A LINEAR REORDERING ALGORITHM FOR PARALLEL PIVOTING
OF CHORDAL GRAPHS*

JOSEPH W. H. LIU† AND ANDRANIK MIRZAIAN†

Abstract. This paper provides an efficient algorithm for generating an ordering suitable for the parallel
elimination of nodes in chordal graphs. The time complexity of the reordering algorithm is shown to be linear
in the size of the chordal graph. The basic parallel pivoting strategy is originally by Jess and Kees [IEEE Trans.
Comput., C-31 (1982), pp. 231–239]. The relevance of the reordering to parallel factorization of sparse matrices
(not necessarily chordal) is also discussed.

Key words. chordal graph, sparse matrix, parallel pivoting, zero deficiency, sparse elimination

AMS(MOS) subject classifications. 65F50, 65F25

1. Introduction. In this paper, we consider the parallel pivoting problem, which
arises in the exploitation of parallelism in the direct solution of large sparse symmetric
positive definite linear systems. For a given large sparse symmetric matrix \( A \), we want
to determine an ordering which is appropriate in terms of preserving sparsity and exploiting
parallelism in its Cholesky factorization. Parallel pivoting strategies have been studied
by Aggabadd and Jordan [1], Betancourt [2], Calahan [3], Huang and Wing [7], Jess
and Kees [8], Peters [10], and others.

A modular approach has been used by Jess and Kees [8], which determines a good
fill-reducing ordering \( P \) for \( A \), and then finds an equivalent reordering \( \tilde{P} \) (that is, one
that preserves the filled graph) suitable for parallel elimination. Since \( PAP \) and \( \tilde{P} A \tilde{P} \)
have the same set of fills, \( \tilde{P} \) has the same fill-reducing property as \( P \). Moreover, it is well
known that the filled graph of \( PAP \) (that is, the graph of \( G(PAP) \) together with the
fill edges due to factorization) is chordal [11]. Therefore, the problem is reduced to
finding a perfect elimination ordering for a chordal graph, which is appropriate for parallel
elimination. In [8], Jess and Kees have also provided such a parallel pivoting strategy
for chordal graphs. It is shown in [9] that the resulting reordering has the desirable
property of minimizing the number of parallel elimination steps among the class of
perfect orderings (orderings with no fill).

In this paper, we provide an efficient algorithm to generate this parallel pivoting
sequence. Central to the algorithm is an effective method to identify nodes with zero
deficiency in a given chordal graph. The method is based on an interesting property
of perfect elimination orderings. Repeated use of this zero-deficiency test results in an overall
reordering algorithm. We prove that the time complexity of this new algorithm is linear
with respect to the number of nodes and number of edges in the chordal graph.

The reader is assumed to be familiar with basic notions related to chordal graphs.
The book by Golubic [6] contains an excellent treatment of the subject. Moreover,
graph-theoretic notions relevant to sparse matrix computation are also assumed. The
reader is referred to George and Liu [5].

An outline of this paper follows. In §2, the parallel pivoting strategy by Jess and
Kees [8] for chordal graphs is reviewed. In §3, we provide a simple, efficient zero-
deficiency test on nodes in a given chordal graph. We apply this result in § 4 to obtain a linear time reordering algorithm that will produce a desirable parallel elimination sequence. Section 5 contains the concluding remarks.


2.1. The parallel elimination algorithm. A graph is called chordal (or triangulated) if every cycle of length 4 or more has a chord; or equivalently, if it has a perfect elimination ordering [6]. It is known that a perfect elimination ordering of a chordal graph can be found in linear time [6], [12]. In [8], Jess and Kees provide a reordering scheme tailored for parallel elimination of a chordal graph. In [9], a minor variant of their strategy is shown to produce orderings with minimum number of parallel node elimination steps among all perfect elimination orderings. It is indeed a relevant scheme for parallel elimination.

