of the cliques it is made of, the amount of storage required to represent those elimination graphs will never exceed that required to store the original graph $\mathcal{G}(A)$.

The above representation technique was first proposed by George [27] and independently by Speelpenning [28, 29].

If $C \in \mathcal{C} \cap \mathcal{C}'$ and $C \subseteq \mathrm{adj}_G(v)$, then $\mathcal{C}' \setminus \{C\}$ is still a clique representation of G_v. Removing such cliques, an idea due to Duff and Reid [30], may speed up the ordering process and has become known under the label *element absorption* [31].

Clique representations may be easily maintained and handled as the disjoint union of two sets, $\mathcal{C}_0$ and $\mathcal{C}_1$. $\mathcal{C}_0 \subseteq E$ contains some of the edges of the current elimination graph G that have already been edges of $\mathcal{G}(A)$, and $\mathcal{C}_1$ contains cliques created by eliminating vertices. For simplicity, we call the pair $(\mathcal{C}_0, \mathcal{C}_1)$ a clique representation as well.

At the beginning, when $G = \mathcal{G}(A)$, we may have $\mathcal{C}_0 = E$ and $\mathcal{C}_1 = \emptyset$. Later, whenever a vertex v is eliminated from G, a clique representation $(\mathcal{C}_0', \mathcal{C}_1')$ of G_v may be obtained from $(\mathcal{C}_0, \mathcal{C}_1)$,

$$\mathcal{C}_0' = \mathcal{C}_0 \setminus \mathcal{P}(\mathrm{adj}_G(v) \cup \{v\}), \tag{9}$$

$$\mathcal{C}_1' = \{C \in \mathcal{C}_1 \mid v \notin C\} \cup \{\mathrm{adj}_G(v)\}, \tag{10}$$

thereby applying the technique of element absorption to $\mathcal{C}_0$ only. That way, cliques of $\mathcal{C}_0$ never get covered by cliques of $\mathcal{C}_1$, i.e., $C_0 \setminus C_1 \neq \emptyset$ for all $C_0 \in \mathcal{C}_0$ and all $C_1 \in \mathcal{C}_1$.

Example 1 (continued) From the set (6) of edges of the graph $\mathcal{G}(A)$ from Fig. 1(b), a trivial clique representation of that graph, we obtain the clique representation

$$\{\{2,4\},\{2,6\},\{3,6\},\{4,5\},\{4,6\},\{2,3,4\}\} \tag{11}$$

of the elimination graph $(\mathcal{G}(A))_1$ from Fig. 1(d) by (8).

Let $\mathcal{C}_1 = \emptyset$ and $\mathcal{C}_0$ be the set (6) of edges of $\mathcal{G}(A)$ so that $(\mathcal{C}_0, \mathcal{C}_1)$ is a clique representation

of $\mathcal{G}(A)$. Then (9)-(10) yield another clique representation $(\mathcal{C}'_0, \mathcal{C}'_1)$ of $(\mathcal{G}(A))_1$,

$$\mathcal{C}'_0 = \{\{2,6\},\{3,6\},\{4,5\},\{4,6\}\}, \tag{12}$$

$$\mathcal{C}'_1 = \{\{2,3,4\}\}, \tag{13}$$

which, in contrary to (11), does not contain the clique $\{2,4\}$. □

D. Indistinguishable vertices and mass elimination

The vertices $v, w \in V$ are *indistinguishable in* G, $v \underset{G}{\sim} w$, if

$$\mathrm{adj}_G(v) \cup \{v\} = \mathrm{adj}_G(w) \cup \{w\}.$$

Hence, the vertices v and w are indistinguishable in $\mathcal{G}(A)$ iff the zero-nonzero pattern of the rows v and w of A coincide, i.e., $A_{v,i} \neq 0$ iff $A_{w,i} \neq 0$ for all i.

Obviously, $\underset{G}{\sim}$ is an equivalence relation, and we may take advantage of its properties as follows:

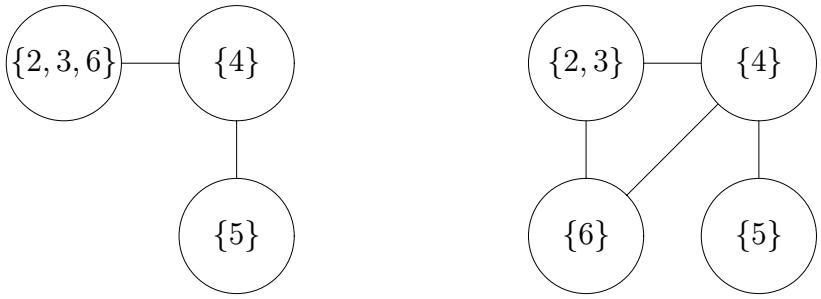
First, $v \underset{G}{\sim} w$ implies $\deg_{G_v}(w) = \deg_G(v) - 1$ for $v, w \in V$, $v \neq w$. In particular, if $v \underset{G}{\sim} w$ and v is a vertex of minimum degree in G, then w is such a vertex in G_v.

Furthermore, if $v \underset{G}{\sim} w$ and $x \in V \setminus \{v, w\}$, then $v \underset{G_x}{\sim} w$. That is, if two vertices are indistinguishable in some elimination graph, they remain indistinguishable in all succeeding elimination graphs until v or w is eliminated [32].

Thus, in the MD algorithm, the vertices in $[v]_{\underset{G}{\sim}}$ are eliminated immediately, in succession. Hence, we may dispense with updating degrees until the whole set is eliminated, a technique due to George and McIntyre [33] and known as *mass elimination* [31]. Since $v \underset{G}{\sim} w$ implies $\mathrm{fill}_{G_v}(w) = \emptyset$ for $v \neq w$, that technique may be combined with other scoring functions as well.

If π is that permutation on V that just exchanges two indistinguishable vertices $v, w \in V$, then $\pi^*(G) = G$, which implies $\deg_G(v) = \deg_G(w)$ and $\mathrm{fill}_G(v) = \mathrm{fill}_G(w)$. Moreover, for any reasonable scoring function, it should suffice to determine the score of one of v and w only, even if the two scores do not coincide. Hence, we may maintain the quotient graph $G/\underset{G}{\sim}$ rather than G itself, where $G/\underset{G}{\sim} = (V/\underset{G}{\sim}, E')$ and

$$E' = \big\{\{[v]_{\underset{G}{\sim}}, [w]_{\underset{G}{\sim}}\} \mid \{v,w\} \in E, [v]_{\underset{G}{\sim}} \neq [w]_{\underset{G}{\sim}}\big\}.$$



(a) (b)

Fig. 2. Illustration of Example 1. (a) The quotient graph $(\mathcal{G}(A))_1 / \underset{(\mathcal{G}(A))_1}{\sim}$. (b) The quotient graph $(\mathcal{G}(A))_1 / \approx$.

While detecting all pairs of indistinguishable vertices is expensive, some may be found easily, since

$$\{C \in \mathcal{C}_1 \mid v \in C\} = \{C \in \mathcal{C}_1 \mid w \in C\}, \tag{14}$$

$$\bigcup_{\substack{C \in \mathcal{C}_0 \\ v \in C}} C = \bigcup_{\substack{C \in \mathcal{C}_0 \\ w \in C}} C \neq \emptyset \vee \left(\{C \in \mathcal{C}_1 \mid v \in C\} \neq \emptyset \wedge \bigcup_{\substack{C \in \mathcal{C}_0 \\ v \in C}} C \setminus \{v\} = \bigcup_{\substack{C \in \mathcal{C}_0 \\ w \in C}} C \setminus \{w\} \right) \tag{15}$$

together imply $v \underset{G}{\sim} w$ for any clique representation $(\mathcal{C}_0, \mathcal{C}_1)$ of G, where $\vee$ and $\wedge$ denote disjunction and conjunction, respectively. Therefore, the quotient graph $G/\approx$ for some equivalence relation $\approx \subseteq \underset{G}{\sim}$, rather than $G/\underset{G}{\sim}$ itself, should be maintained.

