SIAM J. MATRIX ANAL. APPL.
Vol. 20, No. 3, pp. 606–627
© 1999 Society for Industrial and Applied Mathematics

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

MODIFYING A SPARSE CHOLESKY FACTORIZATION*

TIMOTHY A. DAVIS† AND WILLIAM W. HAGER‡

Abstract. Given a sparse symmetric positive definite matrix $\mathbf{AA}^\mathsf{T}$ and an associated sparse Cholesky factorization $\mathbf{LDL}^\mathsf{T}$ or $\mathbf{LL}^\mathsf{T}$, we develop sparse techniques for obtaining the new factorization associated with either adding a column to $\mathbf{A}$ or deleting a column from $\mathbf{A}$. Our techniques are based on an analysis and manipulation of the underlying graph structure and on ideas of Gill et al. [*Math. Comp.*, 28 (1974), pp. 505–535] for modifying a dense Cholesky factorization. We show that our methods extend to the general case where an arbitrary sparse symmetric positive definite matrix is modified. Our methods are optimal in the sense that they take time proportional to the number of nonzero entries in $\mathbf{L}$ and $\mathbf{D}$ that change.

Key words. numerical linear algebra, direct methods, Cholesky factorization, sparse matrices, mathematical software, matrix updates

AMS subject classifications. 65F05, 65F50, 65-04

PII. S0895479897321076

1. Introduction.

This paper presents a method for updating and downdating the sparse Cholesky factorization $\mathbf{LDL}^\mathsf{T}$ or $\mathbf{LL}^\mathsf{T}$ of the matrix $\mathbf{AA}^\mathsf{T}$, where $\mathbf{A}$ is m-by-n. More precisely, we evaluate the Cholesky factorization of $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$, where either σ is $+1$ (corresponding to an update) and $\mathbf{w}$ is arbitrary or σ is -1 (corresponding to a downdate) and $\mathbf{w}$ is a column of $\mathbf{A}$. Both $\mathbf{AA}^\mathsf{T}$ and $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$ must be symmetric and positive definite. From this it follows that $m \le n$. The techniques we develop for the matrix $\mathbf{AA}^\mathsf{T}$ can be extended to determine the effects on the Cholesky factors of a general symmetric positive definite matrix $\mathbf{M}$ of any symmetric change of the form $\mathbf{M} + \sigma\mathbf{ww}^\mathsf{T}$ that preserves positive definiteness. The $\mathbf{AA}^\mathsf{T}$ case is simpler than the general case, which is why we discuss it first. Moreover, the techniques we develop for updating and downdating $\mathbf{AA}^\mathsf{T}$ are used in the algorithm for updating the general matrix $\mathbf{M}$. Our methods take into account the change in the sparsity pattern of $\mathbf{A}$ and $\mathbf{L}$ and are optimal in the sense that they take time proportional to the number of nonzero entries in $\mathbf{L}$ and $\mathbf{D}$ that change.

The importance of this problem has long been recognized [36], but prior sparse methods either are nonoptimal or do not consider changes to the sparsity pattern of $\mathbf{A}$ or $\mathbf{L}$. Both Law's sparse update method [27, 28] and the method of Chan and Brandwajn [6] are based on Bennett's method [3], which needs to be used with caution [20]. Law's symbolic update has nonoptimal asymptotic run time and can take more time than doing the symbolic factorization from scratch. The method of Row, Powell, and Mondkar [32] is for envelope-style factorization only and is also nonoptimal in both its numerical and its symbolic work. Neither of these approaches consider symbolic or numerical downdate. Chan and Brandwajn [6] consider the sparse numerical update and downdate, but with a fixed sparsity pattern.

*Received by the editors May 5, 1997; accepted for publication (in revised form) by S. Vavasis April 28, 1998; published electronically March 2, 1999.

http://www.siam.org/journals/simax/20-3/32107.html

†Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611-6120 (davis@cise.ufl.edu, http://www.cise.ufl.edu/∼davis). The work of this author was supported by National Science Foundation grant DMS-9504974.

‡Department of Mathematics, University of Florida, Gainesville, FL 32611-6120 (hager@math.ufl.edu, http://www.math.ufl.edu/∼hager). The work of this author was supported by National Science Foundation grant DMS-9404431.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

There are many applications of the techniques presented in this paper. In the linear program dual active set algorithm (LP DASA) [26], the $\mathbf{A}$ matrix corresponds to the basic variables in the current basis of the linear program, and in successive iterations, we bring variables in and out of the basis, leading to changes of the form $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$. Other application areas where the techniques developed in this paper are applicable include least-squares problems in statistics, the analysis of electrical circuits and power systems, structural mechanics, sensitivity analysis in linear programming, boundary condition changes in partial differential equations, domain decomposition methods, and boundary element methods. For a discussion of these application areas and others, see [25].

Section 2 introduces our notation. For an introduction to sparse matrix techniques, see [9, 13]. In section 3 we discuss the structure of the nonzero elements in the Cholesky factorization of $\mathbf{AA}^\mathsf{T}$, and in section 4 we discuss the structure associated with the Cholesky factors of $\mathbf{AA}^\mathsf{T} + \sigma\mathbf{ww}^\mathsf{T}$. The symbolic update and downdate methods provide the framework for our sparse version of Method C1 of Gill et al. [20] for modifying a dense Cholesky factorization. We discuss our sparse algorithm in section 5. Section 6 presents the general algorithm for modifying the sparse Cholesky factorization for *any* sparse symmetric positive definite matrix. A single update or downdate in the general case is more complicated and requires both a symbolic update and a symbolic downdate, based on the methods for $\mathbf{AA}^\mathsf{T}$ presented in section 4. The results of a numerical experiment with a large optimization problem from Netlib [8] are presented in section 7. Section 8 concludes with a discussion of future work.

2. Notation. Throughout the paper, matrices are capital bold letters like $\mathbf{A}$ or $\mathbf{L}$, while vectors are lower-case bold letters like $\mathbf{x}$ or $\mathbf{v}$. Sets and multisets are in calligraphic style like $\mathcal{A}$, $\mathcal{L}$, or $\mathcal{P}$. Scalars are either lower-case Greek letters or italic style like σ, k, or m.

Given the location of the nonzero elements of $\mathbf{AA}^\mathsf{T}$, we can perform a *symbolic factorization* (this terminology is introduced by George and Liu in [13]) of the matrix to predict the location of the nonzero elements of the Cholesky factor $\mathbf{L}$. In actuality, some of these predicted nonzeros may be zero due to numerical cancellation during the factorization process. The statement "$l_{ij} \neq 0$" will mean that l_{ij} is *symbolically* nonzero. The main diagonals of $\mathbf{L}$ and $\mathbf{D}$ are always nonzero since the matrices that we factor are positive definite (see [35, p. 253]). The nonzero pattern of column j of $\mathbf{L}$ is denoted $\mathcal{L}_j$,

$$\mathcal{L}_j = \{i : l_{ij} \neq 0\},$$

while $\mathcal{L}$ denotes the collection of patterns:

$$\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_m\}.$$

Similarly, $\mathcal{A}_j$ denotes the nonzero pattern of column j of $\mathbf{A}$,

$$\mathcal{A}_j = \{i : a_{ij} \neq 0\},$$

while $\mathcal{A}$ is the collection of patterns:

$$\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}.$$

The *elimination tree* can be defined in terms of a *parent map* π (see [29]). For any node j, $\pi(j)$ is the row index of the first nonzero element in column j of $\mathbf{L}$ beneath the diagonal element:

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

$$\pi(j) = \min\ \mathcal{L}_j \setminus \{j\},$$

where "min $\mathcal{X}$" denotes the smallest element of $\mathcal{X}$:

$$\min\ \mathcal{X} = \min_{i \in \mathcal{X}}\ i.$$

Our convention is that the min of the empty set is zero. Note that $j < \pi(j)$ except in the case where the diagonal element in column j is the only nonzero element. The inverse π^{-1} of the parent map is the *children multifunction*. That is, the children of node k are the set defined by

$$\pi^{-1}(k) = \{j : \pi(j) = k\}.$$

The *ancestors* of a node j, denoted $\mathcal{P}(j)$, are the set of successive parents:

$$\mathcal{P}(j) = \{j,\ \pi(j),\ \pi(\pi(j)), \ldots\} = \{\pi^0(j),\ \pi^1(j),\ \pi^2(j), \ldots\}.$$

Here the powers of a map are defined in the usual way: π^0 is the identity while π^i for $i > 0$ is the i-fold composition of π with itself. The sequence of nodes j, $\pi(j)$, $\pi(\pi(j))$, ..., forming $\mathcal{P}(k)$, is called the *path* from j to the associated tree *root*. The collection of paths leading to a root form an *elimination tree*. The set of all trees is the *elimination forest*. Typically, there is a single tree whose root is m; however, if column j of $\mathbf{L}$ has only one nonzero element, the diagonal element, then j will be the root of a separate tree.

The number of elements (or size) of a set $\mathcal{X}$ is denoted $|\mathcal{X}|$, while $|\mathcal{A}|$ or $|\mathcal{L}|$ denotes the sum of the sizes of the sets they contain. Let the complement of a set $\mathcal{X}$ be denoted as $\mathcal{X}^c = \{x : x \notin \mathcal{X}\}$.

3. Symbolic factorization.

Any approach for generating the pattern set $\mathcal{L}$ is called *symbolic factorization* [10, 11, 12, 13, 34]. The symbolic factorization of a matrix of the form $\mathbf{AA}^{\mathsf{T}}$ is given in Algorithm 1 (see [14, 29]).

ALGORITHM 1 (symbolic factorization of $\mathbf{AA}^{\mathsf{T}}$).

$\pi(j) = 0$ for each j

for $j = 1$ **to** m **do**

$$\mathcal{L}_j = \{j\} \cup \left(\bigcup_{c \in \pi^{-1}(j)} \mathcal{L}_c \setminus \{c\} \right) \cup \left(\bigcup_{\min\ \mathcal{A}_k = j} \mathcal{A}_k \right)$$

$\pi(j) = \min\ \mathcal{L}_j \setminus \{j\}$

end for

Algorithm 1 basically says that the pattern of column j of $\mathbf{L}$ can be expressed as the union of the patterns of each column of $\mathbf{L}$ whose parent is j and the patterns of the columns of $\mathbf{A}$ whose first nonzero element is j. The elimination tree, connecting each child to its parent, is easily formed during the symbolic factorization. Algorithm 1 can be done in $O(|\mathcal{L}| + |\mathcal{A}|)$ time[1] [14, 29].

Observe that the pattern of the parent of node j contains all entries in the pattern of column j except j itself [33]. That is,

$$\mathcal{L}_j \setminus \{j\} = \mathcal{L}_j \cap \{i : \pi(j) \le i\} \subseteq \mathcal{L}_{\pi(j)}.$$

[1] Asymptotic complexity notation is defined in [7]. We write $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n > n_0$.