In this section, we briefly review the Jess and Kees' strategy in preparation for our algorithm in the next section. We first introduce some terminology. Let \( A \) be a given \( n \)-by-\( n \) sparse symmetric positive definite perfect elimination matrix, that is, its graph \( G(A) \) is chordal. We shall use \( G(A) \) and \( G \) interchangeably to refer to this chordal graph. For convenience, we shall also use \( G \) to refer to the set of nodes in this graph.

Two nodes are said to be independent if they are not adjacent. A node is said to have no (or zero) deficiency if its adjacent set is a clique [11]. In the literature, such a node is also referred to as simplicial [6]. It is well known that a chordal graph always has one node with no deficiency. Furthermore, if the graph is not a clique, there are at least two such independent nodes. Jess and Kees make use of this observation to develop a parallel pivoting strategy [8, procedure "e-tree", p. 233], which we describe in our terminology below.

ALGORITHM 2.1. Parallel-Elimination \((G)\) \( \{ G \) chordal graph \}

begin
\[ G_0 := G; \]
\[ i := 0; \]
while \( G_i \neq \emptyset \) do begin
\[ S_i := \text{the set of all nodes with no deficiency in } G_i; \]
\[ R_i := \text{a maximum independent subset of } S_i; \]
\[ G_{i+1} := G_i - R_i \{ \text{eliminate the nodes of } R_i \text{ from } G_i \}; \]
\[ i := i + 1 \]
end
end.

It should be clear that we can exit from the "while" loop whenever the subgraph \( G_i \) becomes a clique. In such a case, each subsequent \( R_j \) (for \( j \geq i \)) consists of only one node and the remaining nodes can be numbered in any order. Associated with the algorithm is a sequence of chordal graphs:

\[ G_0, G_1, \cdots, G_m, \]

and a sequence of subsets of independent nodes:

\[ R_0, R_1, \cdots, R_m, \]

where each \( R_i \) is a maximum independent set of nodes with no deficiency in \( G_i \).

It is worthwhile to point out that Jess and Kees [8] make the important observation that the set \( S_i \) consists of disjoint cliques. This implies that any maximal independent subset of \( S_i \) is maximum. Therefore it is sufficient to consider maximal independent
subset $R_i$ of $S_i$. Intuitively, the algorithm eliminates as many nodes in “parallel” as possible in each step. This explains the appropriateness of the scheme for parallel factorization.

We include an example in Fig. 2.1 with eight nodes. On applying Algorithm 2.1, the following shows one possible choice of independent subsets:

<table>
<thead>
<tr>
<th>Step</th>
<th>$S_i$</th>
<th>Selected $R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${a, b, c, f, h}$</td>
<td>${a, c, f, h}$</td>
</tr>
<tr>
<td>1</td>
<td>${b, d, g}$</td>
<td>${b, d}$</td>
</tr>
<tr>
<td>2</td>
<td>${e, g}$</td>
<td>${g}$</td>
</tr>
<tr>
<td>3</td>
<td>${e}$</td>
<td>${e}$</td>
</tr>
</tbody>
</table>

The sequence of chordal graphs $\{G_i\}$ is also given in Fig. 2.1 to illustrate the algorithm.

Implicit in Algorithm 2.1 is that nodes are reordered in the order as given by the sequence $\{R_i\}$. Within each subset $R_i$, the nodes can be numbered in any order. Note that the resulting reordering from this algorithm may not be unique due to the different choices of the independent subset $R_i$ at each step, and the freedom to number nodes within $R_i$.

The description of Algorithm 2.1, as it is, does not offer an efficient implementation. Indeed, Jess and Kees [8, p. 238] point out that in this reordering algorithm, “the major source of complexity is the test on zero deficiency.” But they have not addressed this zero-deficiency test problem. Our contribution in this paper is to provide a linear-time algorithm to determine a parallel pivoting sequence as specified by Algorithm 2.1.