Since the operation of forming the quotient and that of eliminating $[v]_\approx$ commute, we may not only store $G/\approx$, but also eliminate the vertex $[v]_\approx$ from $G/\approx$ rather than the set of vertices $[v]_\approx$ from G.

Example 1 (continued) The graph $\mathcal{G}(A)$ from Fig. 1(b) does not have two distict indistinguishable vertices. However, the vertices 2, 3, and 6 are indistinguishable in the elimination graph $(\mathcal{G}(A))_1$ from Fig. 1(d), since $\mathrm{adj}_{(\mathcal{G}(A))_1}(v) \cup \{v\} = \{2, 3, 4, 6\}$ for all $v \in \{2, 3, 6\}$. In particular, the quotient graph $(\mathcal{G}(A))_1 / \underset{(\mathcal{G}(A))_1}{\sim}$ shown in Fig. 2(a) has the vertex set $\{\{2, 3, 6\}, \{4\}, \{5\}\}$ and the edge set $\{\{\{2, 3, 6\}, \{4\}\}, \{\{4\}, \{5\}\}\}$.

Looking for indistinguishable vertices in $(\mathcal{G}(A))_1$ by checking conditions (14),(15) on the clique representation (12),(13) results in $2 \approx 3$ and $2 \not\approx 6$. In particular, the quotient

graph $(\mathcal{G}(A))_1 / \approx$ shown in Fig. 2(b) has the vertex set $\{\{2,3\},\{4\},\{5\},\{6\}\}$ and the edge set $\{\{\{2,3\},\{4\}\},\{\{2,3\},\{6\}\},\{\{4\},\{5\}\},\{\{4\},\{6\}\}\}$. □

E. Outmatched vertices and incomplete score update

The vertex $v \in V$ *outmatches* $w \in V$ *in* G if

$$\mathrm{adj}_G(v) \cup \{v\} \subseteq \mathrm{adj}_G(w) \cup \{w\}.$$

Hence, the vertex v outmatches the vertex w in $\mathcal{G}(A)$ iff the zero-nonzero pattern of the row v of A is a subset of that of the row w, i.e., $A_{v,i} \neq 0$ implies $A_{w,i} \neq 0$ for all i.

Obviously, if v outmatches w in G, then $\deg_G(v) \leq \deg_G(w)$ and $\mathrm{fill}_G(v) \subseteq \mathrm{fill}_G(w)$. Furthermore, if $x \in V \setminus \{v,w\}$, then v outmatches w in G_x, i.e., v outmatches w until one of these two vertices is eliminated.

Consequently, it is often advisable to eliminate v before w, and hence, the score of w does not need to be calculated until v is eliminated. Note, however, that the score of v may exceed that of w even for reasonable scoring functions.

The above technique of *incomplete score update*, which was first used in the Yale Sparse Matrix Package [34], speeds up the ordering process.

While detecting all outmatched vertices is expensive, a number of easily checked conditions may be used to detect many of them:

Let $(\mathcal{C}_0, \mathcal{C}_1)$ be a clique representation of G, $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1$, and let $\mathrm{adj}_G(v) \neq \emptyset$. Then v outmatches w in G if

$$\{C \in \mathcal{C} \mid v \in C\} \subseteq \{C \in \mathcal{C} \mid w \in C\},$$

i.e., v outmatches all vertices in $\bigcap_{\substack{C \in \mathcal{C} \\ v \in C}} C$. In particular, if v is *k-adjacent*, i.e., if $\{C \in \mathcal{C}_0 \mid v \in C\} = \emptyset$ and $|\{C \in \mathcal{C}_1 \mid v \in C\}| = k$, then v outmatches all vertices in $\bigcap_{\substack{C \in \mathcal{C}_1 \\ v \in C}} C$.

F. Multiple elimination

A set $I \subseteq V$ is *independent in* G if $v \notin \mathrm{adj}_G(w)$ for all $v, w \in I$. An independent set I of G is *maximal with w.r.t.* $W \subseteq V$ if $I \subseteq W$ and I is not properly contained in any other independent set $J \subseteq W$ of G.

Let $W \subseteq V$ be the set of vertices of minimum score in G. An independent set $I \subseteq W$ (that is maximal w.r.t. W) is called a *(maximal) independent set of vertices of minimum score*.

Multiple elimination is a technique due to Liu that reduces the number of score updates as follows [35]:

Step 2 of the algorithm in Paragraph II-A is modified so that a maximal independent set I of vertices of minimum score in the current elimination graph is selected, rather than just a single vertex. Then, all vertices in I are eliminated before updating any scores.

That technique eliminates vertices of low but not necessarily minimum score. However, though that is in particular true for the MD algorithm, it has been reported that the incorporation of multiple elimination into that algorithm often reduces and only rarely increases the fill determined by the orderings created [35].

G. Implementation

We have implemented the MD and MF algorithms as well as their variants to be described in Section III in the C language as an extension to the ordering methods of the SPOOLES library [36], with several modifications to the original data structures:

Let G be the current elimination graph, $G = (V, E)$.

We store the quotient graphs G/X for suitable equivalence relations $X \subseteq V \times V$ to be specified below. Each vertex $[v]_X$ of G/X is referred to by a fixed representative, say, v. Each set of vertices of G/X is stored as a set of representatives, and the score heap stores representatives as well. The number of elements in the class $[v]_X$ is stored with the representative, v, and a pointer to v is stored with all other elements of $[v]_X$.

For each clique representation $(\mathcal{C}_0, \mathcal{C}_1)$ of G/X maintained, the elements of the cliques of $\mathcal{C}_1$ are stored in arrays in ascending order. A linked list of pointers to cliques of $\mathcal{C}_1$ containing $[v]_X$ is associated with each vertex $[v]_X$ of G/X. The set $\bigcup_{\substack{C \in \mathcal{C}_0 \\ [v]_X \in C}} C \setminus \{[v]_X\}$ of vertices is stored for any vertex $[v]_X$ of G/X as an array in ascending order. That way, the adjacency set of $[v]_X$ may be obtained in explicit form according to (7) by scanning the array and the list associated to $[v]_X$.

Let $\approx \subseteq \underset{G}{\sim}$ be the equivalence relation the classes of which are sets of indistinguishable vertices of G already detected, let $(\mathcal{C}_0, \mathcal{C}_1)$ be a clique representation of $G/\approx$, and let

$R_1 \subseteq V/\approx$ be the union of those cliques in $\mathcal{C}_1$ that have been created by the most recent elimination of an independent set of vertices. (At the beginning, let $G = \mathcal{G}(A)$, $\approx$ the identity mapping on $\{1, \ldots, n\}$, $\mathcal{C}_0 = E$, $\mathcal{C}_1 = \emptyset$, and $R_1 = V$.) The elements of R_1 are not in the score heap.

Indistinguishable vertices of $G/\approx$ are detected by checking the conditions (14)-(15) for any two vertices $v, w \in R_1$ by first sorting all vertices in R_1 according to some hash function [25], which is similar to the one used in [37,38], but puts 2-adjacent vertices first, and then comparing vertices with the same hash value.

