A NOTE ON THE LDL^T DECOMPOSITION OF MATRICES FROM SADDLE-POINT PROBLEMS

MIROSLAV TŮMA *

Abstract. Sparse linear systems $Kx = b$ are considered where K is a specially structured symmetric indefinite matrix. These systems arise frequently, e.g., from mixed finite element discretizations of PDE problems. The LDL^T factorization of K with diagonal D and unit lower triangular L is known to exist for natural ordering of K but the resulting triangular factors can be rather dense. On the other hand, for a given permutation matrix P, the LDL^T factorization of P^TKP may not exist.

In this paper a new way to obtain a fill-in minimizing permutation based on initial fill-in minimizing ordering is introduced. For an important subclass of matrices arising from mixed and hybrid finite element discretizations the existence of the LDL^T factorization of the permuted matrix is proved. Experimental results on practical problems indicate that the amount of computational savings can be substantial when compared with the approach based on Schur complement.

Key words. Systems of sparse linear algebraic equations, symmetric indefinite systems, direct methods, LDL^T decomposition, supernodal solvers.

AMS subject classifications. 65F10, 65F35, 65F50, 65Y05 .

1. Introduction.

We consider the solution of symmetric linear systems of the form

$$Kx = b \tag{1.1}$$

where the coefficient matrix $K \in \mathbb{R}^{(n+m)\times(n+m)}, n \geq m$ has the following block form

$$K = \begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix}, \tag{1.2}$$

where matrices $H \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{m\times n}$ are large and sparse and H is symmetric and positive definite. Clearly, K is indefinite. We always assume in this paper that K is irreducible (being a technical assumption only) and also nonsingular, i.e. A^T has full column rank. Such matrices will be called later in this paper *saddle-point* matrices.

If a sparse symmetric matrix K is factorized into LDL^T, it is likely that *fill-in* occurs, i.e. the number of nonzero entries in the factors is usually greater than the number of nonzeros in K. It is typically cheaper to factorize a permuted matrix $\bar{K} = PKP^T$ into $\bar{L}^T\bar{D}\bar{L}$ where P is chosen to reduce fill-in. The matrix K is not strongly factorizable, in general. That is, not all the permutation matrices P provide LDL^T-factorizable matrices $\bar{K}$.

In recent years, a lot of work has been done on the stable decomposition of sparse symmetric *general indefinite* matrices and the efficient application of the decomposition to some practical problems (see [7], [8], [12], [23], [9]). The methods have been

*Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8 - Libeň, Czech Republic (tuma@cs.cas.cz, (+4202)66053251). A part of the work was done while the author visited the University of Bergamo. The work was supported by CNR Fellowship for young scientists, grant GA AS CR No. A1030103 and grant GA CR No. 101/00/1035.

based almost exclusively on the Bunch-Parlett factorization. In many situations, however, it is possible to make use of additional properties enjoyed by the saddle-point matrices. For example, we may use the fact that the left upper block H in (1.2) is positive definite. In this case the LDL^T decomposition of (1.2) can be computed by exploiting all the enhancements of supernodal Cholesky solvers. This approach seems to be a viable alternative to other approaches (see, e.g., [24], [22]). Let us mention the LDL^T factorization applied to matrices with the same block structure arising in optimization was used in [24]. The approach is based on the notion of *tiers*, using the block structure of the system rather than on structural properties of the symmetric Gaussian elimination. Zero diagonal elements are deferred until the end of the block phase by some static ordering. Therefore, this strategy is, in fact, a variant of the block Schur complement approach.

The approach proposed in this paper is based on the LDL^T decomposition of the permuted matrix K. The reordering is done in two steps. First, we construct an initial fill-in minimizing permutation of the whole matrix. Second, we strive to modify the permutation in order to obtain a P such that P^TKP is factorizable. The process strongly relies on the structural properties of the symmetric Gaussian elimination. For an important subclass of matrices from mixed-hybrid finite element (FEM) discretizations of the potential fluid flow problem we prove factorizability of permuted matrices. Though we are not able to prove formally the same result for more general systems with saddle-point matrices, we believe that the presented tools can be often successfully used even in these cases. Experimental results that compare the new strategy with the straightforward approach (using fill-in minimizing permutation for the first n rows and columns and then for the Schur complement) show the superiority of our strategy.

The outline of this paper is as follows. In §2 we present an example showing the importance of mixing variables from both blocks of (1.2) in the fill-in reducing ordering of K. We will establish some factorizability conditions on saddle-point matrices in §3. We also describe the class of $\mathcal{F}$-matrices which arise in mixed and hybrid FEM applications. For this class we propose an algorithm which determines a permutation matrix P in such a way that the factorizability of P^TKP is guaranteed. Experimental results in §4 demonstrate the effectiveness of the adopted approach. We assume that the reader is familiar with the basic graph-theoretic terminology used in sparse decompositions (see [6], [10], [18], [1]).

2. Permutations and fill-in for saddle-point matrices. The common approach in many applications with saddle-point matrices is to reduce the matrix to the Schur complement and then to make use of its definiteness (Schur complement $S = -AH^{-1}A^T$ of the matrix (1.2) is negative definite). Unfortunately, S can become rather dense.