2.2. Use of the parallel elimination reordering for nonchordal graphs. Although Algorithm 2.1 is designed for chordal graphs, it is actually applicable in a more practical setting to parallel sparse Cholesky factorization. Consider a given sparse matrix $A$ with graph $G(A)$. Although the graph $G(A)$ is most likely nonchordal, it is well known that its filled graph is chordal [11]. Indeed, the ordering inherited from $A$ will clearly create no additional fill on the filled matrix, and hence is a perfect elimination ordering. The parallel elimination ordering of Algorithm 2.1 is, therefore, applicable to the filled graph of $A$.

Ideally, we want to find an ordering for the matrix $A$, which is appropriate for both reducing fill and exploiting parallelism. A modular approach to this is to determine first a good fill-reducing ordering, which defines the filled graph. Then, a reordering for this filled graph can be obtained by Algorithm 2.1, and the reordering minimizes the number of parallel elimination steps among all perfect elimination orderings of the filled graph. In other words, the competing issues of fill reduction and parallelism exploitation are
being dealt with in separate steps. We may view this modular approach as having the following phases.

(a) (Ordering) Determine a fill-reducing ordering \( P \) for the given \( G(A) \);
(b) (Filled Graph) Form the filled graph of \( G(PAP') \) explicitly or implicitly;
(c) (Reordering) Apply Algorithm 2.1 to the filled graph (which is chordal) to obtain an equivalent reordering \( \tilde{P} \) of \( G(A) \).

It should be pointed out that the problem of finding good orderings for reducing fill and exploiting parallelism is still under active research. The modular approach discussed here has the advantage of being simple and efficient. An alternative approach is to find such orderings directly from the structure of \( G(A) \). The work in [1], [2] explores this direct approach.

3. Zero-deficiency test. The key to Algorithm 2.1 is the step that determines a maximum independent subset \( R_i \) of nodes with no deficiency in the current chordal subgraph \( G_i \). In this section, we provide a linear-time algorithm to find such an independent subset based on an efficient zero-deficiency test. We first provide a characterization of nodes with zero deficiency using a perfect ordering of the given chordal graph.

Let \( G = G(A) \) be the given chordal graph and \( x_1, x_2, \ldots, x_n \) be a perfect elimination node sequence. We shall use the notation \( \text{Adj}_G(x_j) \) for the adjacent set of the node \( x_j \) in \( G \). Following Rose [11], we define the monotonically adjacent set of the node \( x_j \) to be

\[
\text{Mad}_G(x_j) = \{ x_i \in \text{Adj}_G(x_j) \mid i > j \}.
\]

We shall refer to \( \text{deg}_G(x_j) = |\text{Adj}_G(x_j)| \) and \( \text{Mdeg}_G(x_j) = |\text{Mad}_G(x_j)| \) as the degree and monotone degree of \( x_j \), respectively. When the chordal graph \( G \) is clear from context, we shall omit the subscript \( G \) and use \( \text{Adj}(x_j) \), \( \text{Mad}(x_j) \), \( \text{deg}(x_j) \), and \( \text{Mdeg}(x_j) \) instead.

For each node \( x_j \), define \( f_j \) as

\[
f_j = \min \{ k \mid x_k \in \text{Adj}(x_j) \cup \{ x_j \} \}.
\]

In matrix terms, \( f_j \) is the first nonzero (including the diagonal) in the \( j \)th row of the sparse matrix \( A \).

**Lemma 3.1.** \( \text{Mad}(x_j) \cup \{ x_{f_j} \} \subseteq \text{Adj}(x_j) \cup \{ x_j \} \), for \( j = 1, \ldots, n. \)

**Proof.** Let \( k = f_j \). Since the sequence is a perfect elimination sequence, when \( x_k \) is eliminated, \( \text{Mad}(x_k) \) is the set of its adjacent nodes at the elimination time and is therefore a clique. But \( x_j \in \text{Mad}(x_k) \) so that \( x_j \) is adjacent to every node in \( \text{Mad}(x_k) - \{ x_j \} \).