Let $\simeq \subseteq V/\approx \times V/\approx$ be the corresponding equivalence relation, i.e., $v \simeq w$ iff (14)-(15) hold. Since indistinguishability of vertices in $G/\approx$ implies that of their representatives in G, we have obtained an equivalence relation $\approx'$ so that $\approx \subseteq \approx' \subseteq \underset{G}{\sim} \subseteq V \times V$, and $v \approx' w$ iff $[v]_\approx \simeq [w]_\approx$ for $v, w \in V$.

A clique representation $(\mathcal{C}'_0, \mathcal{C}'_1)$ of $G/\approx'$ is obtained from $(\mathcal{C}_0, \mathcal{C}_1)$ by merging $w \in V/\approx$ into $v \in V/\approx$ whenever $v \simeq w$ and $v \neq w$, i.e., $\mathcal{C}'_i = \{f(C) \mid C \in \mathcal{C}_i\}$, $i \in \{0, 1\}$, where $f : V/\approx \to V/\approx' : [x]_\approx \mapsto [x]_{\approx'}$. We further calculate $R'_1 = f(R_1)$.

Technically, this requires the removal of cliques from $\mathcal{C}_0$ that contain w and to remove w from R_1 and all cliques of $\mathcal{C}_1$, since cliques are stored as arrays of representatives.

At the beginning, when $G = \mathcal{G}(A)$, we dispense with merging indistinguishable vertices unless $|V/\approx'| \leq 0.75n$. In that case, $\approx' = \approx$ is the identity mapping on V, $\simeq$ is that on $V/\approx$, $\mathcal{C}'_0 = E$, $\mathcal{C}'_0 = \emptyset$, and $R_1 = R'_1$. Further, we sort the array holding R'_1 into some random order to simulate initial row and column permutations.

We calculate scores of vertices of $G/\approx'$. (The MD and MF scores may be easily expressed in terms of $G/\approx'$.) We always round the scores to integers and use the heap object from the SPOOLES library [36] with minor modifications.

For all scoring functions implemented, calculating the score of a vertex, say $[v]_{\approx'}$, requires its adjacent set in explicit form from the clique representation stored. For those scoring functions that, in addition, require to calculate $|\operatorname{fill}_G(v)|$, it suffices to update the scores of the vertices in $R'_1 \cup R'_2$, where

$$R'_2 = \operatorname{adj}_{G/\approx'}(R'_1).$$

For the remaining scoring functions, it suffices to calculate the scores of the vertices in R'_1,

which is done as follows:

A pointer to the outmatching vertex is stored with each outmatched one so that this information may be reused in this and later iterations of score updates. The scores of those vertices in R'_1 that are not already known to be outmatched are calculated and inserted into the score heap in the order in which they appear in the array that holds R'_1. While processing the 2-adjacent vertices, we detect further outmatched vertices as described in Paragraph II-E.

For the former type of scoring function, we chose to take advantage of multiple elimination rather than of the fill updating technique of WING and HUANG [23, 39]:

The adjacent set of $[v]_{\approx'}$ as well as the number

$$|\operatorname{adj}_G(v) \cap \operatorname{adj}_G(w)| \tag{16}$$

is maintained for each vertex $[v]_{\approx'}$ of $G/{\approx'}$ and each $[w]_{\approx'}$ adjacent to $[v]_{\approx'}$.

For $v \in R'_1 \cup R'_{2,0}$, that data is discarded and recomputed, where $R'_{2,0}$ is the set of those vertices in R'_2 that are adjacent to vertices newly created by merging indistinguishable vertices, as follows:

The set $\operatorname{adj}_{G/{\approx'}}([v]_{\approx'})$ is recomputed from $(\mathcal{C}_0, \mathcal{C}_1)$ for all $[v]_{\approx'} \in R'_1 \cup R'_{2,0}$. For each edge $\{[v]_{\approx'}, [w]_{\approx'}\}$ with $[v]_{\approx'} \in R'_1 \cup R'_2$, the value of (16) is recomputed by determining the set $\operatorname{adj}_{G/{\approx'}}([v]_{\approx'}) \cap \operatorname{adj}_{G/{\approx'}}([w]_{\approx'})$ if $[v]_{\approx'} \in R'_1$ or $[w]_{\approx'} \in R'_{2,0}$. In all other cases, (16) can be retrieved from the data stored with $[w]_{\approx'} \in R'_{2,1}$. $\operatorname{fill}_G(v)$ is calculated from the values of (16) stored with $[v]_{\approx'}$ for all $[v]_{\approx'} \in R'_1 \cup R'_2$ and the score of $[v]_{\approx'}$ is inserted into the score heap.

An independent set $I \subseteq V/{\approx'}$ of vertices of minimum score is retrieved and deleted from the score heap. If the technique of multiple elimination is applied, then I is maximal. Otherwise, I contains exactly one element.

$(G/{\approx'})_I$ is obtained by eliminating all vertices in I from $G/{\approx'}$. The vertices in R''_1, $R''_1 = \bigcup_{v \in I} \operatorname{adj}_{G/{\approx'}}(v)$, are deleted from the score heap. A clique representation $(\mathcal{C}''_0, \mathcal{C}''_1)$ of $(G/{\approx'})_I$ is obtained by successive modification of $(\mathcal{C}'_0, \mathcal{C}'_1)$, analogous to (9)-(10), for each vertex in I to be eliminated. Pointers of newly created cliques are prepended to the linked lists, so that the first pointer in such a list always refers to the most recently created clique among those in that list.

Since $(G/\approx')_I = G_{\tilde{I}}/\approx''$ for $\tilde{I} = \bigcup_{v\in I} v$ and $\approx'' = \approx' \cap ((V \setminus \tilde{I}) \times (V \setminus \tilde{I}))$, we have obtained the data to which the procedure described above may be recursively applied until all vertices are eliminated.

III. Improvements to the Minimum Degree and Minimum Local Fill scoring functions

Let G be the current elimination graph, $G = (V, E)$, $\approx \subseteq \underset{G}{\sim}$ be the equivalence relation of already detected indistinguishable vertices, and let $(\mathcal{C}_0, \mathcal{C}_1)$ be a clique representation of $G/\approx$, $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1$. For simplicity, we denote the class $[v]_\approx$ by $[v]$.

Let $c_{[v],i} = |\{C \in \mathcal{C}_i \mid [v] \in C\}|$ for $i \in \{0, 1\}$, $c_{[v]} = c_{[v],0} + c_{[v],1}$, and let $\kappa_{[v]} : \{1, \ldots, c_{[v]}\} \to \{C \in \mathcal{C} \mid v \in C\}$ be bijective and such that $\kappa_{[v]}(\{1, \ldots, c_{[v],1}\}) = \{C \in \mathcal{C}_1 \mid v \in C\}$ and that $1 \le i < j \le c_{[v],1}$ implies that the clique $\kappa_{[v]}(j)$ has been created earlier in the elimination process than $\kappa_{[v]}(i)$. In particular, $\kappa_{[v]}(1)$ is the most recently created clique containing $[v]$ if $c_{[v],1} \neq 0$.

We define scoring functions in terms of $G/\approx$. To simplify the notation, these functions take a single argument, the vertex of $G/\approx$, though they clearly further depend on G, $\approx$, $\mathcal{C}$, and κ. For $W \subseteq V/\approx$, we define

$$\|W\| = \sum_{[w]\in W} |[w]|.$$

A. Bounds on the local fill

The scoring function of the MD algorithm represents an upper bound on the number of fill-ins introduced by eliminating a vertex from G, since

$$\vartheta : x \mapsto (x^2 - x)/2$$

is monotonic on the set of nonnegative integers and $|\operatorname{fill}_G(v)| \le \vartheta(\deg_G(v))$ for all $v \in V$.