Proceeding by induction, if k is an ancestor of j, then

$$\{i : i \in \mathcal{L}_j, k \leq i\} \subseteq \mathcal{L}_k. \tag{3.1}$$

This leads to the following relation between $\mathcal{L}_j$ and the path $\mathcal{P}(j)$. The first part of this proposition, and its proof, are given in [33]. Our proof differs slightly from the one in [33]. We include it here since the same proof technique is exploited later.

PROPOSITION 3.1. *For each j, we have $\mathcal{L}_j \subseteq \mathcal{P}(j)$; furthermore, for each k and $j \in \mathcal{P}(k)$, $\mathcal{L}_j \subseteq \mathcal{P}(k)$.*

Proof. Obviously, $j \in \mathcal{P}(j)$. Let i be any given element of $\mathcal{L}_j$ with $i \neq j$. Since $j < i$, we see that the following relation holds for $l = 0$:

$$\pi^0(j) < \pi^1(j) < \cdots < \pi^l(j) < i. \tag{3.2}$$

Now suppose that (3.2) holds for some integer $l \geq 0$, and let k denote $\pi^l(j)$. By (3.1) and the fact that $k < i$, we have $i \in \mathcal{L}_k$, which implies that

$$i \geq \pi(k) = \pi(\pi^l(j)) = \pi^{l+1}(j).$$

Hence, either $i = \pi^{l+1}(j)$ or (3.2) holds with l replaced by $l+1$. Since (3.2) is violated for l sufficiently large, we conclude that there exists an l for which $i = \pi^{l+1}(j)$. Consequently, $i \in \mathcal{P}(j)$. Since each element of $\mathcal{L}_j$ is contained in $\mathcal{P}(j)$, we have $\mathcal{L}_j \subseteq \mathcal{P}(j)$. If $j \in \mathcal{P}(k)$ for some k, then j is an ancestor of k and $\mathcal{P}(j) \subseteq \mathcal{P}(k)$. Since we have already shown that $\mathcal{L}_j \subseteq \mathcal{P}(j)$, the proof is complete. ◻

As we will see, the symbolic factorization of $\mathbf{AA}^\mathsf{T} + \mathbf{ww}^\mathsf{T}$ can be obtained by updating the symbolic factorization of $\mathbf{AA}^\mathsf{T}$ using an algorithm that has the same structure as that of Algorithm 1, except that it operates only on nodes in the path $\mathcal{P}(j)$ (of the updated factors) for some node j. The symbolic update algorithm adds new entries to the nonzero pattern, which can be done with a simple union operation.

However, downdating is not as easy as updating. Once a set union has been computed, it cannot be undone without knowledge of how entries entered the set. We can keep track of this information by storing the elements of $\mathcal{L}$ as multisets rather than as sets. The multiset associated with column j has the form

$$\mathcal{L}_j^\sharp = \{(i, m(i,j)) : i \in \mathcal{L}_j\},$$

where the multiplicity $m(i,j)$ is the number of children of j that contain row index i in their pattern plus the number of columns of $\mathcal{A}$ whose smallest entry is j and that contain row index i. Equivalently,

$$m(i,j) = |\{k \in \pi^{-1}(j) : i \in \mathcal{L}_k\}| + |\{k : \min\ \mathcal{A}_k = j \text{ and } i \in \mathcal{A}_k\}|.$$

With this definition, we can undo a set union by subtracting multiplicities.

We now define some operations involving multisets. First, if $\mathcal{X}^\sharp$ is a multiset consisting of pairs $(i, m(i))$ where $m(i)$ is the multiplicity associated with i, then $\mathcal{X}$ is the set obtained by removing the multiplicities. In other words, the multiset $\mathcal{X}^\sharp$ and the associated base set $\mathcal{X}$ satisfy the relation

$$\mathcal{X}^\sharp = \{(i, m(i)) : i \in \mathcal{X}\}.$$

We define the addition of a multiset $\mathcal{X}^\sharp$ and a set $\mathcal{Y}$ in the following way:

$$\mathcal{X}^\sharp + \mathcal{Y} = \{(i, m'(i)) \ : \ i \in \mathcal{X} \text{ or } i \in \mathcal{Y}\},$$

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

where

$$m'(i) = \begin{cases} 1 & \text{if } i \notin \mathcal{X} \text{ and } i \in \mathcal{Y}, \\ m(i) & \text{if } i \in \mathcal{X} \text{ and } i \notin \mathcal{Y}, \\ m(i)+1 & \text{if } i \in \mathcal{X} \text{ and } i \in \mathcal{Y}. \end{cases}$$

Similarly, the subtraction of a set $\mathcal{Y}$ from a multiset $\mathcal{X}^\sharp$ is defined by

$$\mathcal{X}^\sharp - \mathcal{Y} = \{(i, m'(i)) \, : \, i \in \mathcal{X} \text{ and } m'(i) > 0\},$$

where

$$m'(i) = \begin{cases} m(i) & \text{if } i \notin \mathcal{Y}, \\ m(i) - 1 & \text{if } i \in \mathcal{Y}. \end{cases}$$

The multiset subtraction of $\mathcal{Y}$ from $\mathcal{X}^\sharp$ cancels a prior addition. That is, for any multiset $\mathcal{X}^\sharp$ and any set $\mathcal{Y}$, we have

$$((\mathcal{X}^\sharp + \mathcal{Y}) - \mathcal{Y}) = \mathcal{X}^\sharp.$$

In contrast $((\mathcal{X} \cup \mathcal{Y}) \setminus \mathcal{Y})$ is equal to $\mathcal{X}$ if and only if $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets.

Algorithm 2 below performs a symbolic factorization of $\mathbf{AA}^\mathsf{T}$ with each set union operation replaced by a multiset addition. This algorithm is identical to Algorithm 1 except for the bookkeeping associated with multiplicities.

ALGORITHM 2 (symbolic factorization of $\mathbf{AA}^\mathsf{T}$ using multisets).

$\pi(j) = 0$ for each j
for $j = 1$ **to** m **do**
  $\mathcal{L}_j^\sharp = \{(j, 1)\}$
  for each $c \in \pi^{-1}(j)$ **do**
    $\mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + (\mathcal{L}_c \setminus \{c\})$
  end for
  for each k where min $\mathcal{A}_k = j$ **do**
    $\mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{A}_k$
  end for
  $\pi(j) = \min\ \mathcal{L}_j \setminus \{j\}$
end for

We conclude this section with a result concerning the relation between the patterns of $\mathbf{AA}^\mathsf{T}$ and the patterns of $\mathbf{AA}^\mathsf{T} + \mathbf{ww}^\mathsf{T}$.

PROPOSITION 3.2. *Let $\mathcal{C}$ and $\mathcal{D}$ be the patterns associated with the symmetric positive definite matrices $\mathbf{C}$ and $\mathbf{D}$, respectively. Neglecting numerical cancellation, $\mathcal{C}_j \subseteq \mathcal{D}_j$ for each j implies that $(\mathcal{L}_C)_j \subseteq (\mathcal{L}_D)_j$ for each j, where $\mathcal{L}_C$ and $\mathcal{L}_D$ are the patterns associated with the Cholesky factors of $\mathbf{C}$ and $\mathbf{D}$, respectively.*

Proof. In [13, 31] it is shown that an edge (i, j) is contained in the undirected graph of the Cholesky factor of a symmetric positive definite matrix $\mathbf{C}$ if and only if there is a path from i to j in the undirected graph of $\mathbf{C}$ with each intermediate vertex of the path between 1 and min $\{i, j\}$. If $\mathcal{C}_j \subseteq \mathcal{D}_j$ for each j, then the paths associated with the undirected graph of $\mathbf{C}$ are a subset of the paths associated with the undirected graph of $\mathbf{D}$. It follows that $(\mathcal{L}_C)_j \subseteq (\mathcal{L}_D)_j$ for each j. ▯

Ignoring numerical cancellation, the edges in the undirected graph of $\mathbf{AA}^\mathsf{T}$ are a subset of the edges in the undirected graph of $\mathbf{AA}^\mathsf{T} + \mathbf{ww}^\mathsf{T}$. By Proposition 3.2, we conclude that the edges in the undirected graphs of the associated Cholesky factors satisfy the same inclusion.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

4. Modifying the symbolic factors. Let $\overline{\mathbf{A}}$ be the modified version of $\mathbf{A}$. We put a bar over a matrix or a set or a multiset to denote its value after the update or downdate is complete. In an update, $\overline{\mathbf{A}}$ is obtained from $\mathbf{A}$ by appending the column $\mathbf{w}$ on the right, while in a downdate, $\overline{\mathbf{A}}$ is obtained from $\mathbf{A}$ by deleting the column $\mathbf{w}$ from $\mathbf{A}$. Hence, we have

$$\overline{\mathbf{A}}\,\overline{\mathbf{A}}^{\mathsf{T}} = \mathbf{A}\mathbf{A}^{\mathsf{T}} + \sigma \mathbf{w}\mathbf{w}^{\mathsf{T}},$$

where σ is either $+1$ and $\mathbf{w}$ is the last column of $\overline{\mathbf{A}}$ (update) or σ is -1 and $\mathbf{w}$ is a column of $\mathbf{A}$ (downdate). Since $\overline{\mathbf{A}}$ and $\mathbf{A}$ differ by at most a single column, it follows from Proposition 3.2 that $\mathcal{L}_j \subseteq \overline{\mathcal{L}}_j$ for each j during an update, while $\overline{\mathcal{L}}_j \subseteq \mathcal{L}_j$ during a downdate. Moreover, the multisets associated with the Cholesky factor of either the updated or downdated matrix have the structure described in the following theorem.

THEOREM 4.1. *Let k be the index associated with the first nonzero component of* $\mathbf{w}$. *For an update, $\mathcal{P}(k) \subseteq \overline{\mathcal{P}}(k)$. Moreover, $\overline{\mathcal{L}}_i^{\sharp} = \mathcal{L}_i^{\sharp}$ for all $i \in \overline{\mathcal{P}}(k)^c$. That is, $\overline{\mathcal{L}}_i^{\sharp} = \mathcal{L}_i^{\sharp}$ for all i except when i is k or one of the new ancestors of k. For a downdate, $\overline{\mathcal{P}}(k) \subseteq \mathcal{P}(k)$. Moreover, $\overline{\mathcal{L}}_i^{\sharp} = \mathcal{L}_i^{\sharp}$ for all $i \in \mathcal{P}(k)^c$. That is, $\overline{\mathcal{L}}_i^{\sharp} = \mathcal{L}_i^{\sharp}$ for all i except when i is k or one of the old ancestors of k.*

Proof. To begin, let us consider an update. We will show that each element of $\mathcal{P}(k)$ is a member of $\overline{\mathcal{P}}(k)$ as well. Clearly, k lies in both $\mathcal{P}(k)$ and $\overline{\mathcal{P}}(k)$. Proceeding by induction, suppose that

$$\pi^0(k), \pi^1(k), \pi^2(k), \ldots, \pi^j(k) \in \overline{\mathcal{P}}(k),$$

and define $l = \pi^j(k)$. We need to show that

$$\pi(l) = \pi(\pi^j(k)) = \pi^{j+1}(k) \in \overline{\mathcal{P}}(k)$$

to complete the induction. Since $l \in \overline{\mathcal{P}}(k)$, we have $l = \overline{\pi}^h(k)$ for some h. If $\overline{\pi}(l) = \pi(l)$, then

$$\overline{\pi}^{h+1}(k) = \overline{\pi}(\overline{\pi}^h(k)) = \overline{\pi}(l) = \pi(l) = \pi(\pi^j(k)) = \pi^{j+1}(k).$$

Since $\pi^{j+1}(k) = \overline{\pi}^{h+1}(k) \in \overline{\mathcal{P}}(k)$, the induction step is complete and $\pi^{j+1}(k) \in \overline{\mathcal{P}}(k)$.