Consider the example in Figure 1. A structure of matrix K is depicted on the left. Let the block H be determined by its first 6 rows and columns. The Schur complement S is completely dense. On the right we have a structure where the 7-th row and column were moved just after the first row and column, respectively. Then the principal submatrix of the permuted matrix determined by the last 10 rows and columns has the same pattern of nonzeros as K. Consequently, repeating this row and column exchanges recursively we get a matrix with small fill-in in its LDL^T decomposition.

$$
\begin{array}{cc}
 & \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} &
\begin{pmatrix}
* & * & & & & & * & & & & & \\
* & * & * & & & & & * & & & & \\
 & * & * & * & & & & & * & & & \\
 & & * & * & * & & * & & & * & & \\
 & & & * & * & * & & * & & & * & \\
 & & & & * & * & & & * & & & * \\
* & & & * & & & & & & & & \\
 & * & & & * & & & & & & & \\
 & & * & & & * & & & & & & \\
 & & & * & & & & & & & & \\
 & & & & * & & & & & & & \\
 & & & & & * & & & & & &
\end{pmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & \begin{array}{cccccccccccc} 1 & 7 & 2 & 3 & 4 & 5 & 6 & 8 & 9 & 10 & 11 & 12 \end{array} \\
\begin{array}{c} 1 \\ 7 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} &
\begin{pmatrix}
* & * & * & & & & & & & & & \\
* & & & & * & & & & & & & \\
* & & * & * & & & & * & & & & \\
 & & * & * & * & & & & * & & & \\
 & * & & * & * & * & & & & * & & \\
 & & & & * & * & * & * & & & * & \\
 & & & & & * & * & & * & & & * \\
 & & * & & & * & & & & & & \\
 & & & * & & & * & & & & & \\
 & & & & * & & & & & & & \\
 & & & & & * & & & & & & \\
 & & & & & & * & & & & &
\end{pmatrix}
\end{array}
$$

Figure 1: Matrix K providing a dense Schur complement and a sketch of the construction of its permutation inducing less fill-in in the LDL^T decomposition.

In this case, the recursive formation of the permutation resembles the choice of tile pivots in the ANALYZE phase of the MA47 code (see [8]).

3. LDL^T-factorizability of saddle-point matrices. If the right lower block of the matrix K is negative definite then any symmetric permutation of K provides a factorizable matrix (see [24], [25]). However, small diagonal elements often restrict the set of theoretically feasible permutations to some subset in practical applications. Here we consider the problem of permutations from a different point of view. Given a saddle-point matrix K, we ask for the permutation matrices P such that P^TKP is factorizable. Such permutations (orderings) will be called *feasible.*

First, we introduce some notation. A given sparse symmetric matrix Q can be structurally represented by its *associated undirected graph* $G(Q) = (V(Q), E(Q))$, where nodes in $V(Q)$ correspond to rows and columns of Q and edges in $E(Q)$ correspond to offdiagonal nonzero entries in Q. This graph is usually used to find the initial node ordering. (In our experiments in Section 4, we have used the multiple minimum degree (MMD) algorithm, see [11].)

Consider the partition of the vertices $V(Q)$ of $G(Q)$: $V(Q) = R(Q) \cup C(Q)$, $R(Q) \cap C(Q) = \emptyset$. The submatrix $Q(R(Q), C(Q))$ of Q can be represented by its *associated bipartite graph* $B(R(Q), C(Q))$ where the two sets of nodes correspond to $R(Q)$ and $C(Q)$, respectively, and edges correspond to nonzero elements in Q in rows from $R(Q)$ and columns from $C(Q)$. This graph will be used to describe the structure of the offdiagonal block of the matrix (1.2). Let $G(K) = (V(K), E(K))$ be the associated undirected graph of the saddle-point matrix (1.2). Denote by $R \subset V(K)$ the set of nodes $\{1, \ldots, n\}$ corresponding to the first n rows of K and $C \subset V(K)$ the set of nodes $\{n+1, \ldots, n+m\}$ corresponding to the last m columns of K. This notation for the initial partition of vertices will be used throughout the paper. In our strategy, starting from K we transform it by a sequence of permutations. In general, the intermediate permuted matrices will be denoted by $\bar{K}$. If it is necessary to distinguish between the input and output matrices in some algorithm we will denote the output matrix in the new ordering by $\bar{\bar{K}}$. Let $T = T(\bar{K})$ denotes an elimination tree of $\bar{K}$, i.e., the graph with n nodes $\{1, \ldots, n\}$ and with the arcs $(j, parent(j))$ where $parent(j) = min\{i > j | l_{ij} \neq 0\}$. Note that that the elimination tree is well-defined for any symmetric definite or indefinite matrix based on its structure completed by possibly missing diagonal entries. We use $T[i]$ to denote a subtree of T rooted in i. (We shall use $T[i]$ to denote both the subtree itself and the set of nodes in this subtree.) A simple necessary condition for a matrix to be factorizable can be described using the notion of an elimination tree.

PROPOSITION 3.1. *Matrix $\bar{K}$ is factorizable only if the leafs of the elimination tree $T(\bar{K})$ belong to R.*

Proof. If some leaf i of $T(\bar{K})$ is not from R then $a_{ii} = 0$ and $\bar{K}$ is not factorizable. $\square$

Before we proceed further we will show how to transform an input matrix $\bar{K}$ (K after some initial ordering) which may have some leaves of $T(\bar{K})$ in C to the permuted matrix $\bar{\bar{K}}$ satisfying the necessary condition from Proposition 3.1. The fact that the nodes of C form an independent set of $G(\bar{K})$ implies the following result.