That bound may be improved since some of the $\vartheta(\deg_G(v))$ potential fill-in edges in G_v are edges of G that are easy to identify:

First, if v is indistinguishable from w in G, $v \neq w$, then eliminating v from G does not create any new edges adjacent to w in G_v. Hence, the *external degree* $\deg_G([v])$ *of* v

represents an upper bound on $|\operatorname{fill}_G(v)|$,

$$|\operatorname{fill}_G(v)| \leq \vartheta(\deg_G([v])). \tag{17}$$

Taking the external degree as scoring function usually produces less overall fill than taking the degree, a fact first observed by Liu [35].

Further, if $C \in \mathcal{C}$ is a clique, then the elimination of $[v]$ from G cannot create any new edge $\{[u],[w]\} \subseteq C$. Even if only those cliques are considered that contain $[v]$, the bound given in (17) may be improved in several ways:

We define *partial cliques* $\tilde{\kappa}_{[v]}(i)$ by

$$\tilde{\kappa}_{[v]}(i) = (\kappa_{[v]}(i) \setminus \{[v]\}) \setminus \bigcup_{1 \leq j < i} \kappa_{[v]}(i)$$

for all i, $1 \leq i \leq c_{[v]}$ [17].

The *Approximate Minimum Local Fill (AMF)* algorithm of Rothberg and Eisenstat uses the upper bound s_{AMF_0},

$$s_{AMF_0}([v]) = \vartheta(\deg_G([v])) - \begin{cases} 0 & \text{if } c_{[v],1} = 0 \\ \vartheta(\|\tilde{\kappa}_{[v]}(1)\|) & \text{otherwise} \end{cases}$$

as its scoring function, thereby taking into account the most recently eliminated clique only [22]. Using the largest clique instead [22] leads to the upper bound s_{AMF_1},

$$s_{AMF_1}([v]) = \vartheta(\deg_G([v])) - \begin{cases} 0 & \text{if } c_{[v],1} = 0 \\ \max\limits_{1 \leq i \leq c_{[v],1}} \vartheta(\|\kappa_{[v]}(i) \setminus \{[v]\}\|) & \text{otherwise} \end{cases}.$$

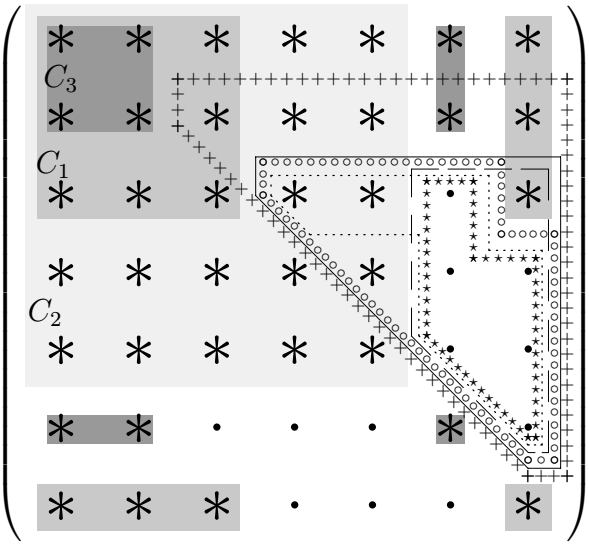
Ng and Raghavan propose the bound s_{AMF_2},

$$s_{AMF_2}([v]) = \frac{1}{2}\left(\deg_G([v])^2 - \sum_{i=1}^{c_{[v]}} \|\tilde{\kappa}_{[v]}(i)\|^2\right),$$

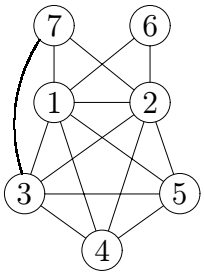
extended by some correction term to be defined in Paragraph III-C, as scoring function of their *Modified Minimum Deficiency (MMDF)* algorithm [17].

A straightforward further improvement not suggested in the literature is the bound s_{AMF_3},

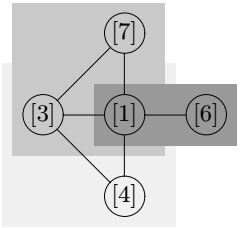
$$s_{AMF_3}([v]) = \frac{1}{2}\left(\deg_G([v])^2 - \sum_{i=1}^{c_{[v]}} \|\tilde{\kappa}_{[v]}(i)\| \left(2\|\kappa_{[v]}(i) \setminus \{[v]\}\| - \|\tilde{\kappa}_{[v]}(i)\|\right)\right),$$



(a)



(b)



(c)

Fig. 3. Illustration of Example 2. (a) Zero-nonzero pattern of a 7×7 matrix A. (Nonzeros are denoted by asterisks ($*$), zeros by dots ($\bullet$).) The regions of potential fill-ins in the upper triangle corresponding to the bounds on $|\operatorname{fill}_G(1)|$ from Tab. I are surrounded by curves of the following styles: $\deg_G(1)$ $+++$, $\deg_G([1])$ —, $s_{AMF_0}([1])$ $\circ\circ\circ$, $s_{AMF_1}([1])$ $- - -$, $s_{AMF_2}([1])$ $\cdots$, $s_{AMF_3}([1])$ $\star\star\star$. (b) The graph $\mathcal{G}(A)$ of A. (c) The graph $\mathcal{G}(A)/\approx$. (In (a) and (c), the shaded regions correspond to the cliques C_1, C_2, and C_3 as indicated.)

which can easily be justified.

Example 2 Consider the zero-nonzero pattern shown in Fig. 3(a) of some 7×7 matrix A. The graph $\mathcal{G}(A)$ of A is shown in Fig. 3(b).

Assume that the classes $\{1, 2\}$ and $\{4, 5\}$ of indistinguishable vertices have been detected so that the set of vertices of the graph $G/\approx$ shown in Fig. 3(c) is $\{[1], [3], [4], [6], [7]\}$ and the set of its edges is $\{\{[1], [3]\}, \{[1], [4]\}, \{[1], [6]\}, \{[1], [7]\}, \{[3], [4]\}, \{[3], [7]\}\}$.

Assume further that the clique representation $(\mathcal{C}_0, \mathcal{C}_1)$ has been obtained from previous

Scoring function	Score([1])	Bound on $\lvert \operatorname{fill}_G(1) \rvert$
degree	6	15
external degree	5	10
s_{AMF_0}	9	9
s_{AMF_1}	7	7
s_{AMF_2}	8	8
s_{AMF_3}	6	6

TABLE I

Scores and bounds on $\lvert \operatorname{fill}_G(1) \rvert$ as in Example 2.

eliminations, where $\mathcal{C}_0 = \{C_3\}$, $\mathcal{C}_0 = \{C_1, C_2\}$,

$$C_1 = \{[1], [3], [7]\},$$
$$C_2 = \{[1], [3], [4]\},$$
$$C_2 = \{[1], [6]\},$$

see Fig. 3(a)-(c), and that the clique C_1 has been created after C_2. In other words, $\tilde{\kappa}_{[1]}(1) = \{[3], [7]\}$, $\tilde{\kappa}_{[1]}(1) = \{[4]\}$, and $\tilde{\kappa}_{[1]}(1) = \{[6]\}$,

Under these assumptions we obtain the scores given in Tab. I for the vertex [1]. The sets of potential fill-ins in the upper triangle of A corresponding to the bounds on $\lvert \operatorname{fill}_G(1) \rvert$ from Tab. I are illustrated in Fig. 3(a).