**Lemma 3.2.** The node \( x_j \) has no deficiency in the chordal graph \( G \) if and only if

\[
\text{Adj}(x_j) \cup \{ x_j \} \subseteq \text{Mad}(x_j) \cup \{ x_{f_j} \}.
\]

**Proof.** "only if" part. Let \( k = f_j \). Assume that the node \( x_j \) has no deficiency in the chordal graph \( G \). Since \( x_k \in \text{Adj}(x_j) \cup \{ x_j \} \), and \( \text{Adj}(x_j) \cup \{ x_j \} \) forms a clique in \( G \), we have

\[
\text{Adj}(x_j) \cup \{ x_j \} \subseteq \text{Adj}(x_k) \cup \{ x_k \}.
\]

The result then follows from the fact that \( x_k \) is the first adjacent node of \( x_j \) in the perfect ordering.

"if" part. Assume that the condition on adjacent sets of \( x_j \) and \( x_{f_j} \) is given. Again, let \( k = f_j \). Since \( \text{Mad}(x_k) \) is the set of adjacent nodes of \( x_k \) at its elimination time, it, together with the node \( x_k \), forms a clique in \( G \). This implies that its subset \( \text{Adj}(x_j) \cup \{ x_j \} \) must also be a clique in \( G \).

**Corollary 3.3.** If \( f_j = j \), then the node \( x_j \) has no deficiency in the chordal graph \( G \).
COROLLARY 3.4. The node $x_j$ has no deficiency in the chordal graph $G$ if and only if $\text{Adj}(x_j) \cup \{x_j\} = \text{Mdeg}(x_j) \cup \{x_j\}$.

Corollary 3.4 provides an efficient zero-deficiency test. By definition, a node $v$ has no deficiency if and only if

$$\text{Adj}(v) \cup \{v\} \subseteq \text{Adj}(w) \cup \{w\},$$

for every node $w$ adjacent to $v$. Corollary 3.4 makes use of a perfect elimination sequence to perform the deficiency test using (at most) one adjacent node of $v$. An even simpler test is provided in the next theorem based only on node degrees.

THEOREM 3.5. The node $x_j$ has no deficiency in the chordal graph $G$ if and only if $\text{deg}(x_j) = \text{Mdeg}(x_j)$.

Proof. The "only if" part follows directly from Corollary 3.4. The "if" part follows from the results of Lemma 3.1 and Corollary 3.4. □

For a given chordal graph $G$ with a perfect elimination ordering, we can use Theorem 3.5 to determine the set of all nodes with no deficiency. The following algorithm adapts this idea to find a maximum independent subset $R$ of nodes with no deficiency. As before, we assume that $x_1, x_2, \cdots, x_n$ is a given perfect elimination sequence for $G$.

ALGORITHM 3.1. No Deficiency ($G$, $R$)

begin

$R := \emptyset$;
for $j := 1$ to $n$ do

compute $\text{deg}(x_j)$, $\text{Mdeg}(x_j)$, and $f_j$;
unmark all nodes in $G$;
for $j := 1$ to $n$ do

if $x_j$ is unmarked and $\text{deg}(x_j) = \text{Mdeg}(x_j)$

then add $x_j$ to $R$ and mark nodes in $\text{Adj}(x_j)$
end.

end.

The marking involved in the algorithm is to ensure that the set $R$ is independent.

It is interesting to note that in Algorithm 3.1, if we omit the zero-deficiency test on degrees, it is essentially the algorithm by Gavril [4] to determine a maximum independent set (not necessarily of no deficiency) of a chordal graph. The linear time complexity of Algorithm 3.1 is clear and we state the following result without proof.

THEOREM 3.6. Algorithm 3.1 finds a maximum independent subset of nodes with no deficiency in $G$ in $O(n + e)$ time, where $n$ is the number of nodes and $e$ the number of edges in the given chordal graph.