LEMMA 3.1. *If $k \in C$ is a leaf of the elimination tree $T(\bar{K})$, then $parent(k) \notin C$.*

Therefore, if a node j has in $T(\bar{K})$ one or more leaves from C then j is from R. Denote the leaves of j that belong to C by $k_1, \ldots, k_\omega$. It is well known that there exists a reordering of $\bar{K}$ (corresponding to a different topological numbering of $T(\bar{K})$) that provides the same fill-in in the factors as $\bar{K}$ and that orders the nodes $k_1, \ldots, k_\omega$ consecutively after all the other subtrees rooted in the children of j (see [18]). We assume that our initial ordering satisfies this property. Moreover, we assume that it is a postordering of the elimination tree, i.e. all the subtrees of $T(\bar{K}))$ have their elements numbered consecutively. This fact will be used later. Denote by T_a the subgraph of the tree $T[j]$ given by $T[j]\backslash\{k_1, \ldots, k_\omega, j\}$. We need to reorder the set of nodes $\{j, k_1, \ldots, k_\omega\}$ so that $\bar{K}$ satisfies the necessary condition from Proposition 3.1. Namely, the elimination of the leaves $\{k_1, \ldots, k_\omega, j\}$ has to be delayed after the elimination of j and the new mutual order of the former leaves should be determined. The impact of delayed elimination on the size of fill-in was studied by Liu in [17]. In our special case of node delays we specify the ordering sequence of $\{j, k_1, \ldots, k_\omega\}$ directly. First j must be choosen. Then we take the nodes $k_i, i = 1, \ldots, \omega$ sequentially using locally the minimum degree strategy. In this way, we get the sequence $(j, \tilde{k}_1, \ldots, \tilde{k}_\omega)$. The situation is depicted on Figure 2. A "double-edge" is used to indicate a chain of nodes numbered consecutively in the tree.

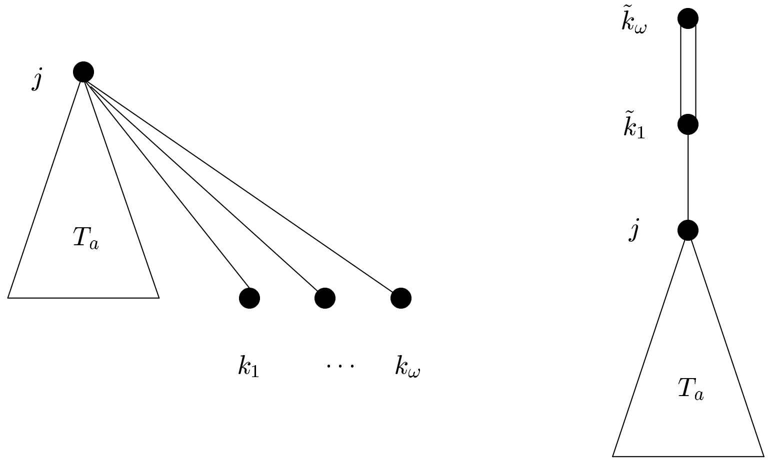


Figure 2: Elimination trees before and after local reordering of leaves.

After reordering all the sets of leaves from C in this way we get from $\bar{K}$ the permuted matrix $\bar{\bar{K}}$ satisfying the necessary condition in Proposition 3.1. In order to implement the procedure for finding a new reordering efficiently we must be able to efficiently find the nodes having one or more leaves. The following result may be used.

LEMMA 3.2. *Let the vertices of the elimination tree T of $\bar{K}$ be postordered. Let the $size(i), i = 1, \ldots, n+m$ be defined as follows:*

$$size(i) = \begin{cases} |T[i]| & \text{if } i \text{ is not a leaf of T} \\ 0 & \text{otherwise} \end{cases}$$

Let $first(i) = min\ T[i] \cap \{1, \ldots, n+m\}$. Then, if the number of leaves of i in T is denoted by $no\_of\_leaves(i)$ we have

$$no\_of\_leaves(i) = i - first(i) - \sum_{parent(c_j)=i} size(c_j).$$

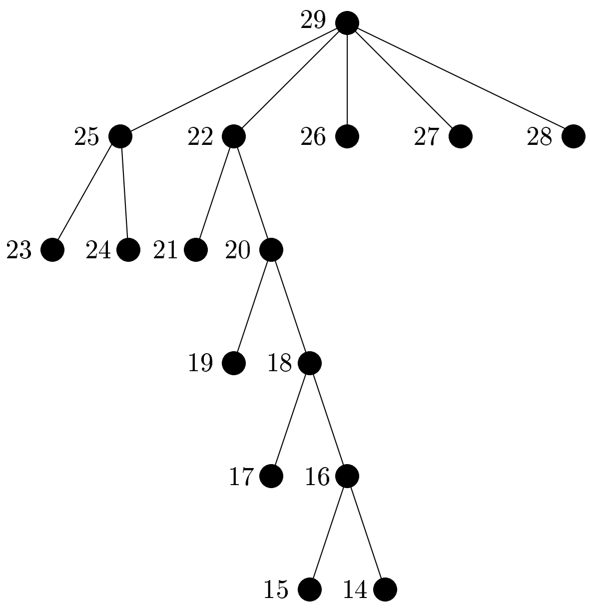


Figure 3: An example of a part of an postordered elimination tree.

The lemma is a simple consequence of the basic properties of postorderings (see [1], [15]). An example is given in Figure 3, which shows a part of a postordered elimination tree. The number of leaves of root 29 of the tree from Figure 3 can be computed as $no\_of\_leaves(29) = 29 - first(29) - (size(T[22]) + size(T[25])) = 29 - 14 - (3+9) = 3$.