Note that $\lvert \operatorname{fill}_G(1) \rvert = 6$, i.e., in this example, the upper bound yielded by s_{AMF_3} is tight. □

B. Looking ahead

While local ordering algorithms usually consider the fill introduced in the next elimination step only, the concept of indistinguishability provides a simple means to look some steps ahead:

Recall that $\operatorname{fill}_{G_v}(w) = \emptyset$ for any $w \in [v] \setminus \{v\}$, and that the total number of fill-ins created when all vertices in $[v]$ are eliminated from G immediately upon each other is just $\lvert \operatorname{fill}_G(v) \rvert$. Consequently, ROTHBERG and EISENSTAT consider dividing fill bounds by $\lvert [v] \rvert$

[22]. Their *Minimum Mean Local Fill (MMF)* and *Approximate Minimum Mean Local Fill (AMMF)* heuristics are based on the scoring functions s_{MMF^α} and s_{AMMF},

$$s_{MMF^\alpha}([v]) = |\operatorname{fill}_G(v)|/|[v]|^\alpha,$$
$$s_{AMMF}([v]) = s_{AMF_0}([v])/|[v]|^\alpha$$

with $\alpha = 1$. According to [22], a version with $\alpha = 1/2$ "produced slightly better results".

We combine that trick with other fill bounds and use the scoring functions $s_{AMMF_i^\alpha}$,

$$s_{AMMF_i^\alpha}([v]) = s_{AMF_i}([v])/|[v]|^\alpha$$

for $i \in \{0, 1, 2, 3\}$ and $\alpha \in \{1/2, 2/3, 1\}$.

C. Correction terms

Rothberg and Eisenstat suggest considering not only edges created by eliminating a vertex, but also those that are removed. Their *Minimum Increase In Neighborhood Degree (MIND)* and *Approximate Minimum Increase In Neighborhood Degree (AMIND)* heuristics use the scoring functions s_{MIND} and s_{AMIND},

$$s_{MIND}([v]) = |\operatorname{fill}_G(v)| - \deg_G([v])|[v]|,$$
$$s_{AMIND}([v]) = s_{AMF_0}([v]) - \deg_G([v])|[v]|,$$

the last term of which is interpreted as the number of edges removed [22].

The MMDF algorithm of Ng and Raghavan is determined by its scoring function

$$s_{MMDF}([v]) = s_{AMF_2}([v]) - \deg_G([v])|[v]|. \tag{18}$$

Contrary to the explanation in [22], the last term in (18) is referred to as a correction term, which is to improve the approximation of $|\operatorname{fill}_G(v)|$ by the scoring function [17].

D. Other approaches

Among the various improvements of the MD and MF algorithms that we do not discuss in detail in this paper are the following:

Amestoy, Davis and Duff use a scoring function that is an approximation to the external degree [38]. The computational cost of their *Approximate Minimum Degree (AMD)*

algorithm is less than that of the MD algorithm, though the orderings obtained create about the same amount of fill. That idea may be combined with other scoring functions as well [17, 22].

Cavers [19] and Meszaros [21] show that breaking ties in the MD algorithm on the basis of local fill-in counts significantly reduces the number of fill-ins.

The *Modified Multiple Minimum Degree (MMMD)* algorithm of Ng and Raghavan [17] uses the scoring function s_{MMMD}, $s_{MMMD}([v]) = 2\deg_G([v]) - \max_{1\leq i\leq c_{[v]}} \|\tilde{\kappa}_{[v]}(i)\|$.

IV. Computational results

In addition to the MD and MF scores, we have implemented the scoring functions defined in Section III on the basis of the generic algorithm of Paragraph II-G.

The code was compiled with the cc compiler (Workshop Compilers 4.2 30 Oct 1996 C 4.2) using the options "-fast -fsimple=2 -xtarget=ultra -xarch=v8ultra" under SunOS Release 5.7 and run on one of the CPUs (sparcv9+vis, 400 MHz clock rate, 4 MB cache, 17.4 SPECint95, 25.7 SPECfp95) of a SUN Enterprise E4500 workstation with 6 Gbytes of memory. Of course, other jobs were running on the same machine in parallel to ours, which may have affected the CPU times reported in this section to some extent.

Our test suite of input data contains 15 matrices extracted from the circuit simulator TITAN [2], *circuit matrices* in the following, and 18 matrices from other fields of application [40, 41].

In each of the following two paragraphs, we chose a reference algorithm. For all other algorithms, we report ratios, i.e., quantities calculated for these algorithms divided by the corresponding quantities for the reference algorithm. We judge algorithms by comparing the geometric mean of the ratios calculated separately for the set of circuit matrices and the other matrices. For example, by "the number of factorization operations for method M is x% (y%) less than for method M' " we will express that the geometric means over the ratios of the operation counts measured for method M over the set of circuit matrices and the set of the other matrices, divided by the corresponding quantity for method M', is $1 - x/100$ and $1 - y/100$, respectively. (We round the values of x and y to two decimal digits.)

Our primary measure for comparing ordering algorithms is the number

$$\sum_{i=1}^{n} c_i(1 + r_i) \tag{19}$$

of factorization operations determined by the pivoting orders obtained, which represents divisions and multiplications in (3)-(5), where c_i and r_i are the number of off-diagonal nonzeros in column and row i of the factors L and U, respectively, of PAP^T, and P is the permutation matrix corresponding to the pivoting order.

A. Comparison of symmetric ordering methods

For the circuit matrices, some initial steps of Markowitz' algorithm were applied to remove rows and columns containing diagonal elements with zero Markowitz product, and the remaining submatrices were symmetrized as described in Section I. The resulting symmetric matrices are listed in Tab. II under the names "bag" through "X".

The matrices "bcsstk15" through "crystk03", also listed in Tab. II, had already been symmetric.

It is well known that ordering heuristics are sensitive to permutations of the rows and columns of the input matrices. Therefore, the arithmetic means over the quantities measured for 11 runs with different random orders for inserting the initial scores into the score heap, as described in Paragraph II-G, are the basis for subsequent comparisons.

We chose the *Multiple Minimum Degree (MMD)* algorithm [31], a version of the MD algorithm based on external degrees, which uses multiple elimination and other techniques described in Section II and is probably the most widely used local symmetric ordering method today, as our reference algorithm. Tab. II contains the performance of that algorithm on our test matrices.

Except for the MMD and MF algorithms, all algorithms considered here are named by their scoring functions.

All the algorithms tested apply the techniques explained in Section II, except for the MMDF algorithm, which "does not allow 'multiple eliminations.' " [17]. However, in tests which we do not report here in detail, we found that incorporating multiple elimination into the MMDF heuristic reduces the number of factor operations by 0% (3.6%), the final matrix size including fill by 0% (2.1%), and the running time by 48% (3%).