If $\overline{\pi}(l) \neq \pi(l)$, then by Proposition 3.2, $\overline{\pi}(l) < \pi(l)$ and the following relation holds for $p = 1$:

$$\overline{\pi}^1(l) < \overline{\pi}^2(l) < \cdots < \overline{\pi}^p(l) < \pi(l). \tag{4.1}$$

Now suppose that (4.1) holds for some integer $p \geq 1$, and let q denote $\overline{\pi}^p(l)$. By Proposition 3.2, $\pi(l) \in \mathcal{L}_l \subseteq \overline{\mathcal{L}}_l$, and combining this with (4.1), $q = \overline{\pi}^p(l) < \pi(l) \in \overline{\mathcal{L}}_l$. It follows from (3.1) that $\pi(l) \in \overline{\mathcal{L}}_q$ for $q = \overline{\pi}^p(l)$. By the definition of the parent,

$$\overline{\pi}(q) = \overline{\pi}(\overline{\pi}^p(l)) = \overline{\pi}^{p+1}(l) \leq \pi(l) \in \overline{\mathcal{L}}_q.$$

Hence, either $\overline{\pi}^{p+1}(l) = \pi(l)$ or (4.1) holds with p replaced by $p+1$. Since (4.1) is violated for p sufficiently large, we conclude that there exists an integer p such that $\overline{\pi}^p(l) = \pi(l)$, from which it follows that

$$\pi^{j+1}(k) = \pi(\pi^j(k)) = \pi(l) = \overline{\pi}^p(l) = \overline{\pi}^p(\overline{\pi}^h(k)) = \overline{\pi}^{p+h}(k) \in \overline{\mathcal{P}}(k).$$

Since $\pi^{j+1}(k) \in \overline{\mathcal{P}}(k)$, the induction step is complete and $\mathcal{P}(k) \subseteq \overline{\mathcal{P}}(k)$.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Suppose that $l \in \overline{\mathcal{P}}(k)^c$. It is now important to recall that k is the index of the first nonzero component of $\mathbf{w}$, the vector appearing in the update. Observe that l cannot equal k since $l \in \overline{\mathcal{P}}(k)^c$ and $k \in \overline{\mathcal{P}}(k)$. The proof that $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$ is by induction on the height h defined by

$$h(l) = \max\{i : \pi^i(j) = l \text{ for some } j\}.$$

If $h(l) = 0$, then l has no children and the child loop of Algorithm 2 will be skipped when either $\overline{\mathcal{L}}_l^\sharp$ or $\mathcal{L}_l^\sharp$ are evaluated. And since $l \neq k$, the pattern associated with $\mathbf{w}$ cannot be added into $\overline{\mathcal{L}}_l^\sharp$. Hence, when $h(l) = 0$, the identity $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$ is trivial. Now, assuming that for some $p \geq 0$ we have $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$ whenever $l \in \overline{\mathcal{P}}(k)^c$ and $h(l) \leq p$, let us suppose that $h(l) = p + 1$. If $i \in \pi^{-1}(l)$, then $h(i) \leq p$. Hence, $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for $i \in \pi^{-1}(l)$ by the induction assumption. And since $l \neq k$, the pattern of $\mathbf{w}$ is not added to $\overline{\mathcal{L}}_l^\sharp$. Consequently, when Algorithm 2 is executed, we have $\overline{\mathcal{L}}_l^\sharp = \mathcal{L}_l^\sharp$, which completes the induction step.

Now consider the downdate part of the theorem. Rearranging the downdate relation $\overline{\mathbf{A}}\,\overline{\mathbf{A}}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T} - \mathbf{w}\mathbf{w}^\mathsf{T}$, we have

$$\mathbf{A}\mathbf{A}^\mathsf{T} = \overline{\mathbf{A}}\,\overline{\mathbf{A}}^\mathsf{T} + \mathbf{w}\mathbf{w}^\mathsf{T}.$$

Hence, in a downdate, we can think of $\mathbf{A}$ as the updated version of $\overline{\mathbf{A}}$. Consequently, the second part of the theorem follows directly from the first part. ∎

4.1. Symbolic update algorithm. We now present an algorithm for evaluating the new pattern $\overline{\mathcal{L}}$ associated with an update. Based on Theorem 4.1, the only sets $\overline{\mathcal{L}}_j$ that change are those associated with $\overline{\mathcal{P}}(k)$ where k is the index of the first nonzero component of $\mathbf{w}$. Referring to Algorithm 2, we can set $j = k$, $j = \overline{\pi}(k)$, $j = \overline{\pi}^2(k)$, and so on, marching up the path from k, and we can evaluate all the changes induced by the additional column in $\mathbf{A}$. In order to do the bookkeeping, there are at most four cases to consider:

Case 1: $j = k$. At the start of the new path, we need to add the pattern for $\mathbf{w}$ to $\mathcal{L}_j^\sharp$.

Case 2: $c \in \overline{\mathcal{P}}(k)$, $j \in \overline{\mathcal{P}}(k)$, $j > k$, and $c \in \overline{\pi}^{-1}(j) \cap \pi^{-1}(j)$. In this case, c is a child of j in both the new and the old elimination trees. Since the pattern $\overline{\mathcal{L}}_c$ may differ from $\mathcal{L}_c$, we need to add the difference to $\overline{\mathcal{L}}_j^\sharp$. Since j has a unique child on the path $\overline{\mathcal{P}}(k)$, there is at most one node c that satisfies these conditions. Also note that if

$$c \in \overline{\pi}^{-1}(j) \cap \pi^{-1}(j) \cap \overline{\mathcal{P}}(k)^c,$$

then by Theorem 4.1 $\overline{\mathcal{L}}_c = \mathcal{L}_c$ and hence this node c does not lead to an adjustment to $\overline{\mathcal{L}}_j^\sharp$ in Algorithm 2.

Case 3: $j \in \overline{\mathcal{P}}(k)$, $j > k$, and $c \in \overline{\pi}^{-1}(j) \setminus \pi^{-1}(j)$. In this case, c is a child of j in the new elimination tree, but not in the old tree, and the entire set $\overline{\mathcal{L}}_c$ should be added to $\overline{\mathcal{L}}_j^\sharp$ since it was not included in $\mathcal{L}_j^\sharp$. By Theorem 4.1, $\overline{\pi}(p) = \pi(p)$ for all $p \in \overline{\mathcal{P}}(k)^c$. Since $c \in \overline{\pi}^{-1}(j)$ but $c \notin \pi^{-1}(j)$, it follows that $\overline{\pi}(c) = j \neq \pi(c)$, and hence, $c \notin \overline{\mathcal{P}}(k)^c$ or, equivalently, $c \in \overline{\mathcal{P}}(k)$. Again, since each node on the path $\overline{\mathcal{P}}(k)$ from k has only one child on the path, there is at most one node c satisfying these conditions and it lies on the path $\overline{\mathcal{P}}(k)$.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Case 4: $j \in \overline{\mathcal{P}}(k)$, $j > k$, and $c \in \pi^{-1}(j) \setminus \overline{\pi}^{-1}(j)$. In this case, c is a child of j in the old elimination tree, but not in the new tree, and the set $\mathcal{L}_c$ should be subtracted from $\overline{\mathcal{L}}_j^\sharp$ since it was previously added to $\mathcal{L}_j^\sharp$. Since $\pi(p) = \overline{\pi}(p)$ for each $p \in \overline{\mathcal{P}}(k)^c$, the fact that $\pi(c) = j \neq \overline{\pi}(c)$ implies that $c \in \overline{\mathcal{P}}(k)$. In the algorithm that follows, we refer to nodes c that satisfy these conditions as *lost children.* A node j in the elimination tree can lose multiple children.

In each of the cases above, every node c that led to adjustments in the pattern was located on the path $\overline{\mathcal{P}}(k)$. To make these changes, Algorithm 3 (below) simply marches up the path $\overline{\mathcal{P}}(k)$ from k to the root making the adjustments enumerated in Cases 1 through 4 above. Consider a node c. If its parent changes, we have $\overline{\pi}(c) < \pi(c)$ by Proposition 3.2. Both the new parent $\overline{\pi}(c)$ and the old parent $\pi(c)$ are on the path $\overline{\mathcal{P}}(k)$. Node c is a new child of $\overline{\pi}(c)$ (Case 3), which is the next node in the path $\overline{\mathcal{P}}(k)$. Node c is a lost child of $\pi(c)$ (Case 4). That is, one node's new child is another node's lost child. If Algorithm 3 is at node $j = \overline{\pi}(c)$ and we notice a single new child c, we can place that node in a *lost-child-queue* for node $\pi(c)$ and process that queue when we come to the node $\pi(c)$ later in the path. We could instead modify node $\pi(c)$ the moment we find that it loses a child, but this could not be done in a simple left-to-right pass of the columns corresponding to the nodes in the path $\overline{\mathcal{P}}(k)$. As we will see in section 5, this will allow us to combine the symbolic and numeric algorithms into a single pass.

ALGORITHM 3 (symbolic update, add new column $\mathbf{w}$).

Case 1: *first node in the path*
$\mathcal{W} = \{i : w_i \neq 0\}$
$k = \min\ \mathcal{W}$
$\overline{\mathcal{L}}_k^\sharp = \mathcal{L}_k^\sharp + \mathcal{W}$
$\overline{\pi}(k) = \min\ \overline{\mathcal{L}}_k \setminus \{k\}$
$c = k$
$j = \overline{\pi}(c)$
while $j \neq 0$ **do**
  if $j = \pi(c)$ **then**
    Case 2: *c is an old child of j, possibly changed*
    $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \mathcal{L}_c)$
  else
    Case 3: *c is a new child of j and a lost child of* $\pi(c)$
    $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \{c\})$
    place c in lost-child-queue of $\pi(c)$
  end if
  Case 4: *consider each lost child of j*
  for each c in lost-child-queue of j **do**
    $\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp - (\mathcal{L}_c \setminus \{c\})$
  end for
  $\overline{\pi}(j) = \min\ \overline{\mathcal{L}}_j \setminus \{j\}$
  $c = j$
  $j = \overline{\pi}(c)$
end while
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp$ and $\overline{\pi}(j) = \pi(j)$ for all $j \in \overline{\mathcal{P}}(k)^c$
end Algorithm 3

The time taken by this algorithm is given by the following lemma.