Lemma 3.2 suggests an efficient way how to *implement* the whole modification of $\bar{K}$ to get $\bar{\bar{K}}$. First, we obtain a postordered elimination tree by standard algorithms (see [15]). Then we reorder it so that the leaves in each node are ordered consecutively after all the other subtrees rooted in children of the node. In fact, a similar transformation that reorders children sets in the elimination tree is a component of any up-to-date sparse symmetric Cholesky solver. Then, traversing the postordered elimination tree, we determine the modified permutation such that the output matrix $\bar{\bar{K}}$ satisfies the condition of Proposition 3.1.

Pseudocode of the algorithm *Leaves_Reordering* is provided below. The output of the algorithm is the new ordering sequence *newperm* (with respect to $\bar{K}$) which provides a $\bar{\bar{K}}$ satisfying the necessary condition from Proposition 3.1. Note that vector *size* keeps also the partial sums used in Lemma 3.2 for computations of *number_of_leaves*.

ALGORITHM 3.1. Leaves_Reordering.
input: The postordered elimination tree of an input matrix $\bar{K}$. The leaves in each node are ordered consecutively after all the larger subtrees rooted in children of the node.
output: New ordering sequence *newperm* such that the resulting matrix $\bar{\bar{K}}$ ordered according to the *newperm* satisfies the necessary condition from Proposition 3.1.

```
for j=1 to m+n
   first(j)=j;  size(j)=0;
end for
for  j=1 to m + n
   no_of_leaves=j-size(j)-first(j);
    if j  does not have leaves from C  then
        put j to newperm;
    else
       reorder j and its leaves which belong to C;
       put j, k̃_1, ..., k̃_ω to  newperm;
    end if
   first(parent(j))= min(first(parent(j)),j);
   size(j)=size(j)+no_of_leaves(j)+1;
   size(parent(j))=size(parent(j))+size(j);
end for
```

Another condition for the factorizability of $\bar{K}$ that can be efficiently tested is given in Proposition 3.2.

PROPOSITION 3.2. *Matrix $\bar{K}$ is factorizable only if each of its leading principal submatrices have the number of indices from R greater or equal to the number of indices from C.*

In order to permute $\bar{K}$ into $\bar{\bar{K}}$ such that the latter matrix satisfies the necessary condition of Proposition 3.2 we traverse the elimination tree using its postordering. The nodes are either put into the new ordering sequence or delayed. We delay putting of the current node $i \in C$ into the new ordering sequence whenever the current number of the indices from C in the sequence is equal to the current number of the indices from R. A delayed node is stored in a stack and tested for inclusion into the new ordering sequence after its father has been processed. The postordering guarantees efficient stack search in the same way as for multifrontal implementation of Cholesky decomposition (see [7]). Clearly, the elimination tree traversal after which $\bar{\bar{K}}$ satisfies the necessary conditions from both Propositions 3.1 and 3.2 can be implemented in one routine. Nevertheless, fulfilling both these conditions still does not guarantee that our matrix is factorizable. A condition that is both necessary and sufficient is given in Proposition 3.3. First, some additional notation. Denote for $i = 1, \ldots, n+m$ a sequence of leading principal submatrices of $\bar{K}$ by $\bar{K}^{(i)}$. Let their corresponding associated unsymmetric graphs be denoted by $\bar{G}^{(i)} = (\bar{V}^{(i)}, \bar{E}^{(i)})$. Furthermore, let $\bar{R}^{(i)} = \bar{V}^{(i)} \cap R, \bar{C}^{(i)} = \bar{V}^{(i)} \cap C$.

PROPOSITION 3.3. *Matrix $\bar{K}$ is factorizable if and only if all its submatrices $\bar{K}(\bar{R}^{(i)}, \bar{C}^{(i)})$, $i = 1, \ldots, n+m$ have full column rank.*

Whether these conditions are satisfied depends (in general) on the numerical values of the offdiagonal block of K. In the following, we will concentrate on the matrices arising in an important application of mixed-hybrid finite element discretizations of the fluid flow in porous media problem (see, e.g. [5]) where we can guarantee the matrix factorizability.

DEFINITION 3.1. *Let K be a saddle-point matrix (1.2). Using the notation introduced above we say that K is an $\mathcal{F}$-matrix if the submatrix $K(R, C)$ contains at most 2 elements per row. Moreover, if there are two nonzeros in a row of $K(R, C)$ then their sum is zero.*

Mixed-hybrid finite element formulation of some important problems like fluid flow in porous media is of a great interest in many fields including e.g. oil recovery and groundwater pollution modelling. Effective numerical simulations of these phenomena often require accurate renderings of the fluid velocities. For this reason, such type of formulations was extensively studied during the past years. $\mathcal{F}$-matrices are often the result of the discretizations of fluid flow domains. The offdiagonal structure (expressed by $K(R, C)$) is determined by the fact that the hybridization technique introduces into the matrix a simple topological information through the Lagrangian multipliers. For more details about the relation of the matrix structure and problem formulation we refer, e.g., to [5], [20], [19].

Depth-first or breadth-first *search* is a standard technique of systematically exploring nodes in a graph (see [1], [18]). The search starts from a node, called the root, which is marked as visited. Then the unvisited nodes are recursively searched using edges from marked to new (unmarked) nodes. Each time we visit a new node v by an edge from a marked node u, we mark it as visited and the arc (u, v) is added to the tree edges. Visited nodes and induced oriented arcs form a search tree. The length (number of arcs) of the longest path in the search tree S from the root to a leaf is called its depth and is denoted by $depth(S)$. Nodes of S having the same distance $\ell, 0 \leq \ell \leq depth(S)$ from the root form so-called *ℓ-th level set.* Note that while the arcs of the search tree are not determined in an unique way, level sets form the same partition of the graph nodes for all the search trees rooted in a given node.