Problem			MMD		
Name	n	m	o	m'	t/sec.
bag	143854	922936	$3.3 \cdot 10^7$	$2.0 \cdot 10^6$	7.42
cor	123118	659182	$1.7 \cdot 10^6$	$7.8 \cdot 10^5$	5.57
eng	4893	33993	$1.0 \cdot 10^6$	$8.3 \cdot 10^4$	0.16
jac	903	17967	$1.1 \cdot 10^6$	$3.6 \cdot 10^4$	0.05
m8	10464	197374	$3.0 \cdot 10^7$	$7.3 \cdot 10^5$	1.27
m14	15766	348968	$5.6 \cdot 10^7$	$1.3 \cdot 10^6$	2.44
m24	26120	613812	$3.9 \cdot 10^8$	$3.7 \cdot 10^6$	5.01
m40	156035	1789325	$3.4 \cdot 10^9$	$1.9 \cdot 10^7$	25.99
sei	3539	280785	$1.5 \cdot 10^8$	$7.6 \cdot 10^5$	0.51
buc	9751	53427	$8.7 \cdot 10^5$	$1.1 \cdot 10^5$	0.21
gue	86086	390074	$1.4 \cdot 10^7$	$7.9 \cdot 10^5$	2.27
te	59418	253584	$6.9 \cdot 10^5$	$3.4 \cdot 10^5$	1.33
tei	174702	804620	$4.6 \cdot 10^6$	$1.2 \cdot 10^6$	6.38
xch	10560	55852	$8.8 \cdot 10^5$	$1.2 \cdot 10^5$	0.23
X	16063	127887	$3.1 \cdot 10^6$	$2.4 \cdot 10^5$	0.46
bcsstk15	3948	117816	$1.7 \cdot 10^8$	$1.3 \cdot 10^6$	0.26
bcsstk16	4884	290378	$1.5 \cdot 10^8$	$1.5 \cdot 10^6$	0.06
bcsstk17	10974	428650	$1.9 \cdot 10^8$	$2.2 \cdot 10^6$	0.18
bcsstk18	11948	149090	$1.3 \cdot 10^8$	$1.3 \cdot 10^6$	0.55
bcsstk23	3134	45178	$1.4 \cdot 10^8$	$9.1 \cdot 10^5$	0.24
bcsstk25	15439	252241	$3.3 \cdot 10^8$	$3.0 \cdot 10^6$	0.77
bcsstk29	13992	619488	$4.5 \cdot 10^8$	$3.5 \cdot 10^6$	0.42
bcsstk30	28924	2043492	$8.8 \cdot 10^8$	$7.5 \cdot 10^6$	0.36
bcsstk31	35588	1181416	$2.5 \cdot 10^9$	$1.0 \cdot 10^7$	0.91
bcsstk32	44609	2014701	$1.1 \cdot 10^9$	$1.0 \cdot 10^7$	0.60
bcsstk33	8738	591904	$1.3 \cdot 10^9$	$5.2 \cdot 10^6$	0.24
bcsstk35	30237	1450163	$3.9 \cdot 10^8$	$5.5 \cdot 10^6$	0.20
bcsstk36	23052	1143140	$6.1 \cdot 10^8$	$5.5 \cdot 10^6$	0.12
bcsstk37	25503	1140977	$5.5 \cdot 10^8$	$5.6 \cdot 10^6$	0.22
bcsstk38	8032	355460	$1.2 \cdot 10^8$	$1.5 \cdot 10^6$	0.16
crystk01	4875	315891	$3.5 \cdot 10^8$	$2.2 \cdot 10^6$	0.07
crystk02	13965	968583	$4.4 \cdot 10^9$	$1.2 \cdot 10^7$	0.24
crystk03	24696	1751178	$1.3 \cdot 10^{10}$	$2.8 \cdot 10^7$	0.47

TABLE II

Test problems and performance of the MMD algorithm. n and m are the number of rows and nonzeros, respectively, in the matrix. o, m' and t denote the number (19) of factorization operations, the number of nonzeros including the fill, and the CPU time of the MMD algorithm. The values of o and m' are rounded to two decimal digits, those of t to a precision of 10^{-2}sec..

	circuit matrices			other matrices		
Algorithm	o	m'	t	o	m'	t
AMF_0	0.95	0.98	1.3	0.86	0.93	1.3
AMMF_0^1	0.91	0.97	1.5	0.78	0.91	1.9
AMIND	0.92	0.98	1.3	0.77	0.90	1.4
MMDF	0.93	0.98	2.6	0.83	0.93	1.6
MF	0.76	0.92	13	0.74	0.87	13
MMF^1	0.74	0.91	15	0.68	0.85	17
MIND	0.79	0.95	15	0.68	0.85	14
$\mathrm{AMMF}_1^{1/2}$	0.85	0.95	1.4	0.75	0.88	1.4
$\mathrm{AMMF}_3^{1/2}$	0.83	0.95	1.5	0.77	0.90	1.8
$\mathrm{MMF}^{1/2}$	0.69	0.90	14	0.66	0.83	14

TABLE III

Performance of symmetric ordering methods relative to the MMD algorithm. o, m' and t denote the number (19) of factorization operations, the number of nonzeros including the fill, and the CPU time of the ordering algorithm. The reported values are geometric means over the quantities for the circuit matrices and the other matrices, respectively, divided by those for the MMD algorithm and rounded to two decimal digits.

We have also verified that "the performance of MMMD and ... AMF ... are similar" [17] and do not further consider the former.

Computational results for some of the algorithms discussed earlier in this paper are presented in Tab. III.

We first observe that the improvements of the local fill bound beyond the one represented by the external degree always lead to better pivoting orders. However, tighter bounds do not necessarily correspond to better pivoting orders, as we can see by comparing, for example, the results for the AMIND and MMDF algorithms.

It has been reported that "tightening the fill bound beyond the information available from the most recently formed clique did not improve ordering quality" [22]. In further tests which we do not report here in detail, we found the following:

Taking into account the largest rather than the most recently created clique slightly improves the orders obtained. In particular, the AMF_1 and AMMF_1^α algorithms lead to $1.1\% - 2.3\%$ $(0 - 1.3\%)$ fewer factorization operations than AMF_0 and AMMF_0^α for $\alpha \in \{1/2, 2/3, 1\}$. However, the AMF_2 and AMMF_2^α algorithms lead to $0 - 2.2\%$ $(2.7\% - 5.3\%)$ more factorization operations than AMF_0 and AMMF_0^α. Further, the AMF_3 and AMMF_3^α algorithms lead to $1.1\% - 3.4\%$ fewer factorization operations than AMF_1 and AMMF_1^α for the circuit matrices, and to $2.7\% - 6.4\%$ more for the other matrices.

From the data of Tab. III, we see that those algorithms that rely on the exact local fill yield significantly better pivoting orders than the others. Further, the introduction of a correction term as described in Paragraph III-C is considerably less useful for the circuit matrices than for the others. In particular, for the circuit matrices, the MIND algorithm leads to 3.9% more factorization operations than the MF algorithm.

Dividing bounds on the local fill by a power of the cardinality of the class of indistinguishable vertices in question, as explained in Paragraph III-B, turns out to be very useful. In tests which we do not report here in detail, we found the following:

The AMMF_i^1 and MMF^1 algorithms lead to $2.6\% - 4.2\%$ $(7.8\% - 9.3\%)$ fewer factorization operations than AMF_i and MF, $\mathrm{AMMF}_i^{2/3}$ and $\mathrm{MMF}^{2/3}$ to $2.2\% - 6.8\%$ $(2.9\% - 7.3\%)$ fewer than AMMF_i^1 and MMF^1, and $\mathrm{AMMF}_i^{1/2}$ and $\mathrm{MMF}^{1/2}$ to $4.4\% - 6.8\%$ $(2.6\% - 7.2\%)$ fewer than AMMF_i^1 and MMF^1 for $i \in \{0, 1, 2, 3\}$. The exponent $1/2$ was better than $2/3$ for the circuit matrices and worse for the others.

The running times for those heuristics that are based on bounds on the local fill appear to be roughly the same, and those heuristics that are based on exact local fill counts are more than one order of magnitude slower.

After all, for the circuit matrices, the $\mathrm{MMF}^{1/2}$ algorithm leads to the fewest number of factorization operations, which is 17% and 19% less than that for the $\mathrm{AMMF}_3^{1/2}$ and the $\mathrm{AMMF}_1^{1/2}$ algorithms, respectively, the best among the algorithms based on bounds. A final decision on which of those heuristics is best does not only depend on the field of application, but also on the computer architecture and the specific numerical factorization algorithm used for the simulations [20] and is beyond the scope of this paper.

The data presented unveil differences between the circuit and the other matrices. First,

the running times on the circuit matrices are significantly higher than on the others, even if we look at examples of roughly the same size.