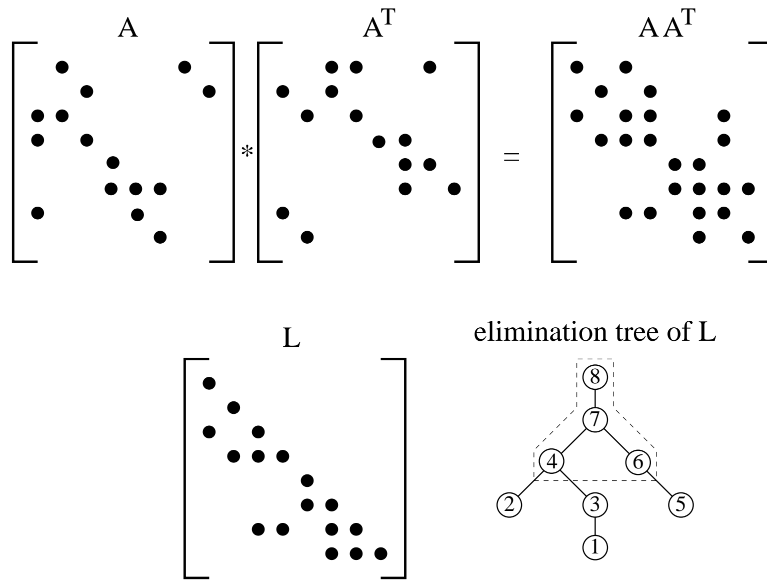

FIG. 4.1. *An example matrix, its factor, and its elimination tree.*

LEMMA 4.2. *The time to execute Algorithm* 3 *is bounded above by a constant times the number of entries associated with patterns for nodes on the new path* $\overline{\mathcal{P}}(k)$*. That is, the time is*

$$O\left(\sum_{j \in \overline{\mathcal{P}}(k)} |\overline{\mathcal{L}}_j|\right).$$

Proof. In Algorithm 3, we simply march up the path $\overline{\mathcal{P}}(k)$ making adjustments to $\overline{\mathcal{L}}_j^\sharp$ as we proceed. At each node j, preparing column j for all set subtractions or additions takes $O(|\mathcal{L}_j|)$ time (a scatter operation), and it takes $O(|\overline{\mathcal{L}}_j|)$ time to gather the results at the end of step j. For each child of j, the set subtraction/addition adds time proportional to the size of the subtracted/added set. Each node is visited as a child c at most twice, since it falls into one or two of the four cases enumerated above (a node c can be a new child of one node and a lost child of another). If this work (proportional to $|\mathcal{L}_c|$ or $|\overline{\mathcal{L}}_c|$) is accounted to step c instead of j, the time to make the adjustment to the pattern is bounded above by a constant times either $|\mathcal{L}_j|$ or $|\overline{\mathcal{L}}_j|$. Since $|\mathcal{L}_j| \leq |\overline{\mathcal{L}}_j|$ by Theorem 4.1, the proof is complete. ∎

In practice, we can reduce the execution time for Algorithm 3 by skipping over the current node j if it has no lost children and if its child c falls under Case 2 with $\overline{\mathcal{L}}_c = \mathcal{L}_c$. This check can be made in constant time, and if true, implies that $\mathcal{L}_j^\sharp = \overline{\mathcal{L}}_j^\sharp$.

We illustrate Algorithm 3 with an example. Figure 4.1 shows the nonzero pattern of an 8-by-8 matrix $\mathbf{A}$, the patterns of $\mathbf{A}\mathbf{A}^\mathsf{T}$ and its factor $\mathbf{L}$, and the elimination tree of $\mathbf{L}$. The highlighted portion of the elimination tree of $\mathbf{L}$ corresponds to the nodes in the path $\overline{\mathcal{P}}(k)$ if $\mathbf{A}\mathbf{A}^\mathsf{T}$ is updated with $\mathbf{w}\mathbf{w}^\mathsf{T}$, where the nonzero pattern of $\mathbf{w}$ is $\mathcal{W} = \{4, 6, 8\}$. The updated matrix $\overline{\mathbf{A}}\,\overline{\mathbf{A}}^\mathsf{T}$, its factor $\overline{\mathbf{L}}$, and the elimination tree of $\overline{\mathbf{L}}$ are shown in Figure 4.2. The new column $\mathbf{w}$ is appended to the original matrix $\mathbf{A}$.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

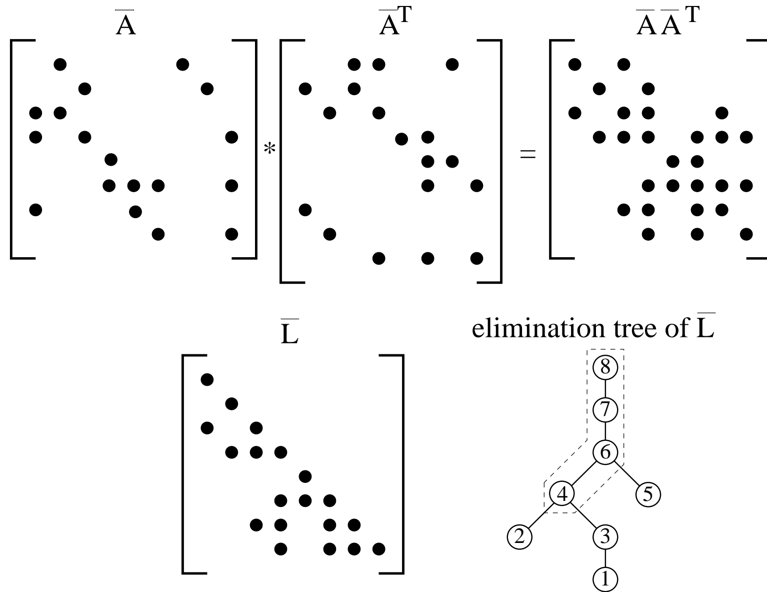

FIG. 4.2. *An example matrix after update.*

For this example, we have $\mathcal{W} = \{4, 6, 8\}$ and thus $k = 4$. The pattern of column 4 of $\overline{\mathbf{L}}$ falls under Case 1 and becomes $\overline{\mathcal{L}}_4 = \{4, 6, 7, 8\}$. The new parent of node 4 is node 6. At node 6, we find that $c = 4$ is a lost child of $\pi(4) = 7$ and a new child of node 6 (Case 3). The pattern of column 6 does not change, although its multiplicities do. The parent of node 6 is thus unchanged (node 7). Node 4 is placed in column 7's lost-child-queue. Node 6's lost-child-queue is empty. At column 7, we find that $c = 6$ is the old child of node 7 (Case 2), and node 4 is found in node 7's lost-child-queue (Case 4). This changes the multiplicities of column 7, but not its pattern. Finally, at node 8, nothing changes.

4.2. Symbolic downdate algorithm. Let us consider the removal of a column $\mathbf{w}$ from $\mathbf{A}$, and let k be the index of the first nonzero entry in $\mathbf{w}$. The symbolic downdate algorithm is analogous to the symbolic update algorithm, but the roles of $\mathcal{P}(k)$ and $\overline{\mathcal{P}}(k)$ are interchanged in accordance with Theorem 4.1. Instead of adding entries to $\mathcal{L}_j^\sharp$, we subtract entries; instead of lost-child-queues, we have new-child-queues; instead of walking up the path $\overline{\mathcal{P}}(k)$, we walk up the path $\mathcal{P}(k) \supseteq \overline{\mathcal{P}}(k)$.

ALGORITHM 4 (symbolic downdate, remove column $\mathbf{w}$).

Case 1: *first node in the path*
$\mathcal{W} = \{i : w_i \neq 0\}$
$k = \min\ \mathcal{W}$
$\overline{\mathcal{L}}_k^\sharp = \mathcal{L}_k^\sharp - \mathcal{W}$
$\overline{\pi}(k) = \min\ \overline{\mathcal{L}}_k \setminus \{k\}$
$c = k$
$j = \pi(c)$
while $j \neq 0$ **do**
  if $j = \overline{\pi}(c)$ **then**

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Case 2: *c is an old child of j, possibly changed*
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp - (\mathcal{L}_c \setminus \overline{\mathcal{L}}_c)$
else
Case 3: *c is a lost child of j and a new child of* $\overline{\pi}(c)$
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp - (\mathcal{L}_c \setminus \{c\})$
place c in new-child-queue of $\overline{\pi}(c)$
end if
Case 4: *consider each new child of j*
for each c in new-child-queue of j **do**
$\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \{c\})$
end for
$\overline{\pi}(j) = \min\ \overline{\mathcal{L}}_j \setminus \{j\}$
$c = j$
$j = \pi(c)$
end while
$\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp$ and $\overline{\pi}(j) = \pi(j)$ for all $j \in \mathcal{P}(k)^c$
end Algorithm 4

Similar to Algorithm 3, the execution time obeys the following estimate.

LEMMA 4.3. *The time to execute Algorithm 4 is bounded above by a constant times the number of entries associated with patterns for nodes on the old path $\mathcal{P}(k)$. That is, the time is*

$$O\left(\sum_{j\in\mathcal{P}(k)} |\mathcal{L}_j|\right).$$

We can skip over some nodes that do not change in a similar manner as Algorithm 3. Figures 4.1 and 4.2 can be viewed as an example of symbolic downdate, $\overline{\mathbf{A}}\,\overline{\mathbf{A}}^\mathsf{T} - \mathbf{w}\mathbf{w}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T}$, where $\mathbf{w}$ is the ninth column of $\overline{\mathbf{A}}$. The roles of $\mathbf{A}$ and $\overline{\mathbf{A}}$ are reversed.

5. The numerical factors.

When we add or delete a column in $\mathbf{A}$, we update or downdate the symbolic factorization in order to determine the location in the Cholesky factor of either new nonzero entries or old nonzero entries that are now zero. Knowing the location of the nonzero entries, we can update the numerical value of these entries. We first consider the case when $\mathbf{A}$ and $\mathbf{L}$ are dense and draw on the ideas of [20]. Then we show how the method extends to the sparse case.

5.1. Modifying a dense factorization.

Our algorithm to implement the numerical update and downdate is based on a modification of Method C1 in [20] for dense matrices. Although this algorithm is portrayed as a nonorthogonal algorithm in [20], it is equivalent, after a diagonal scaling, to the orthogonal algorithm of Pan [4, 30]. Hence, Method C1 should possess strong numerical stability properties comparable to those of Pan's method. Other dense update or downdate methods include [2, 5, 21, 22].