Let $\hat{C} \subseteq C$, $\hat{R} \subseteq R$. Let Adj denote the adjacency operator in the bipartite graph $B(\hat{R}, \hat{C})$. Let $c \in \hat{C}$. Let $K(\hat{R}, \hat{C}) = (B_{ij})_{i \in \hat{R}, j \in \hat{C}}$. Define $Adj^{-1}(c) = \emptyset$, $Adj^{0}(c) = \{c\}$. Furthermore, for an integer α, $1 \leq \alpha$, define $Adj^{\alpha}(c) = Adj(Adj^{\alpha-1}(c))$. The following lemma describes some properties of the search tree in the bipartite subgraph associated with an $\mathcal{F}$-matrix.

LEMMA 3.3. *Let $\bar{K}$ be an $\mathcal{F}$-matrix. Let $R_1 \subseteq R$, $C_1 \subseteq C$. Let S be a search tree of $B(R_1, C_1)$ rooted in $c \in C_1$ and let Adj be the adjacency operator in $B(R_1, C_1)$. Let α, $0 \leq \alpha \leq depth(S)$ be zero or an even integer and assume that there are no paths* *of odd lengths from c to a leaf in S shorter than $\alpha+1$. Then we have*

$$\left(\sum_{k\in Adj^{\alpha}(c)} B_{*k}\right)_r \begin{cases} \neq 0 & if\ r \in Adj^{\alpha+1}(c)\backslash Adj^{\alpha-1}(c) \\ = 0 & otherwise. \end{cases}$$

In other words, $\left(\sum_{k\in Adj^{\alpha}(c)} B_{*k}\right)$ *has nonzeros only in the rows corresponding to vertices of the* $(\alpha+1)$*-th level set of the search tree* S.

Proof. For $\alpha = 0$ the assertion is clearly true. Assume that we have an even α, $2 \leq \alpha \leq depth(S)$. From the definition of adjacency set we have $\left(\sum_{k\in Adj^{\alpha}(c)} B_{*k}\right)_r =$ 0 for $r \notin Adj^{\alpha+1}(c)$. If $r \in Adj^{\alpha+1}(c) \cap Adj^{\alpha-1}(c)$ then, since each row of K contains at most two nonzeros, there exist uniquely determined indices k' and k'' and an even integer $\beta, 2 \leq \beta \leq \alpha$ such that (k', r) and (r, k'') are arcs of S and $k' \in Adj^{\beta-2}(c)$, $k'' \in Adj^{\beta}(c)$. The zero sum condition on the $\mathcal{F}$-matrix then implies $\left(\sum_{k\in Adj^{\alpha}(c)} B_{*k}\right)_r =$ 0. □

The following Theorem gives a necessary and sufficient condition for factorizability of the $\mathcal{F}$-matrix. Moreover, this condition can be tested cheaply.

Theorem 3.1. *Let $\bar{K}$ be an $\mathcal{F}$-matrix. Let $R_1 \subseteq R$, $C_1 \cup \{c\} \subseteq C, c \notin C_1, Adj(c) \neq \emptyset$. Let S be a search tree of $B(R_1, C_1 \cup \{c\})$ rooted in c and let Adj be the adjacency operator in $B(R_1, C_1 \cup \{c\})$. Assume that $rank(\bar{K}(R_1, C_1)) = |C_1|$. Then*

$$rank(\bar{K}(R_1, C_1 \cup \{c\})) < |C_1 \cup \{c\}|$$

if and only if there are no paths of odd length from the root to a leaf in S.

Proof. Suppose that $rank(\bar{K}(R_1, C_1 \cup \{c\})) < |C_1 \cup \{c\}|$. Using the assumption on $rank(\bar{K}(R_1, C_1))$, it follows that there exist a $\Gamma = \{\gamma_1, \ldots, \gamma_{|\Gamma|}\} \subseteq C_1$, and coefficients $c_{\gamma_1}, \ldots, c_{\gamma_{|\Gamma|}}$ such that

$$B_{*c} + \sum_{j=1}^{|\Gamma|} c_{\gamma_j} B_{*\gamma_j} = 0. \tag{3.1}$$

Matrix and search tree properties then imply that in order to satisfy (3.1), all nodes of S should be involved in this linear combination and all the coefficients c_{γ_j} are 1. Therefore, (3.1) transforms into $\sum_{\delta\in S} B_{*\delta} = 0$.

Assume now that there is a path of odd length from the root to a leaf in S. Let α be the length of one of such shortest paths. Denote by d its terminal node being the leaf node of S. Then we have $d \in Adj^{\alpha}(c)$. Using Lemma 3.3 we get that $\left(\sum_{k\in Adj^{\alpha-1}(c)} B_{*k}\right)_d \neq 0$. From the search tree properties it follows then that $\left(\sum_{\delta\in S} B_{*\delta}\right)_d \neq 0$ and we obtain a contradiction.

On the other hand, if there are no paths of odd length from the root to leaves in S then $\sum_{k\in Adj^{depth(S)}(c)} B_{*k} = 0$ for even $depth(S) \geq 2$. Consequently, $rank(\bar{K}(R_1, C_1 \cup \{c\})) < |C_1 \cup \{c\}|$, which completes the proof. □

Note that the technical assumption $Adj(c) \neq \emptyset$ used in Theorem 3.1 is very natural as the following result shows.