4. A linear reordering algorithm. It is easy to incorporate Algorithm 3.1 ("No Deficiency") into the basic parallel elimination algorithm of §2. The "while" loop in Algorithm 2.1 can now be replaced by the following:

while $G_i \neq \emptyset$ do begin

No Deficiency ($G_i$, $R_i$);

$G_{i+1} := G_i - R_i$;

$i := i + 1$
end

Unfortunately, this simple replacement does not make the time complexity of the overall parallel pivoting scheme linear. We need some more fine-tuning of the algorithm.

Recall in Algorithm 2.1, at step $i$, $G_i$ is the chordal graph under consideration. We use the notation $S_i$ to represent the set of all nodes with no deficiency in $G_i$. Since $R_i$ contains independent nodes, if $v \in R_i$, the adjacent nodes of $v$ can be excluded from $R_i$. 


The next observation can be used to further discard nodes for membership consideration in $R_i$. It says that we only need to look at the adjacent nodes of $R_{i-1}$ in the search for the next independent set $R_i$.

**THEOREM 4.1 [8].** For $i > 0$, $R_i \subseteq S_i \subseteq Adj(R_{i-1}) \cap G_i$.

**COROLLARY 4.2.** For $i > 0$, $|R_i| \leq |R_{i-1}|$.

**Proof.** Consider a node $x$ in $R_{i-1}$. Since $Adj(x) \cap G_i$ is a clique and nodes in $R_i$ are independent, by Theorem 4.1, at most one vertex from $Adj(x) \cap G_i$ can belong to $R_i$. Therefore, the number of nodes in $R_i$ cannot exceed that of $R_{i-1}$. \qed

Before we give the overall reordering algorithm, we first discuss the computation of $f_j$, a quantity required for the zero-deficiency test of node $x_j$. We need a fast way to compute and update $f_j$ for each $j$, since the value of $f_j$ may change as edges and nodes are being removed from the graph during the course of the algorithm. In order to overcome this problem, we presort each adjacency list in ascending order of the node indices according to the given perfect elimination ordering. All adjacency lists can be sorted in $O(n + e)$ time by a careful application of bucket sort. After having adjacency lists sorted, then for each node $x_j$ we can find the node $x_{f_j}$ in $O(1)$ time, since $x_{f_j}$ is either $x_j$ or the first node in the adjacency list of $x_j$, whichever has the smaller index. It should be added that in practice, the adjacency lists are often already in this desired ascending order (see, for example, [9]).

**ALGORITHM 4.1.** Parallel Elimination $(G)$ { $G$ chordal graph }

**begin**

determine a perfect elimination sequence $x_1, x_2, \ldots, x_n$ ;

$S_0 := \emptyset$ ;

for $j := 1$ to $n$ do begin

sort $Adj(x_j)$ in ascending order;

compute $deg(x_j)$ and $Mdeg(x_j)$ ;

$mark(x_j) := 0$ ;

if $deg(x_j) = Mdeg(x_{f_j})$ then add $x_j$ to $S_0$

end;

$i := 0$ ;

$G_0 := G$ ;

while $G_i \neq \emptyset$ do begin

$R_i := \emptyset$ ;

$S_{i+1} := \emptyset$ ;

for each $x_k$ of $S_i$ do begin

if $mark(x_k) = i$ then begin

add $x_k$ to $R_i$ ;

for each node $x_j \in Adj_G(x_k)$ in ascending order do begin

remove $x_k$ from $Adj_G(x_j)$ and update $f_j$ if necessary;

$deg(x_j) := deg(x_j) - 1$ ;

if $j < k$ then $Mdeg(x_j) := Mdeg(x_j) - 1$ ;

if $deg(x_j) = Mdeg(x_{f_j})$ and $mark(x_j) \leq i$ then add $x_j$ to $S_{i+1}$ and set $mark(x_j) := i + 1$

end

end

end

$G_{i+1} := G_i - R_i$ ;

$i := i + 1$

end**
At step \( i \) of the algorithm, the main “while-loop” determines the sets \( R_i \) and \( S_{i+1} \) simultaneously, using \( S_i \). It makes an implicit use of Theorem 4.1 in the determination of \( S_{i+1} \). The marker vector \( \text{mark}(\bullet) \) is employed in the algorithm to ensure the correct selection of nodes in \( R_i \) and \( S_{i+1} \).