To summarize Gill et al.'s approach, we can write

$$\overline{\mathbf{A}}\,\overline{\mathbf{A}}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T} = \mathbf{L}\mathbf{D}\mathbf{L}^\mathsf{T} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T} = \mathbf{L}(\mathbf{D} + \sigma\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{L}^\mathsf{T} = \mathbf{L}(\tilde{\mathbf{L}}\tilde{\mathbf{D}}\tilde{\mathbf{L}}^\mathsf{T})\mathbf{L}^\mathsf{T} = \overline{\mathbf{L}}\,\overline{\mathbf{D}}\,\overline{\mathbf{L}}^\mathsf{T},$$

where $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$, $\overline{\mathbf{L}} = \mathbf{L}\tilde{\mathbf{L}}$, and $\overline{\mathbf{D}} = \tilde{\mathbf{D}}$. In Algorithm 5 below, we evaluate the new Cholesky factor $\overline{\mathbf{L}} = \mathbf{L}\tilde{\mathbf{L}}$ without forming $\tilde{\mathbf{L}}$ explicitly [20] by taking advantage of the

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

special structure of $\tilde{\mathbf{L}}$. The product is computed column by column, moving from left to right. In practice, $\mathbf{L}$ can be overwritten with $\overline{\mathbf{L}}$.

ALGORITHM 5 (dense numeric update/downdate; Method C1, modified).

$\alpha = 1$
for $j = 1$ to m **do**
  $\overline{\alpha} = \alpha + \sigma w_j^2/d_j$
  $\gamma = w_j/(\overline{\alpha} d_j)$
  $\overline{d}_j = (\overline{\alpha}/\alpha) d_j$
  $\alpha = \overline{\alpha}$
  $\mathbf{w}_{j+1,\ldots,m} = \mathbf{w}_{j+1,\ldots,m} - w_j \mathbf{L}_{j+1,\ldots,m,j}$
  $\overline{\mathbf{L}}_{j+1,\ldots,m,j} = \mathbf{L}_{j+1,\ldots,m,j} + \sigma\gamma \mathbf{w}_{j+1,\ldots,m}$
end for

Note that Algorithm 5 also overwrites $\mathbf{w}$ with $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$. In the jth iteration, w_j is simply equal to v_j. This observation is important to the sparse case discussed in the next section. Since σ is ± 1, the floating point operation count of Algorithm 5 is precisely $2m^2 + 5m$, counting multiplications, divisions, subtractions, and additions separately. As noted above, if we introduce a diagonal scaling in Algorithm 5, we obtain Pan's method [4, 30] for modifying the sparse $\mathbf{LL}^\mathsf{T}$ factorization, for which the corresponding operation count is $2.5m^2 + 6.5m$ plus an additional m square roots. Whether we use Algorithm 5 or Pan's algorithm to update the numerical factors, the symbolic algorithms are unchanged.

5.2. Modifying a sparse factorization. In the sparse case, $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ is sparse and its nonzero pattern is crucial. In Algorithm 5, we can essentially by-pass those executable statements associated with values of j for which the variable w_j vanishes. That is, when w_j vanishes, the values of α, d_j, $\mathbf{w}$, and column j of $\mathbf{L}$ are unchanged. Since w_j has been overwritten by v_j, it follows that when executing Algorithm 5, column j of $\mathbf{L}$ changes only when v_j does not vanish. The nonzero pattern of $\mathbf{v}$ can be found using the following lemma. The lemma is based on the directed graph $G(\mathbf{L}^\mathsf{T}) = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{1, 2, \ldots, m\}$ is the vertex set and $\mathcal{E} = \{(j, i) \mid i \in \mathcal{L}_j\}$ is the directed edge set.[2]

LEMMA 5.1. *The nodes reachable from any given node k by path(s) in the directed graph $G(\mathbf{L}^\mathsf{T})$ coincide with the path $\mathcal{P}(k)$.*

Proof. If $\mathcal{P}(k)$ has a single element, the lemma holds. Proceeding by induction, suppose that the lemma holds for all k for which $|\mathcal{P}(k)| \le j$. Now, if $\mathcal{P}(k)$ has $j + 1$ elements, then by the induction hypothesis, the nodes reachable from $\pi(k)$ by path(s) in the directed graph $G(\mathbf{L}^\mathsf{T})$ coincide with the path $\mathcal{P}(\pi(k))$. The nodes reachable in one step from k consist of the elements of $\mathcal{L}_k$. By Proposition 3.1, each of the elements of $\mathcal{L}_k$ is contained in the path $\mathcal{P}(k)$. If $i \in \mathcal{L}_k$, $i \ne k$, then $|\mathcal{P}(i)| \le j$. By the induction hypothesis, the nodes reachable from i coincide with $\mathcal{P}(i) \subseteq \mathcal{P}(k)$. The nodes reachable from k consist of the union of $\{k\}$ with the nodes reachable from $\mathcal{L}_k$. Since $k \in \mathcal{P}(k)$, it follows that the nodes reachable from k are contained in $\mathcal{P}(k)$. On the other hand, for each p, the element of $\mathbf{L}^\mathsf{T}$ in row $\pi^p(k)$ and column $\pi^{p+1}(k)$ is nonzero. Hence, all the elements of $\mathcal{P}(k)$ are reachable from k. Since the nodes in $\mathcal{P}(k)$ coincide with the nodes reachable from k by path(s) in the directed graph $G(\mathbf{L}^\mathsf{T})$, the induction step is complete. ∎

[2] Note that this definition of $G(\mathbf{L}^\mathsf{T})$ includes self-loops corresponding to the diagonal entries of $\mathbf{L}$.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

THEOREM 5.2. *During symbolic downdate $\overline{\mathbf{A}}\,\overline{\mathbf{A}}^{\mathsf{T}} = \mathbf{A}\mathbf{A}^{\mathsf{T}} - \mathbf{w}\mathbf{w}^{\mathsf{T}}$ (where $\mathbf{w}$ is a column of $\mathbf{A}$), the nonzero pattern of $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ is equal to the path $\mathcal{P}(k)$ in the (old) elimination tree of $\mathbf{L}$ where*

$$k = \min \{i : w_i \neq 0\}. \tag{5.1}$$

Proof. Let $\mathcal{W} = \{i : w_i \neq 0\}$. Theorem 5.1 of Gilbert [15, 16, 19] states that the nonzero pattern of $\mathbf{v}$ is the set of nodes reachable from the nodes in $\mathcal{W}$ by paths in the directed graph $G(\mathbf{L}^{\mathsf{T}})$. By Algorithm 1, $\mathcal{W} \subseteq \mathcal{L}_k$. Hence, each element of $\mathcal{W}$ is reachable from k by a path of length one, and the nodes reachable from $\mathcal{W}$ are a subset of the nodes reachable from k. Conversely, since $k \in \mathcal{W}$, the nodes reachable from k are a subset of the nodes reachable from $\mathcal{W}$. Combining these inclusions, the nodes reachable from k and from $\mathcal{W}$ are the same, and by Lemma 5.1, the nodes reachable from k coincide with the path $\mathcal{P}(k)$. ∎

COROLLARY 5.3. *During symbolic update $\overline{\mathbf{A}}\,\overline{\mathbf{A}}^{\mathsf{T}} = \mathbf{A}\mathbf{A}^{\mathsf{T}} + \mathbf{w}\mathbf{w}^{\mathsf{T}}$, the nonzero pattern of $\mathbf{v} = \mathbf{L}^{-1}\mathbf{w}$ is equal to the path $\overline{\mathcal{P}}(k)$ in the (new) elimination tree of $\overline{\mathbf{L}}$ where k is defined in* (5.1).

Proof. Since $\mathbf{L}\mathbf{D}\mathbf{L}^{\mathsf{T}} = \overline{\mathbf{A}}\,\overline{\mathbf{A}}^{\mathsf{T}} - \mathbf{w}\mathbf{w}^{\mathsf{T}}$, we can view $\mathbf{L}$ as the Cholesky factor for the downdate $\overline{\mathbf{A}}\,\overline{\mathbf{A}}^{\mathsf{T}} - \mathbf{w}\mathbf{w}^{\mathsf{T}}$. Hence, we can apply Theorem 5.2, in effect replacing $\mathcal{P}$ by $\overline{\mathcal{P}}$. ∎

As a result of Theorem 5.2, a sparse downdate algorithm can skip over any column $j \in \mathcal{P}(k)^c$. Similarly, as a result of Corollary 5.3, a sparse update algorithm can skip over any column $j \in \overline{\mathcal{P}}(k)^c$. In both cases, the jth iteration of Algorithm 5 requires both the old and new nonzero patterns of the jth column of $\mathbf{L}$ (that is, $\mathcal{L}_j$ and $\overline{\mathcal{L}}_j$). These are computed in the symbolic update and downdate Algorithms 3 and 4. Finally, note that the symbolic and numeric algorithms have the same structure. They both iterate over the columns in the associated path. Therefore, Algorithms 3 and 5 can be combined to obtain a complete sparse update algorithm that makes just one pass of the matrix $\mathbf{L}$. The jth iteration of Algorithm 5 is placed just before the $j = \overline{\pi}(c)$ statement in Algorithm 3. Since $\mathcal{L}_j \subseteq \overline{\mathcal{L}}_j$ during a sparse update, the total time taken for Algorithms 3 and 5 is

$$O\left(\sum_{j \in \overline{\mathcal{P}}(k)} |\overline{\mathcal{L}}_j| \right).$$

Similarly, Algorithms 4 and 5 can be combined to obtain a complete sparse downdate algorithm. Since $\overline{\mathcal{L}}_j \subseteq \mathcal{L}_j$ during a sparse downdate, the time taken for Algorithms 4 and 5 is

$$O\left(\sum_{j \in \mathcal{P}(k)} |\mathcal{L}_j| \right).$$

This time can be much less than the $O(m^2)$ time taken by Algorithm 5 in the dense case.

6. Arbitrary symbolic and numerical factors.

The methods we have developed for computing the modification to the Cholesky factors of $\mathbf{A}\mathbf{A}^{\mathsf{T}}$ corresponding to the addition or deletion of columns in $\mathbf{A}$ can be used to determine the effect on

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

the Cholesky factors of a general symmetric positive definite matrix $\mathbf{M}$ of any symmetric change of the form $\mathbf{M}+\sigma\mathbf{w}\mathbf{w}^\mathsf{T}$ that preserves positive definiteness. We briefly describe how Algorithms 1 through 5 are modified for the general case.