Further, heuristics based on local fill bounds, especially those taking into account one clique only, are less useful for the circuit matrices than for the others.

Also, the introduction of a correction term is not always advantageous for circuit matrices.

We think that those differences are, at least in part, due to the higher number of cliques generated during the ordering process for the circuit matrices: In tests which we do not report here in detail, we found that the maximum number of cliques created that contain a specific vertex was between 57 and 8194 for the circuit matrices, whereas that number never exceeded 25 for the others. We conjecture that the application of the technique of element absorption to a larger set of cliques than described in Paragraph II-C would decrease running times significantly. (See [38] for "aggressive" absorption.)

B. Combination of Markowitz' algorithm with local symmetric ordering methods

The previous paragraph includes a comparison of symmetric ordering methods on a test set of symmetric matrices, which have resulted from the application of initial steps of Markowitz' algorithm to unsymmetric circuit matrices, followed by a symmetrization, as described in Section I.

In this paragraph, we show that the pivoting orders obtained for the unsymmetric circuit matrices themselves by the above combination of Markowitz' algorithm with symmetric methods are significantly better than those obtained from Markowitz' algorithm alone, in some cases at virtually no extra computational cost.

We chose the implementation of Markowitz' algorithm of the circuit simulator TITAN [2], version 6.1a, as our reference algorithm.

For the combinations of Markowitz' algorithm with symmetric ordering methods to the problems "bag" through "sei" listed in Tab. IV, we report the arithmetic mean of the number of factorization operations over 11 runs as in the previous paragraph. For Markowitz' algorithm, and for the problems buc through X of Tab. IV, we report the result of one run only.

From the problem data presented in Tab. II and Tab. IV, we see that the initial steps

Problem						Markowitz' A.	MMD	$\text{AMMF}_1^{1/2}$	$\text{MMF}^{1/2}$
			$\frac{m_1}{m_0}$	$\frac{u_0}{m_0}$	$\frac{u_1}{m_1}$				
Name	n_0	m_0	in %	in %	in %	o	o	o	o
bag	143956	995495	93	7.3	0	$3.5 \cdot 10^7$	0.95	0.81	0.68
cor	123159	891955	74	26	0	$1.8 \cdot 10^6$	0.87	0.87	0.87
eng	4951	42410	80	20	0	$1.1 \cdot 10^6$	0.96	0.97	0.88
jac	935	21769	83	17	0	$1.0 \cdot 10^6$	1.12	1.28	0.92
m8	10534	236683	83	17	0	$3.9 \cdot 10^7$	0.77	0.56	0.43
m14	15944	417551	84	16	0	$7.1 \cdot 10^7$	0.79	0.64	0.55
m24	26202	709681	86	13	0	$4.1 \cdot 10^8$	0.94	0.72	0.49
m40	156115	2039389	88	12	0	$4.2 \cdot 10^9$	0.81	0.61	0.48
sei	3573	291742	96	3.7	0	$2.5 \cdot 10^8$	0.61	0.34	0.28
buc	10049	78116	58	51	17	$8.4 \cdot 10^5$	0.92	0.86	0.69
gue	88714	549047	69	32	2.3	$1.6 \cdot 10^7$	0.82	0.88	0.43
te	59716	398494	64	36	0.032	$7.6 \cdot 10^5$	1.00	1.01	0.99
tei	174880	1205722	67	33	0.0099	$4.7 \cdot 10^6$	1.08	1.04	0.86
xch	10948	83644	57	52	17	$8.3 \cdot 10^5$	1.02	0.82	0.73
X	16492	170454	67	40	11	$2.8 \cdot 10^6$	0.86	0.76	0.63
total							0.89	0.78	0.62

TABLE IV

Circuit matrices and performance of Markowitz' algorithm and its combination with the MMD, $\text{AMMF}_1^{1/2}$ and $\text{MMF}^{1/2}$ algorithms. n_0, m_0, and u_0 denote the number of rows, nonzeros, and structurally unsymmetric nonzeros, respectively. m_1 and u_1 denote the number of nonzeros and structurally unsymmetric nonzeros, respectively, in the rest matrices remaining after the initial steps of Markowitz' algorithm. o denotes the number of factorization operations (19), for the combinations divided by the corresponding value for Markowitz' algorithm. The values in the row "total" are geometric means of the ratios reported in the respective column.The values of n_0 and m_0 are exact, all ratios are rounded to a precision of 10^{-2}, and the remaining quantities are rounded to two decimal digits.

of Markowitz' algorithm only slightly reduce the dimension of the problem, but remove $4\% - 43\%$ of the nonzeros. Moreover, while $3.7\% - 52\%$ of the nonzeros of the original matrices are structurally unsymmetric, those initial steps remove most or all of them. In particular, the rest matrices for the problems "bag" through "sei" are symmetric, and only about $0.01\% - 17\%$ of the nonzeros of the other rest matrices are structurally unsymmetric.

For problems leading to symmetric rest matrices, it is obvious that the advantages of the symmetric ordering methods investigated over the basic MD heuristic carry over to the combination of those methods with Markowitz' algorithm. The results presented in Tab. IV show that these advantages carry over to the combination even if the rest matrices are structurally unsymmetric.

After all, the combination of Markowitz' algorithm with the MMD, $\text{AMMF}_1^{1/2}$, and $\text{MMF}^{1/2}$ algorithms leads to 11%, 22%, and 38% fewer factorization operations than Markowitz' algorithm alone. Furthermore, it is evident that the running time of the above combination with the MMD algorithm should never exceed that of an analogous but unsymmetric implementation of Markowitz' algorithm. In fact, the code of Markowitz' algorithm we used was always much slower than its combination with both the MMD and the $\text{AMMF}_1^{1/2}$ heuristic.

As noted earlier, a final decision on which of those heuristics is best would not only depend on the kind of circuits to be simulated, but also on the computer architecture and the specific numerical factorization algorithm used [20] and is beyond the scope of this paper. However, let us show by an example that the savings in factorization operations of the $\text{MMF}^{1/2}$ over the $\text{AMMF}_1^{1/2}$ heuristic may very well reduce the overall simulation time:

The largest CPU time for the $\text{MMF}^{1/2}$ algorithm was 2h43min, achieved for the "m40" example for which application of the $\text{AMMF}_1^{1/2}$ algorithm took only 47sec.. However, a transient simulation of that example with TITAN [2], version 6.1*a*, using the pivoting order obtained from Markowitz' algorithm, spent over 138h on factorizations, so that the savings of the $\text{MMF}^{1/2}$ algorithm in factorization operations of 21% relative to the $\text{AMMF}_1^{1/2}$ heuristic should outweigh the larger CPU time of the former.

V. Conclusions

We have reviewed recently proposed local symmetric ordering methods, i.e., methods for obtaining pivoting orders for sparse symmetric matrices, as well as some straightforward improvements to them. Our comparison of the performance of these methods with that of the Minimum Degree and Minimum Local Fill algorithms, mainly in terms of the resulting number of factorization operations, shows that pivoting orders significantly better than those yielded by the Minimum Degree and Minimum Local Fill algorithms can be obtained, at virtually no extra computational cost.

Furthermore, we have shown that for the purpose of circuit simulation, a combination of Markowitz' algorithm with symmetric ordering methods yields pivoting orders significantly better than those obtained from Markowitz' algorithm alone, again, in some cases at virtually no extra computational cost.

We could not make a final decision on what ordering algorithm is best. That decision would not only depend on the kind of circuits to be simulated, but also on the computer architecture and the specific numerical factorization algorithm used and is beyond the scope of this paper.