During the execution of step \( i \), for each node \( x_j \) in \( G_i \), \( \text{mark}(x_j) \in \{0, i, i + 1\} \). \( \text{mark}(x_j) = i \) means \( x_j \in S_i - S_{i+1} \) (unless \( i = 0 \)); and \( \text{mark}(x_j) = i + 1 \) implies \( x_j \in S_{i+1} \). Furthermore, if \( x_j \) is not zero deficient in \( G_i \), then \( \text{mark}(x_j) = 0 \). A node \( x_k \) in \( S_i \) with \( \text{mark}(x_k) > i \) is not included into \( R_i \) since its marker value must be \( i + 1 \). This means \( x_k \) was adjacent to some node \( x_{k'} \) which was removed from the graph into \( R_i \) at an earlier time of step \( i \). At the end of this step, \( S_{i+1} \) contains all nodes with no deficiency in the (chordal) subgraph \( G_i - R_i \). Moreover, at the end of the algorithm, we have \( \text{mark}(x_j) = i \) if and only if \( x_j \in R_i \).

There are two places in the algorithm where the zero-deficiency test of the form \( \text{deg}(x_j) = \text{Mdeg}(x_j) \) is performed. It should be emphasized that at the point of testing, both \( \text{deg}(x_j) \) and \( \text{Mdeg}(x_j) \) have their correct values. The correctness of \( \text{deg}(x_j) \) is rather obvious. The correctness of \( \text{Mdeg}(x_j) \) follows from the fact that \( f_j \leq j \) and that the degree update “for-loop” goes through nodes in ascending order. This implies that \( \text{Mdeg}(x_j) \) has already been updated.

The time for each operation in the algorithm can be charged to either a node or an edge of the graph so that the total charge received by each node or edge is \( O(1) \). This guarantees that the time complexity of the algorithm is \( O(n + e) \). Indeed, it is clear that for each \( x_k \) in \( R_i \), the loop for degree update of nodes in \( \text{Adj}_G(x_k) \) and membership consideration of \( S_{i+1} \) can be done in time proportional to \( |\text{Adj}_G(x_k)| \). Summing over all nodes, we have it bounded by \( O(e) \). Next, consider the total time for membership selection of \( R_i \) from \( S_i \) for all \( i \). By Theorem 4.1, this total time is bounded by

\[
|S_0| + \sum_{i=1}^{m} |S_i| \leq n + \sum_{i=1}^{m} |\text{Adj}_G(R_{i-1})| \\
\leq n + \sum_{x} |\text{Adj}_G(x)| = O(n + e).
\]

Furthermore, since a perfect elimination ordering of a chordal graph can be obtained in linear time [6], [12], we have thus proved the following theorem.

**Theorem 4.3.** Given a chordal graph \( G \) with \( n \) nodes and \( e \) edges, Algorithm 4.1 correctly computes a parallel elimination ordering of \( G \) in \( O(n + e) \) time.

5. Concluding remarks. Algorithm 4.1 is a linear time reordering algorithm that generates a perfect ordering for a given chordal graph suitable for parallel elimination. Indeed, it may be regarded as an efficient implementation of the parallel pivoting strategy by Jess and Kees [8].

This reordering scheme can be used in the parallel factorization of sparse matrices (whose graphs are not necessarily chordal). In such a case, this algorithm should then be applied to the filled graph of the given sparse matrix. The resulting equivalent reordering will be appropriate for the original matrix in terms of both preserving sparsity and exploiting parallelism.

Acknowledgment. The authors thank Professor Alex Pothen for his comments.

REFERENCES