LEMMA 3.4. *Take sequentially nodes of the postordered elimination tree* $T(\bar{K})$. *Let* $R_1 \cup C_1$ *for* $R_1 \subseteq R, C_1 \subseteq C$ *be the first* $|R_1 \cup C_1|$ *nodes and* $c \in C$ *be the* $(|R_1 \cup C_1| + 1)$*-th node. If* c *is not a leaf of* $T(\bar{K})$ *then* $Adj_{B(R_1, C_1 \cup \{c\})}(c) \neq 0$.

The proof of Lemma 3.4 follows directly from Theorem 2.4 in [15]. Therefore, if our $\bar{K}$ satisfies the necessary conditions of Propositions 3.1 and 3.2, then the condition $Adj(c) \neq \emptyset$ is also satisfied using the sequential construction of R_1 and C_1.

Theorem 3.1 implies how to test the rank of a $\mathcal{F}$-matrix that is extended by adding a new column c and (symmetrically) a new row. The algorithm uses the graph of the offdiagonal block determined by current sets R_1, C_1 and by the column $c \in C_1$. This test is used within the procedure for modification of the current permutation of a $\mathcal{F}$-matrix $\bar{K}$. Namely, the nodes of a postordered elimination tree of the matrix $\bar{K}$ are traversed. The condition of Theorem 3.1 is used to test a new column c from C. The actual test strategy is based on the *breadth-first search.* In each such search we either find the above mentioned path of odd length or the whole search tree of an even length is found. In the former case we put c into the new ordering sequence. In the latter case we put c on the stack as the infeasible one. Then the *feasible* delayed columns are tested for inclusion into the new ordering sequence in the same way. Subsequent test of a delayed column for inclusion into the new ordering sequence is performed after its parent was processed. At this time we change its status to feasible. At the end of the procedure we get the factorizable matrix $\bar{\bar{K}}$.

The following two algorithms describe our strategy. The algorithm *Search* checks whether the condition of Theorem 3.1 is satisfied for a given node c. The algorithm *Test_Rank* then describes the whole procedure. In order to test the linear dependency of columns in the *Search* algorithm we use the bipartite graph $B(\bar{\bar{R}}, \bar{\bar{C}})$ formed from the reordered columns and rows which are temporarily stored in the new ordering sequence *newperm*. The columns which are temporarily delayed are kept in a heap.

ALGORITHM 3.2. Search($\bar{\bar{R}}$,$\bar{\bar{C}}$,p,i).
input: The bipartite graph $B(\bar{\bar{R}}, \bar{\bar{C}})$ constructed using the first p nodes in the new ordering *newperm*; node i.
output: Logical value *found* denoting whether the path of odd length from i to a leaf of the search tree S rooted in i was found.

Construct the breadth-first search tree of $B(\bar{\bar{R}}, \bar{\bar{C}})$ rooted in i;
if the above mentioned path was found **then**
    found=*true;*
else
    found=*false;*
end if

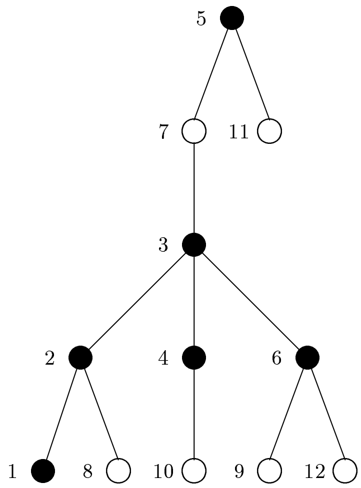
ALGORITHM 3.3. Test_Rank.
input: The input $\mathcal{F}$-matrix $\bar{K}$ and its postordered elimination tree.
output: The new permutation sequence *newperm* (related to $\bar{K}$) and $\bar{\bar{K}}$ (permuted $\bar{K}$) such that all the submatrices $\bar{\bar{K}}(\bar{\bar{R}}^{(i)}, \bar{\bar{C}}^{(i)})$ of $\bar{\bar{K}}$ have full column rank.

heap=$\emptyset$*;*
$p = 0$*;*
$\bar{\bar{R}} = \emptyset$

$\bar{\bar{C}} = \emptyset$
for j=*1*,$n+m$
    do while $heap \neq \emptyset$
        choose k from *heap* having minimal $after(k)$ among all nodes in *heap;*
        if *after(k)* $> j$ **then**
            break*;*
        end if*;*
        Search($\bar{\bar{R}}$,$\bar{\bar{C}}$,*p*,*k*)*;*
        if *(.not.found)* **then**
            after(k)=parent(k);
        else
            remove k from *heap*;
            put k to *newperm;*
            $\bar{\bar{C}} = \bar{\bar{C}} \cup \{k\}$*;*
            $p = p + 1$*;*
        end if*;*
    end do*;*
    if $j \in R$ **then**
        put j to *newperm;*
        $\bar{\bar{R}} = \bar{\bar{R}} \cup \{j\}$*;*
        $p = p + 1$*;*
    else
        Search($\bar{\bar{R}}$,$\bar{\bar{C}}$,*p*,*j*)*;*
        if *(.not.found)* **then**
            after(j)=parent(j);
            put j *to heap;*
        else
            put j to *newperm;*
            $\bar{\bar{C}} = \bar{\bar{C}} \cup \{j\}$*;*
            $p = p + 1$*;*
        end if*;*
    end if*;*
end for.

Note that the simple structure of the offdiagonal block suggests efficient breadth-first searches. We need to construct the search tree in the bipartite graph of the current offdiagonal block for each column from C and from the heap. The basic implementational trick is to reuse the constructed search tree if the new search tree can be obtained by extending the current one. We observed in our experiments that the search was finished rather early in most cases. This can be atributed to the fact that MMD reordering tends to construct wide elimination trees for problems arising from discretization of topologically regular meshes. Nevertheless, it may happen that some of the search trees have many levels.