However, we think that a further improvement of the running times of those symmetric methods that are based on exact local fill counts, for example, through a combination of multiple elimination with the fill updating method of Wing and Huang [23, 39], would make them superior to any other local ordering method known today when combined with Markowitz' algorithm as described in [24] and applied in this paper.

Acknowledgment

I thank C. Ashcraft (Livermore, WA), G. Denk (München), F. Grund (Berlin), T. Klimpel (München), P. Raghavan (Knoxville, TN), and E. Rothberg (Mountain View, CA) for their kind support, valuable hints and comments.

References

[1] L. O. Chua, C. A. Desoer, and E. S. Kuh, *Linear and Nonlinear Circuits.* McGraw–Hill, 1987.

[2] U. Feldmann, U. A. Wever, Q. Zheng, R. Schultz, and H. Wriedt, "Algorithms for modern circuit simulation," *Archiv für Elektronik und Übertragungstechnik (AEÜ)*, vol. 46, no. 4, pp. 274–285, 1992.

[3] L. W. Nagel, "SPICE 2: A computer program to simulate semiconductor circuits," Tech. Rep. ERL-M520, Univ. of Calif. Berkeley, Electronic Res. Lab., Berkeley, CA, 1975.

[4] I. N. Hajj, P. Yang, and T. N. Trick, "Avoiding zero pivots in the modified nodal approach," *IEEE Trans. Circuits and Systems*, vol. 28, no. 4, pp. 271–278, 1981.

[5] G.-L. Tan, "An algorithms for avoiding zero pivots in the modified nodal approach," *IEEE Trans. Circuits and Systems*, vol. 33, no. 4, pp. 431–434, 1986.

[6] L. O. Chua and P.-M. Lin, *Computer–Aided Analysis of Electronic Circuits.* Englewood Cliffs, NJ: Prentice–Hall, 1975.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations.* The John Hopkins Univ. Press, 2 ed., 1993.

[8] J. M. Ortega, "The ijk forms of factorization methods. I. Vector computers," *Parallel Comput.*, vol. 7, no. 2, pp. 135–147, 1988.

[9] A. Chang, "Application of sparse matrix methods in electric power system analysis," in *Proc. Symp. Sparse Matrices and Their Applications, IBM Watson Res. Center, Sept. 9-10, 1968* (R. A. Willoughby, ed.), no. RA 1 (# 11707) in Numerical Analysis, pp. 113–121, 12. Mar. 1969.

[10] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices.* Oxford University Press, 1986.

[11] D. J. Rose and R. E. Tarjan, "Algorithmic aspects of vertex elimination on directed graphs," *SIAM J. Appl. Math.*, vol. 34, no. 1, pp. 176–197, 1978.

[12] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM J. Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[13] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Sci.*, vol. 3, pp. 255–269, 1957.

[14] W. F. Tinney and J. W. Walker, "Direct solution of sparse network equations by optimally ordered triangular factorization," *Proc. IEEE*, vol. 55, pp. 1801–1809, 1967.

[15] S. V. Parter, "The use of linear graphs in Gauss elimination," *SIAM Rev.*, vol. 3, pp. 119–130, 1961.

[16] D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," in *Graph Theory and Computing* (R. C. Read, ed.), pp. 183–217, Academic Press, 1972.

[17] E. G. Ng and P. Raghavan, "Performance of greedy ordering heuristics for sparse Cholesky factorization," *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 4, pp. 902–914, 1999.

[18] I. S. Duff, "A survey of sparse matrix research," *Proc. IEEE*, vol. 65, no. 4, pp. 500–535, 1977.

[19] I. A. Cavers, "Using deficiency measure for tiebreaking the minimum degree algorithm," Tech. Rep. 89-2, Dept. Comp. Sci., The Univ. Of British Columbia, Vancouver, British Columbia, Canada V6T 1W5, 10. Jan. 1989.

[20] I. J. Lustig, R. E. Marsten, and D. F. Shanno, "The interaction of algorithms and architectures for interior point methods.," in *Advances in optimization and parallel computing* (P. M. Pardalos, ed.), pp. 190–204, North-Holland, 1992.

[21] C. Mészáros, "Ordering heuristics in interior point LP methods," in *New trends in mathematical programming* (F. Giannessi, S. Komlósi, and T. Rapcśak, eds.), pp. 203–221, Boston, MA, U.S.A.: Kluwer Acad. Publ., 1998.

[22] E. Rothberg and S. C. Eisenstat, "Node selection strategies for bottom-up sparse matrix ordering," *SIAM J. Matrix Anal. Appl.*, vol. 19, pp. 682–695, July 1998.

[23] J. Vlach and K. Singhal, *Computer methods for circuit analysis and design.* Van Nostrand Rheinhold Company, 1983.

[24] G. Reißig and T. Klimpel, "Fill-In Minimierung in der Schaltkreissimulation." Erfindungsmeldung, Infineon Technologies, MP PTS, München, 28 Mar. 2000.

[25] D. E. Knuth, *The Art of Computer Programming*, vol. 3, Sorting and Searching. Addison Wesley, 1973.

[26] F. Harary, *Graph theory.* Addison-Wesley Publishing Co., Reading, Mass.-Menlo Park, Calif.-London, 1969.

[27] A. George, "Nested dissection of a regular finite element mesh," *SIAM J. Numer. Anal.*, vol. 10, pp. 345–363, 1973.

[28] B. Speelpenning, "The generalized element method. Preliminary report." Notices Amer. Math. Soc., vol. 20, no. 2, p. A-280, 73T-C18, Feb. 1973.

[29] B. Speelpenning, "The generalized element method," Tech. Rep. UIUCDCS-R-78-946, Univ. Illinois at Urbana-Champaign, Dept. Comp. Sci., Urbana, IL, Nov. 1978. Reprint of 1973.

[30] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear equations," *ACM Trans. Math. Software*, vol. 9, no. 3, pp. 302–325, 1983.

[31] A. George and J. W. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Rev.*, vol. 31, pp. 1–19, Mar. 1989.

[32] J. A. George and J. W. Liu, *Computer solution of large sparse positive definite systems.* Prentice-Hall, 1981.

[33] A. George and D. R. McIntyre, "On the application of the minimum degree algorithm to finite element systems," *SIAM J. Numer. Anal.*, vol. 15, no. 1, pp. 90–112, 1978.

[34] A. H. Sherman, "Yale sparse matrix package - User's guide," Tech. Rep. UCID-30114, Lawrence Livermore Lab., Univ. of California, Livermore, CA, Aug. 1975.

[35] J. W. H. Liu, "Modification of the minimum-degree algorithm by multiple elimination," *ACM Trans. Math. Software*, vol. 11, no. 2, pp. 141–153, 1985.

[36] "SPOOLES — An object oriented software library for solving sparse linear systems of equations." `netlib.bell-labs.com/netlib/linalg/spooles/`, 1999.

[37] C. Ashcraft, "Compressed graphs and the minimum degree algorithm," *SIAM J. Sci. Comput.*, vol. 16, no. 6, pp. 1404–1411, 1995.

[38] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM J. Matrix Anal. Appl.*, vol. 17, no. 4, pp. 886–905, 1996.

[39] O. Wing and J. Huang, "SCAP - a sparse matrix circuit analysis program," in *Proc. 1975 IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 213–215, 1975.

[40] I. S. Duff, R. G. Grimes, and J. G. Lewis, "Users' guide for the Harwell–Boeing sparse matrix collection (Release I)," Tech. Rep. TR/PA/92/86, Rutherford Appleton Lab., Oxon, England, Boeing Computer Services, Res. and Techn. Div., Seattle, WA, U.S.A, Oct. 1992.

[41] R. G. Grimes, "Sparse matrices." http://www.cise.ufl.edu/~davis/sparse/Boeing/, 2. June 1995.