Let $\mathcal{M}_j$ denote the nonzero pattern of the *lower triangular* part of $\mathbf{M}$:

$$\mathcal{M}_j = \{i : m_{ij} \neq 0 \text{ and } i \geq j\}.$$

The symbolic factorization of $\mathbf{M}$ [10, 11, 12, 13, 34] is obtained by replacing the union of $\mathcal{A}_k$ terms in Algorithm 1 with the set $\mathcal{M}_j$. With this change, $\mathcal{L}_j$ of Algorithm 1 is given by

$$\mathcal{L}_j = \{j\} \cup \left(\bigcup_{c \in \pi^{-1}(j)} \mathcal{L}_c \setminus \{c\} \right) \cup \mathcal{M}_j.$$

This leads to a change in Algorithm 2 for computing the multiplicities. The multiplicity of an index i in $\mathcal{L}_j$ becomes

$$m(i,j) = |\{k \in \pi^{-1}(j) : i \in \mathcal{L}_k\}| + (1 \text{ if } i \in \mathcal{M}_j, \text{ or } 0 \text{ otherwise}).$$

The loop involving the $\mathcal{A}_k$ terms in Algorithm 2 is replaced by the single statement $\mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{M}_j$. More precisely, we have

$$\left.\begin{array}{l} \textbf{for } \text{each } k \text{ where } \min \mathcal{A}_k = j \textbf{ do} \\ \qquad \mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{A}_k \\ \textbf{end for} \end{array}\right\} \quad \Rightarrow \quad \mathcal{L}_j^\sharp = \mathcal{L}_j^\sharp + \mathcal{M}_j.$$

Entries are removed or added symbolically from $\mathbf{A}\mathbf{A}^\mathsf{T}$ by the deletion or addition of columns of $\mathbf{A}$, and numerical cancellation is ignored. Numerical cancellation of entries in $\mathbf{M}$ should not be ignored, however, because this is the only way that entries can be dropped from $\mathbf{M}$. When numerical cancellation is taken into account, neither of the inclusions $\mathcal{M}_j \subseteq \overline{\mathcal{M}}_j$ nor $\overline{\mathcal{M}}_j \subseteq \mathcal{M}_j$ may hold. We resolve this problem by using a symbolic modification scheme with two steps: a symbolic update phase in which new nonzero entries in $\mathbf{M}+\sigma\mathbf{w}\mathbf{w}^\mathsf{T}$ are taken into account, followed by a separate symbolic downdate phase to handle entries that become numerically zero. Since each modification step now involves an update phase followed by a downdate phase, we attach (in this section) an overbar to quantities associated with the update and an underbar to quantities associated with the downdate.

Let $\mathcal{W}$ be the nonzero pattern of $\mathbf{w}$, namely, $\mathcal{W} = \{i : w_i \neq 0\}$. In the first symbolic phase, entries from $\mathcal{W}$ are symbolically added to $\mathcal{M}_j$ for each $j \in \mathcal{W}$. That is, if $i \notin \mathcal{M}_j$, but $i, j \in \mathcal{W}$ with $i > j$, then we add i to $\mathcal{M}_j$:

$$\overline{\mathcal{M}}_j = \mathcal{M}_j \cup \{i \in \mathcal{W} : i > j\}.$$

In the second symbolic phase, entries from $\mathcal{W}$ are symbolically deleted for each $j \in \mathcal{W}$:

$$\underline{\mathcal{M}}_j = \overline{\mathcal{M}}_j \setminus \{i \in \mathcal{W} : i > j,\ \ m_{ij} + \sigma w_i w_j = 0\}. \tag{6.1}$$

In practice, we need to introduce a drop tolerance t and replace the equality

$$m_{ij} + \sigma w_i w_j = 0$$

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

in (6.1) by the inequality $|m_{ij} + \sigma w_i w_j| \le t$. For a general matrix, the analogue of Theorem 4.1 is the following.

THEOREM 6.1. *If α is the first index for which $\mathcal{M}_\alpha \neq \overline{\mathcal{M}}_\alpha$, then $\mathcal{P}(\alpha) \subseteq \overline{\mathcal{P}}(\alpha)$. Moreover, $\overline{\mathcal{L}}_i^\sharp = \mathcal{L}_i^\sharp$ for all $i \in \overline{\mathcal{P}}(\alpha)^c$. If β is the first index for which $\overline{\mathcal{M}}_\beta \neq \underline{\mathcal{M}}_\beta$, then $\underline{\mathcal{P}}(\beta) \subseteq \overline{\mathcal{P}}(\beta)$. Moreover, $\underline{\mathcal{L}}_i^\sharp = \overline{\mathcal{L}}_i^\sharp$ for all $i \in \overline{\mathcal{P}}(\beta)^c$.*

In evaluating the modification in the symbolic factorization associated with $\mathbf{M} + \sigma\mathbf{w}\mathbf{w}^\mathsf{T}$, we start at the first index α where $\mathcal{M}_\alpha \neq \overline{\mathcal{M}}_\alpha$ and we march up the path $\overline{\mathcal{P}}(\alpha)$ making changes to $\mathcal{L}_j^\sharp$, obtaining $\overline{\mathcal{L}}_j^\sharp$. In the second phase, we start at the first index where $\overline{\mathcal{M}}_\beta \neq \underline{\mathcal{M}}_\beta$ and we march up the path $\overline{\mathcal{P}}(\beta)$ making changes to $\overline{\mathcal{L}}_j^\sharp$, obtaining $\underline{\mathcal{L}}_j^\sharp$. The analogue of Algorithm 3 in the general case differs only in the starting index (now α) and in the addition of the sets $\overline{\mathcal{M}}_j \setminus \mathcal{M}_j$ in each pass through the j-loop.

ALGORITHM 6a (symbolic update phase, general matrix).

> *Case* 1: *first node in the path*
> $\alpha = \min\,\{i : \mathcal{M}_i \neq \overline{\mathcal{M}}_i\}$
> $\overline{\mathcal{L}}_\alpha^\sharp = \mathcal{L}_\alpha^\sharp + \overline{\mathcal{M}}_\alpha \setminus \mathcal{M}_\alpha$
> $\overline{\pi}(\alpha) = \min\ \overline{\mathcal{L}}_\alpha \setminus \{\alpha\}$
> $c = \alpha$
> $j = \overline{\pi}(c)$
> **while** $j \neq 0$ **do**
> $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp + \overline{\mathcal{M}}_j \setminus \mathcal{M}_j$
>   **if** $j = \pi(c)$ **then**
>   *Case* 2: *c is an old child of j, possibly changed*
> $\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \mathcal{L}_c)$
>   **else**
>   *Case* 3: *c is a new child of j and a lost child of* $\pi(c)$
> $\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp + (\overline{\mathcal{L}}_c \setminus \{c\})$
>   place c in lost-child-queue of $\pi(c)$
>   **end if**
>   *Case* 4: *consider each lost child of j*
>   **for** each c in lost-child-queue of j **do**
> $\overline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp - (\mathcal{L}_c \setminus \{c\})$
>   **end for**
> $\overline{\pi}(j) = \min\ \overline{\mathcal{L}}_j \setminus \{j\}$
> $c = j$
> $j = \overline{\pi}(c)$
> **end while**
> $\overline{\mathcal{L}}_j^\sharp = \mathcal{L}_j^\sharp$ and $\overline{\pi}(j) = \pi(j)$ for all $j \in \overline{\mathcal{P}}(\alpha)^c$
>
> **end Algorithm 6a**

Similarly, the analogue of Algorithm 4 in the general case differs only in the starting index (now β) and in the subtraction of the sets $\overline{\mathcal{M}}_j \setminus \underline{\mathcal{M}}_j$ in each pass through the j-loop.

ALGORITHM 6b (symbolic downdate phase, general matrix).

> *Case* 1: *first node in the path*
> $\beta = \min\,\{i : \overline{\mathcal{M}}_i \neq \underline{\mathcal{M}}_i\}$
> $\underline{\mathcal{L}}_\beta^\sharp = \overline{\mathcal{L}}_\beta^\sharp - \overline{\mathcal{M}}_\beta \setminus \underline{\mathcal{M}}_\beta$
> $\underline{\pi}(\beta) = \min\ \underline{\mathcal{L}}_\beta \setminus \{\beta\}$

$c = \beta$
$j = \overline{\pi}(c)$
while $j \neq 0$ **do**
$\underline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp - \overline{\mathcal{M}}_j \setminus \underline{\mathcal{M}}_j$
if $j = \underline{\pi}(c)$ **then**
Case 2: *c is an old child of j, possibly changed*
$\underline{\mathcal{L}}_j^\sharp = \underline{\mathcal{L}}_j^\sharp - (\overline{\mathcal{L}}_c \setminus \underline{\mathcal{L}}_c)$
else
Case 3: *c is a lost child of j and a new child of* $\underline{\pi}(c)$
$\underline{\mathcal{L}}_j^\sharp = \underline{\mathcal{L}}_j^\sharp - (\overline{\mathcal{L}}_c \setminus \{c\})$
place c in new-child-queue of $\underline{\pi}(c)$
end if
Case 4: *consider each new child of j*
for each c in new-child-queue of j **do**
$\underline{\mathcal{L}}_j^\sharp = \underline{\mathcal{L}}_j^\sharp + (\underline{\mathcal{L}}_c \setminus \{c\})$
end for
$\underline{\pi}(j) = \min\ \underline{\mathcal{L}}_j \setminus \{j\}$
$c = j$
$j = \overline{\pi}(c)$
end while
$\underline{\mathcal{L}}_j^\sharp = \overline{\mathcal{L}}_j^\sharp$ and $\underline{\pi}(j) = \overline{\pi}(j)$ for all $j \in \overline{\mathcal{P}}(\beta)^c$
end Algorithm 6b

Algorithm 5 is completely unchanged in the general case. It can be applied after the completion of Algorithm 6b so that we know the location of new nonzero entries in the Cholesky factor. It processes the submatrix associated with rows and columns in $\overline{\mathcal{P}}(k)$, where k is the index of the first nonzero element of $\mathbf{w}$. When $\mathbf{M}$ has the form $\mathbf{AA}^\mathsf{T}$ and when $\overline{\mathbf{M}}$ is found by either adding or deleting a column in $\mathbf{A}$, then assuming no numerical cancellations, Algorithm 6b can be skipped when we add a column to $\mathbf{A}$ since $\overline{\mathcal{M}}_j = \underline{\mathcal{M}}_j$ for each j. Similarly, when a column is removed from $\mathbf{A}$, Algorithm 6a can be skipped since $\overline{\mathcal{M}}_j = \mathcal{M}_j$ for each j. Hence, when Algorithm 6a followed by Algorithm 6b is applied to a matrix of the form $\mathbf{AA}^\mathsf{T}$, only Algorithm 6a takes effect during an update, while only Algorithm 6b takes effect during a downdate. Thus the approach we have presented in this section for an arbitrary symmetric positive definite matrix generalizes the earlier approach where we focus on matrices of the form $\mathbf{AA}^\mathsf{T}$.

7. Experimental results.

We have developed Matlab codes to experiment with all the algorithms presented in this paper, including the algorithms of section 6 for a general symmetric, positive definite matrix. In this section, we present the results of a numerical experiment with a large sparse optimization problem from Netlib [8] in the context of the LP DASA [26]. The computer used for this experiment was a Model 170 UltraSparc, equipped with 256MB of memory and with Matlab Version 4.2c.