The procedure $Test\_Rank$ can be used even for more general matrices. However, in this case we do not have guaranteed the factorizability of the output matrix and, in addition, the search can be more time consuming.



$$
\begin{array}{cc}
 & \begin{array}{cccccccccccc} 1 & 2 & 8 & 4 & 10 & 6 & 9 & 12 & 3 & 7 & 5 & 11 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 8 \\ 4 \\ 10 \\ 6 \\ 9 \\ 12 \\ 3 \\ 7 \\ 5 \\ 11 \end{array} &
\begin{pmatrix}
* & * & & & & & & & & * & & \\
* & * & * & & & & & & * & f & & \\
 & * & & & & & & & f & f & * & \\
 & & & * & * & & & & * & * & * & \\
 & & & * & & & & & & & & \\
 & & & & & * & * & * & & & * & \\
 & & & & & * & & f & * & & f & \\
 & & & & & * & f & & f & & f & \\
 & * & f & * & & & * & f & * & f & f & \\
* & f & f & * & & & & & f & & f & \\
 & & * & * & & * & f & f & f & f & * & * \\
 & & & & & & & & & & * &
\end{pmatrix}
\end{array}
$$

Figure 4: The elimination tree for the matrix from Figure 1 reordered by MMD (left) and the final reordered matrix after application of Leaves_Reordering algorithm (right). Circles which are not filled correspond to nodes from C.

Consider the example matrix from Figure 1. MMD reordering of the matrix provides the elimination tree depicted in Figure 4 on the left. Here we used the reordering provided by the Matlab function *symmmd*. The matrix in Figure 4 on the right is then reordered by the output permutation coming from the algorithm *Leaves_Reordering* that was applied to the postordered elimination tree. We can see that that processing of a lot of leaves was delayed. Nevertheless, the overall fill-in is clearly modest. The algorithm $Test\_Rank$ then does not change the obtained order. This was a typical behaviour in our experiments where the $Test\_Rank$ algorithm delays pivots only occasionally.

4. Implementation and experimental results. In this section, we present results of our experiments with the new ordering strategy on several $\mathcal{F}$-matrices arising in mixed hybrid FEM discretizations of 3D potential fluid flow problems (see [20]). Note that in practical calculations the matrix patterns for flow field computations are often used repeatedly, e.g., in the chemical transport outer loops. We are interested in the amount of fill-in in the factors because this crucially determines the time of numerical decomposition. The amount of fill-in was compared with that of a solver based on the Schur complement approach. In this case the MMD algorithm was applied to the principal leading submatrix H and then to the Schur complement system. This ordering strategy (ordering in stages) can be interpreted as using MMD ordering with the constraint that nodes from R are ordered first (see [16], [2]).

In our experiments we computed right-hand sides corresponding to a solution of all ones. We did not face any significant numerical instability in the computation. This seems to correspond to the results in [13], cf. [22]. Namely, the stability of the computation is determined by a ratio of the maximal singular value of the offdiagonal block and of the maximal eigenvalue of the inverse of the diagonal block H of the

$\mathcal{F}$-matrix. In our case, the ratio can be reasonably bounded using discretization parameter h of the problem. For details we refer to [19].

Our numerical procedure for solving the linear algebraic systems with saddle-point matrices consists of three stages:

- MMD ordering, elimination tree and postordering construction.
- Modification of permutations so that the permuted matrix satisfies conditions from Propositions 3.1, 3.2 and Theorem 3.1.
- Left-looking supernodal LDL^T solver (see [21]).

Table 1 compares the size of L factors of the final matrix $\bar{\bar{K}}$ for both approaches. Test matrices come from the mixed hybrid FEM discretizations of 3D potential fluid flow problem based on Raviart-Thomas elements of the lowest order. Namely, the velocity is approximated by vector functions linear on every element and piezometric potential is approximated by elementwise constant functions. The matrix dimension and the dimension of its left upper block are $n + m$ and n, respectively, nnz denotes number of nonzeros in the symmetric part of the original matrix. The column marked "Schur" contains factor size (number of the nonzeros) in the Schur complement approach. The column labelled "LDL^T" contains factor size required by LDL^T decomposition using the new strategy proposed in the paper.

$Matrix$	$n+m$	n	nnz	$Schur$	LDL^T
S3P	477	270	1539	3151	2957
LI5	868	480	2208	5098	4686
LI6	3707	2100	9575	40269	34498
M3P	3744	2160	12312	46504	44002
DORT2	12992	7515	42935	296818	231312
DORT	22967	13360	76199	641828	551215

Table 1: Size of the factor L in the LDL^T decomposition using two different ordering strategies.

It is clear that the new ordering provided better results. This is in agreement with general opinion that the ordering that is more general than the one used in the Schur complement approach may also provide less fill-in (see, e.g., [25]). An interesting question is how much the original MMD ordering had to be modified for matrices in these experiments. It is natural that MMD tends to choose the leaves from C since these nodes have degrees at most two. In our test cases more than 80% of the leaves were taken from C and had to be delayed. This behavior can be observed in Figure 4 as well. The delay induced an increase in fill around 30%. Delay of nodes based on Theorem 3.1 was rather rare with only a few cases for each matrix. Note that there were no delays of nodes based on Theorem 3.1 in the two smallest matrices.

5. Conclusions. This paper presents a new ordering strategy for the class of symmetric saddle-point matrices. The approach is theoretically justified for an important subclass of matrices arising in mixed and hybrid formulations of the finite element method. The fill-in size obtained during the LDL^T decomposition of the matrices is compared with that obtained in the straightforward Schur complement strategy using efficient fill-in minimizing ordering. Our ordering is shown to be superior.