7.1. Experimental design.

In the LP DASA, the columns of the matrix $\mathbf{A}$ in the product $\mathbf{AA}^\mathsf{T}$ are all chosen from among the columns of some fixed matrix $\mathbf{B}$. After a few updates or downdates, the system $\mathbf{AA}^\mathsf{T}\mathbf{x} = \mathbf{b}$ must be solved with a dense right-hand side $\mathbf{b}$.

The Cholesky factorization of the initial $\mathbf{AA}^\mathsf{T}$ (before any columns are added or deleted) is often preceded by a fill-reducing permutation $\mathbf{P}$ of the rows and columns

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

([1], for example). We can compute a permutation to reduce the fill for $\mathbf{B}\mathbf{B}^\mathsf{T}$ since the Cholesky factors of $\mathbf{A}\mathbf{A}^\mathsf{T}$ will be at least as sparse as those of $\mathbf{B}\mathbf{B}^\mathsf{T}$ by Proposition 3.2, regardless of how the columns of $\mathbf{A}$ are chosen from the columns of $\mathbf{B}$. Based on the number of nonzeros in each column of the Cholesky factors of $\mathbf{B}\mathbf{B}^\mathsf{T}$, we allocate a static storage structure that will always contain the Cholesky factors of each $\mathbf{A}\mathbf{A}^\mathsf{T}$. This can lead to wasted space if the number of nonzeros in the Cholesky factors of $\mathbf{A}\mathbf{A}^\mathsf{T}$ is far less than the number of nonzeros in the Cholesky factors of $\mathbf{B}\mathbf{B}^\mathsf{T}$. Alternatively, we could store the Cholesky factor of the current $\mathbf{A}\mathbf{A}^\mathsf{T}$ in a smaller space and reallocate storage during the updates and downdates, based on the changes in the nonzero patterns.

We selected an optimization problem from airline scheduling (DFL001). Its constraint matrix $\mathbf{B}$ is 6,071-by-12,230 with 35,632 nonzeros. We rescaled the variables so that the columns of $\mathbf{B}$ are all unit vectors, and we augmented $\mathbf{B}$ by appending on its right the matrix $10^{-6}\mathbf{I}$. This ensures that $\mathbf{B}\mathbf{B}^\mathsf{T}$ is strictly positive definite. The matrix $\mathbf{B}\mathbf{B}^\mathsf{T}$ has 37,923 nonzeros in its strictly lower triangular part. The Cholesky factor $\mathbf{L}_B$ has 1.49 million nonzeros (with a fill-minimizing permutation $\mathbf{P}_B$ of the rows of $\mathbf{B}$, described below) and requires 1.12 billion floating point operations and 115 seconds to compute (the LP DASA does not require this matrix; however, as noted above, this is an upper bound on the number of nonzeros that can occur during the execution of the LP DASA). This high level of fill-in in $\mathbf{L}_B$ is the result of the highly irregular nonzero pattern of $\mathbf{B}$. The matrix $\mathbf{A}_0$, corresponding to an optimal solution of the linear programming problem, has 5,446 columns taken from the original columns of $\mathbf{B}$ plus the 6,071 columns of the appended matrix $10^{-6}\mathbf{I}$.

We wrote a set of Matlab scripts that implements our complete sparse Cholesky update/downdate algorithm, discussed in section 4 and section 5. We first found $\mathbf{P}_B$, using 101 trials of Matlab's column multiple minimum degree ordering algorithm (`colmmd` [17]), 100 of them with a different random permutation of the rows of $\mathbf{B}$. We then took the best permutation found. The time for the initial Cholesky factorization of $\mathbf{A}\mathbf{A}^\mathsf{T}$ is given by (see [13]) the following expression:

$$O\left(\sum_{j=1}^{m} |\mathcal{L}_j|^2\right),$$

which is $O(m^3)$ if $\mathbf{L}$ is dense. With our permutation $\mathbf{P}_B$, the factor $\mathbf{L}$ of $\mathbf{A}_0\mathbf{A}_0^\mathsf{T}$ has 831 thousand nonzeros and took 481 million floating point operations and 51 seconds to compute (using Matlab's `chol`). Following the method used in LP DASA, we added 10^{-12} to the diagonal to ensure positive definiteness (the columns of $\mathbf{B}$ are scaled to be unit vectors). We used the same permutation $\mathbf{P}_B$ for the entire experiment. The initial symbolic factorization took 15 seconds (Algorithm 2). It is this matrix and its factor that are required by the LP DASA.

We did not use Matlab's sparse matrix data structure since Matlab removes explicit zeros. Changing the nonzero pattern by a single entry can cause Matlab to make a new copy of the entire matrix. This would defeat the asymptotic performance of our algorithms. Instead, the column-oriented data structure we use for $\mathcal{L}$, $\mathcal{L}^\sharp$, and $\mathbf{L}$ consists of three arrays of length $|\mathbf{L}_B|$, an array of length m that contains indices to the first entry in each column, and an array of length m holding the number of nonzeros in each column. The columns are allocated so that each column can hold as many nonzeros as the corresponding column of $\mathbf{L}_B$ without reallocation.

Starting with the matrix $\mathbf{A}_0$, we added one column at a time until all 12,230

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

columns from the original $\mathbf{B}$ were present, and then we removed them one at a time (in a first-in first-out order) to obtain the starting $\mathbf{A}_0$. No linear programming solver does so much work, but this provides a simple contrived test of our methods under a wide range of conditions that could occur in practice. The average time and work required to modify the factors at each step was 3.5 seconds and 2.6 million floating point operations. By comparison, solving a linear system $\mathbf{LDL}^\mathsf{T}\mathbf{x} = \mathbf{b}$ with a dense right-hand side $\mathbf{b}$ (using our column-oriented data structure for $\mathbf{L}$) at each step took an average of 6.9 seconds and 5.0 million floating point operations (each solve takes $O(|\mathcal{L}|)$ time). Thus, modifying the factors takes about half the time and work as using the factors to solve a linear system.

The time taken for the entire update/downdate computation would be much smaller if our code was written in a compiled language. Solving one system $\mathbf{LDL}^\mathsf{T}\mathbf{x} = \mathbf{b}$ with a dense right-hand side (using the factorization of the matrix $\mathbf{A}_0$) takes 5.5 seconds using our column-oriented data structure, 1.3 seconds using a Matlab sparse matrix for $\mathbf{L}$, and 0.22 seconds using Fortran 77. Hence, for the DFL001 test problem, we expect that our computation (both symbolic and numerical) would take about a tenth of a second per update or downdate in Fortran 77, on average.

7.2. Numerical accuracy. In order to measure the error in the computed Cholesky factorization, we evaluated the difference $\|\mathbf{AA}^\mathsf{T} - \hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}\|_1$, where $\hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}$ is the computed Cholesky factorization. For the airline scheduling matrix of section 7, $\hat{\mathbf{L}}$ has up to 1.49 million nonzeros and it is impractical to compute the product $\hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}$ after each update. To obtain a quick and accurate estimate for $\|\mathbf{E}\|_1$, where $\mathbf{E} = \mathbf{AA}^\mathsf{T} - \hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}$, we applied the strategy presented in [23] (see [24, p. 139] for a symbolic statement of the algorithm) to estimate the 1-norm of a matrix. That is, we used a gradient ascent approach to compute a local maximum for the following problem:

$$\max\{\|\mathbf{Ex}\|_1 : \|\mathbf{x}\|_1 = 1\}.$$

Since $\hat{\mathbf{L}}$ is used multiple times in the following algorithm, we copied our data structure for $\hat{\mathbf{L}}$ into a Matlab sparse matrix. In exact arithmetic, Algorithm 7 computes a lower bound on the 1-norm of $\mathbf{E}$.[3]

Algorithm 7 (estimate 1-norm of an m-by-m matrix $\mathbf{E}$).

```
    x_i = 1/m for 1 ≤ i ≤ m
    ρ = 0 (ρ is the current estimate for ||E||)
    while ||Ex||_1 > ρ do
        ρ = ||Ex||_1
        for i = 1 to m do
            y_i =  1 if (Ex)_i ≥ 0
            y_i = −1 if (Ex)_i < 0
        end for
        z = E^T y
        j = arg max {|z_i| : i = 1 to m}
        if |z_j| ≤ z^T x return
        x_i = 0 for i = 1 to n
        x_j = 1
    end while
end Algorithm 7
```

[3]Algorithm 7 is available in Mathwork's contributed m-files ftp site, ftp.mathworks.com, as the file pub/contrib/v4/linalg/normest1.m.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

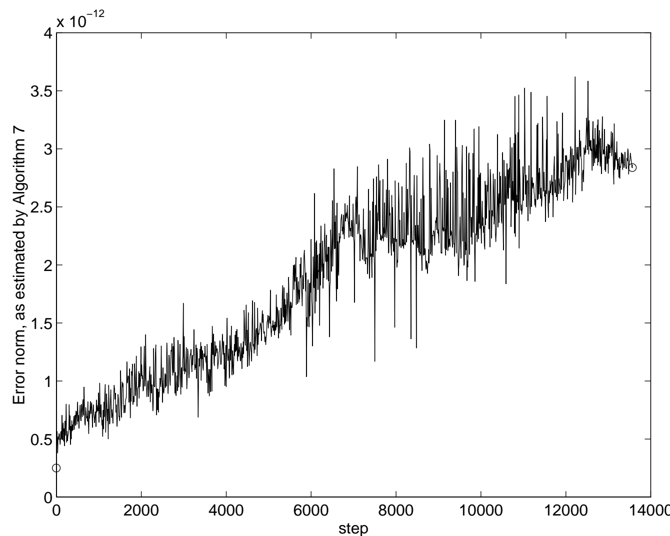

FIG. 7.1. *Estimated* 1*-norm of error in the* $\mathbf{LDL}^\mathsf{T}$ *factorization.*

To improve the accuracy of the 1-norm estimate, we used Algorithm 7 three times. In the second and third trials, a different starting vector $\mathbf{x}$ was used as described in [23]. Observe that Algorithm 7 only makes use of the product between the matrix $\mathbf{E}$ and a vector. This feature is important in the context of sparse matrices since $\mathbf{E}$ contains the term $\hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}$. It is impractical to compute the product $\hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}$, but it is practical to multiply $\hat{\mathbf{L}}\hat{\mathbf{D}}\hat{\mathbf{L}}^\mathsf{T}$ by a vector. For the airline scheduling matrix of section 7, the values for $\|\mathbf{E}\|_1$ initially, at step 6,784, and at the end were 2.5×10^{-13}, 2.4×10^{-12}, and 3.0×10^{-12}, respectively. The estimates obtained using Algorithm 7 were nearly identical at the same three steps (2.6×10^{-13}, 2.5×10^{-12}, and 2.9×10^{-12}, respectively). On the other hand, the times to compute $\|\mathbf{E}\|_1$ at the initial step and at step 6,784 were 119.4 and 266.4 seconds, while the times for three trials of Algorithm 7 were 8.3 and 13.5 seconds, respectively (excluding the time to construct the Matlab sparse matrix for $\hat{\mathbf{L}}$).

Our methods were quite accurate for this problem. After 6,784 updates and 6,784 downdates, or 13,568 changes in $\mathbf{A}$, the 1-norm of $\mathbf{E}$ increased by only a factor 12. Figure 7.1 shows the estimated value of $\|\mathbf{E}\|_1$ computed every 10 steps using Algorithm 7. The initial and final estimates are circled. The 1-norm of the matrix $\mathbf{AA}^\mathsf{T}$ increases from 458.0 initially to 1107.0 at iteration 6,784, then returns to 458.0 at iteration 13,568. Hence, the product of the computed Cholesky factors agrees with the product $\mathbf{AA}^\mathsf{T}$ to about 15 significant digits initially, while the products agree to about 14 significant digits after 13,568 modifications of $\mathbf{A}$.

7.3. Alternative permutations. Our methods are optimal in the sense that they take time proportional to the number of nonzero entries in $\mathbf{L}$ and $\mathbf{D}$ that change at each step. However, they are not optimal with respect to fill-in, since we assume a single initial permutation and no subsequent permutations. A fill-reducing ordering of $\mathbf{BB}^\mathsf{T}$ might not be the best ordering to use for all the $\mathbf{A}$ matrices. A simple pathological example is the m-by-n matrix $\mathbf{B}$, where $n = m(m-1)/2$ and the nonzero pattern of each column of $\mathbf{B}$ is a unique pair of integers from the set $\{1, 2, \ldots, m\}$.

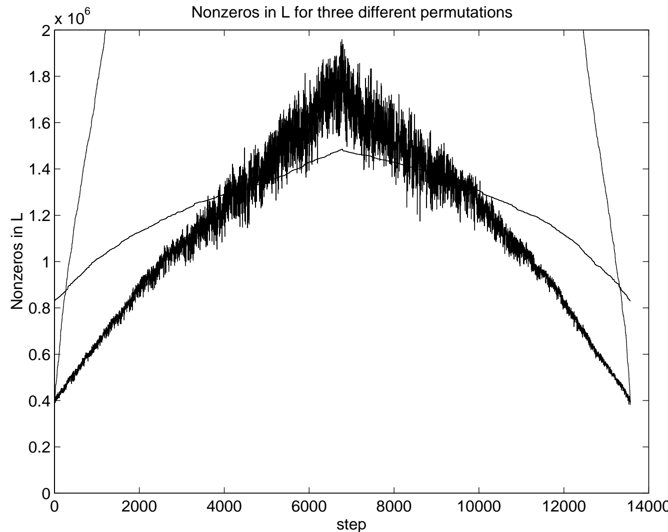

FIG. 7.2. *Nonzeros in* $\mathbf{L}$ *using three different permutations.*

In this case, every element of $\mathbf{BB}^\mathsf{T}$ is nonzero, while the nonzero pattern of $\mathbf{AA}^\mathsf{T}$ is arbitrary. As the matrix $\mathbf{A}$ changes, it might be advantageous to compute a fill-reducing ordering of $\mathbf{AA}^\mathsf{T}$ if the size of its factors grow "too large." A refactorization with the new permutation would then be required.

We found a fill-reducing permutation $\mathbf{P}_A$ of the starting matrix $\mathbf{A}_0\mathbf{A}_0^\mathsf{T}$ (again, the best of 101 trials of `colmmd`). This results in a factor $\mathbf{L}$ with 381 thousand nonzeros, requiring only 169 million floating point operations to compute. This is significantly less than the number of nonzeros (831 thousand) and floating point operations (481 million) associated with the fill-reducing permutation for $\mathbf{BB}^\mathsf{T}$. We also computed an ordering of $\mathbf{AA}^\mathsf{T}$ at each step, using `colmmd` just once, and then computed the number of nonzeros in the factor if we were to factorize $\mathbf{AA}^\mathsf{T}$ using this permutation ($\mathbf{P}_s$). Although it only takes about 1 second to compute the ordering [17] and symbolic factorization [18], it is not practical to use 100 random trials at each step.

Figure 7.2 depicts the nonzero counts of $\mathbf{L}$ for these three different permutations at each of the 13,568 steps. The fixed permutation $\mathbf{P}_B$ results in the smooth curve starting at 831 thousand and peaking at 1.49 million. The fixed permutation $\mathbf{P}_A$ results in a number of nonzeros in $\mathbf{L}$ that starts at 381 thousand and rises quickly, leaving the figure at step 1,206 and peaking at 7.4 million in the middle. It surpasses $\mathbf{P}_B$ at step 267. Using a permutation $\mathbf{P}_s$, computed at each step s, gives the erratic line in the figure, starting at 390 thousand and peaking at 1.9 million in the middle. These results indicate that it might be advantageous to start with the fixed permutation $\mathbf{P}_A$, use it for 267 steps, and then refactorize with the permutation $\mathbf{P}_s$ computed at step 267. This results in a new factor with only 463 thousand nonzeros. Near the center of the figure, however, $\mathbf{A}$ includes most of the columns in $\mathbf{B}$, and in this case the $\mathbf{P}_B$ permutation should be used.

8. Summary. We have presented a new method for updating and downdating the factorization $\mathbf{LDL}^\mathsf{T}$ or $\mathbf{LL}^\mathsf{T}$ of a sparse symmetric positive definite matrix $\mathbf{AA}^\mathsf{T}$. Our experimental results show that the method should be fast and accurate in practice. Extensions to an arbitrary sparse symmetric positive definite matrix, $\mathbf{M}$, have been discussed. We mention additional extensions to our work that would be useful.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

One drawback of our approach is the increase in storage. With the compressed pattern (a supernodal form of **L**) [13], the storage of **L** is dominated by the floating point values. In our storage scheme, we require two integers per floating point value (a row index and its multiplicity). Our methods almost double the storage for **L** (assuming 8-byte floating point values and 4-byte integers). A method based on a supernodal form could require less time and storage. However, supernodes would merge during update and split during downdate, which would be complicated to manage. Although implementing supernodes in the context of our current update and downdate methods is difficult, the numerical factorization can still be based on a supernodal method.

Some applications, such as local mesh refinement and coarsening and primal active set algorithms in optimization, require changes to the dimension of a symmetric positive definite matrix. These changes can be implemented using the techniques presented in this paper.

REFERENCES

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] R. H. Bartels, G. H. Golub, and M. A. Saunders, *Numerical techniques in mathematical programming*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian, and K. Ritter, eds., Academic Press, New York, 1970, pp. 123–176.

[3] J. M. Bennett, *Triangular factors of modified matrices*, Numer. Math., 7 (1965), pp. 217–221.

[4] C. H. Bischof, C.-T. Pan, and P. T. P. Tang, *A Cholesky up- and downdating algorithm for systolic and SIMD architectures*, SIAM J. Sci. Comput., 14 (1993), pp. 670–676.

[5] N. A. Carlson, *Fast triangular factorization of the square root filter*, AIIA J., 11 (1973), pp. 1259–1265.

[6] S. M. Chan and V. Brandwajn, *Partial matrix refactorization*, IEEE Trans. on Power Systems, PWRS-1 (1986), pp. 193–200.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Electrical Engineering and Computer Science Series, MIT Press, Cambridge, MA; McGraw–Hill, New York, 1990.

[8] J. J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403–407.

[9] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.

[10] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *Yale sparse matrix package,* I: *The symmetric codes*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1145–1151.

[11] A. George and J. W. H. Liu, *The design of a user interface for a sparse matrix package*, ACM Trans. Math. Software, 5 (1979), pp. 139–162.

[12] A. George and J. W. H. Liu, *An optimal algorithm for symbolic factorization of symmetric matrices*, SIAM J. Comput., 9 (1980), pp. 583–593.

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

[14] A. George, J. Liu, and E. Ng, *A data structure for sparse QR and LU factorizations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 100–121.

[15] J. R. Gilbert, *Predicting Structure in Sparse Matrix Computations*, Tech. report CS-86-750, Computer Science Dept., Cornell Univ., Ithaca, NY, 1986.

[16] J. R. Gilbert, *Predicting structure in sparse matrix computations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 62–79.

[17] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

[18] J. R. Gilbert, E. G. Ng, and B. W. Peyton, *An efficient algorithm to compute row and column counts for sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1075–1091.

[19] J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 862–874.

[20] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.
[21] P. E. Gill and W. Murray, *Quasi-Newton methods for unconstrained optimization*, J. Inst. Math. Appl., 9 (1972), pp. 91–108.
[22] P. E. Gill, W. Murray, and M. A. Saunders, *Methods for computing and modifying the LDV factors of a matrix*, Math. Comp., 29 (1975), pp. 1051–1077.
[23] W. W. Hager, *Condition estimates*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 311–316.
[24] W. W. Hager, *Applied Numerical Linear Algebra*, Prentice–Hall, Englewood Cliffs, NJ, 1988.
[25] W. W. Hager, *Updating the inverse of a matrix*, SIAM Rev., 31 (1989), pp. 221–239.
[26] W. W. Hager, *The LP dual active set algorithm,* in High Performance Algorithms and Software in Nonlinear Optimization, R. De Leone, A. Murli, P. M. Pardalos, and G. Toraldo, eds., Kluwer Academic Publishers, Norwell, MA, to appear.
[27] K. H. Law, *Sparse matrix factor modification in structural reanalysis*, Internat. J. Numer. Methods Engrg., 21 (1985), pp. 37–63.
[28] K. H. Law, *On updating the structure of sparse matrix factors*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 2339–2360.
[29] J. W. H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.
[30] C.-T. Pan, *A modification to the LINPACK downdating algorithm*, BIT, 30 (1990), pp. 707–722.
[31] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.
[32] D. G. Row, G. H. Powell, and D. P. Mondkar, *Solution of progressively changing equilibrium equations for nonlinear structures*, Comput. & Structures, 7 (1977), pp. 659–665.
[33] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.
[34] A. H. Sherman, *On the Efficient Solution of Sparse Systems of Linear and Nonlinear Equations*, Tech. report 46, Dept. of Computer Science, Yale Univ., New Haven, CT, 1975.
[35] G. Strang, *Linear Algebra and Its Applications*, Academic Press, New York, 1980.
[36] J. H. Wilkinson, *Linear algebra algorithms*, in Software for Numerical Mathematics, Proceedings of the Loughborough Univ. of Technology Conf. of the Inst. of Mathematics and its Appl., 1973, D. J. Evans, ed., Academic Press, New York, 1974, pp. 17–28.

Downloaded 07/28/17 to 128.227.188.129. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php