In spite of lack of more general theoretical results we would like to consider the techniques mentioned here as ordering routines that help to solve saddle-point systems in various applications. Namely, if we change the initial fill-in minimizing permutation so that $\bar{K}$ satisfies Propositions 3.1 and 3.2, the most important sources of instabilities of the decomposition will be removed. This can be combined with other practical rules, e.g., as a pivot modification (see [26]) in both direct solvers and preconditioner computations. We will study use of the algorithms as preprocessing schemes in the Bunch-Parlett sparse supernodal left-looking or right-looking (MA47) linear solvers for saddle-point matrices. Recently, an interesting class of preconditioners constructed from spanning trees of matrix graphs or their extensions and based on work of P. Vaidya was introduced [3], [4]. Construction of the preconditioners needs to test linear independence of sets of structurally simple vectors having thus connecting links to our work.

6. Acknowledgment. The author gratefully acknowledges the hospitality provided by Emilio Spedicato at the University of Bergamo. He would also like to thank Jiří Mužák and Dalibor Frydrych for preparing the test problems and to Michele Benzi and Zdeněk Strakoš for their comments on the previous version of the paper.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman. Data Structures and Algorithms. *Addison-Wesley, Reading, MA*, 1983.

[2] C. Ashcraft, D. Pierce, D.K. Wah and J. Wu. The Reference Manual for SPOOLES, Release 2.2: An Object Oriented Software Library for Solving Sparse Linear Systems of Equations. Software Description Report, *Boeing Computer Services*, available from `www.netlib.org`, 1999.

[3] M. Bern, J.R. Gilbert, B. Hendrickson N. Nguyen and S. Toledo. Support graph preconditioners, *Sandia Labs Technical Report, submitted to SIAM J. Matrix Anal. Appl.*, 2000

[4] E.G. Boman, D. Chen, B. Hendrickson and S. Toledo. Maximum-weight-basis preconditioners, *Sandia Labs Technical Report, submitted to Numerical Linear Algebra Appl.*, 2001

[5] F. Brezzi and M. Fortin. Mixed and Hybrid Finite Element Methods. *Springer Verlag*, 1991.

[6] I. Duff, A. Erisman and J. Reid. Direct methods for sparse matrices. *Clarendon Press*, 1986.

[7] I.S. Duff and J.K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, 9:302–325, 1983.

[8] I.S. Duff, N.I.M. Gould, J.K. Reid, J.A. Scott and K. Turner. The factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.*, 11:181–204, 1991.

[9] R. Fourer and S. Mehrotra. Solving symmetric indefinite systems in an interior-point method for linear programming. *Math. Prog.*, 62:15–39, 1993.

[10] J.A. George and J.W.H. Liu. Computer Solution of Large Sparse Positive Definite Systems. *Prentice-Hall, Englewood Cliffs, NJ*, 1981.

[11] J.A. George and J.W.H. Liu. The evolution of the minimum degree algorithm. *SIAM Rev.*, 31:1–19, 1989.

[12] P.E. Gill, W. Murray, D.B. Poncelen and M.A. Saunders. Solving reduced KKT systems in barrier methods for linear and quadratic programming. *Technical report SOL 91-7, Systems Optimization Laboratory, Stanford University, Stanford, CA*, 1991.

[13] P.E. Gill, M.A. Saunders and J.R. Shinnerl. On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM J. Matrix Anal. Appl.*, 17:35–46, 1996.

[14] G.H. Golub and C.F. Van Loan. Unsymmetric positive definite linear systems. *Linear Algebra Appl.*, 28:85–98, 1979.

[15] J.W.H. Liu. A compact row storage scheme for Cholesky factors using elimination trees. *ACM Trans. Math. Software*, 12:127–148, 1986.

[16] J.W.H. Liu. The minimum degree ordering with constraints. *SIAM J. Sci. Statist. Comput.*, 10:1136–1145, 1989.

[17] J.W.H. Liu. A tree model for sparse symmetric indefinite matrix factorization. *SIAM J. Matrix Anal. Appl.*, 9:26–39, 1988.

[18] J.W.H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11:134–172, 1990.

[19] J. Maryška, M. Rozložník and M. Tůma. The potential fluid flow problem and the convergence rate of the minimal residual method. *Numerical Linear Algebra Appl.*, 3:525–542, 1996.

[20] J. Maryška, M. Rozložník and M. Tůma. Schur complement systems in the mixed-hybrid finite element approximation of the potential fluid flow problem, *SIAM J. Sci. Comput.*, 22 (2000), pp. 704–723.

[21] E.G. Ng and B.W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14:1034–1056, 1993.

[22] M.A. Saunders and J.A. Tomlin. Stable reduction to KKT systems in barrier methods for linear and quadratic programming. *Preprint. Presented at the International Symposium on Optimization and Computation, Hayama, Kanagawa, Japan, August 12–16, 1996.*

[23] K. Turner. Computing projections for the Karmarkar algorithm. *Linear Algebra Appl.*, 152:141–154, 1991.

[24] R.J. Vanderbei. Symmetric quasi-definite matrices. *Report SOR–91–10, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ*, 1991.

[25] R.J. Vanderbei. Symmetric quasi-definite matrices. *SIAM J. Optim.*, 5:100–113, 1995.

[26] R.J. Vanderbei and T.J. Carpenter. Symmetric indefinite systems for interior point methods. *Math. Prog.*, 58:1–32, 1993.