ILUs and Factorized Approximate Inverses are Strongly Related. Part I: Overview of Results

*Matthias Bollhöfer and †Yousef Saad

March 21, 2000

Abstract

This paper discusses the relations between a broad class of incomplete LU factorization techniques and factorized sparse approximate inverse techniques based on computing triangular matrices Z, W such that $Z^\top AW$ is approximately diagonal. We will show that most of these approaches are essentially equivalent to approximately inverting the triangular factors obtained from a modified incomplete LU factorization of the initial system.
Keywords: sparse matrices, ILU, modified ILU, sparse approximate inverse, $AINV$.
AMS subject classification: 65F05, 65F10, 65F50.

1 Introduction

Iterative methods which combine preconditioning techniques, such as incomplete factorizations, with Krylov–based acceleration, see e.g., [21, 10, 2, 12], are among the most efficient techniques for solving linear systems of the form:

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

*Fakultät für Mathematik, Technische Universität Chemnitz, D–09107 Chemnitz, Germany. Supported by the University of Minnesota and by grants of the DFG BO 1680/1-1. This research was performed while visiting the University of Minnesota at Minneapolis. email: bolle@mathematik.tu-chemnitz.de, URL: http://www.tu-chemnitz.de/∼bolle/.

†Dep. of Computer Science and Engineering, University of Minnesota, 4–192 EE/CSci Building, 200 Union St., SE, Minneapolis, MN 55455–0154. Work supported by NSF and by the Minnesota Supercomputing Institute. email: saad@cs.umn.edu, URL: http://www.cs.umn.edu/∼saad/

where $A \in \mathbb{R}^{n,n}$ is nonsingular and $b \in \mathbb{R}^n$ is a given right hand side. Traditional preconditioners are based on approximately solving the system (1) and the most popular of these are based on approximate factorizations obtained from direct solution methods, such as the LU factorization [9], pp. 92ff. Alternative techniques recently appeared which compute approximate solutions of (1) via an approximate inverse of A, instead of a factorization. One motivation for using such preconditioning techniques is parallelism. Another motivation is that the ILU preconditioners, which have been developed for M matrices [19], often fail for indefinite matrices.

A few of these approximate inverse techniques are based on minimizing the norm $\|I - AM\|$ in some appropriate norm [15, 13, 11, 7]. Others compute the approximate inverse in factored form by seeking two sparse unit upper triangular matrices W, Z, and a diagonal D, such that $Z^\top AW = D$, see e.g. [21, 4, 5, 3, 14]. The latter class of preconditioners turns out to have an algebraic behavior that is similar to the well–known case of the incomplete LU decompositions, e.g. they are stable for M– and H–matrices. This is the perfect analogy to the result on incomplete LU decompositions in [19, 18].

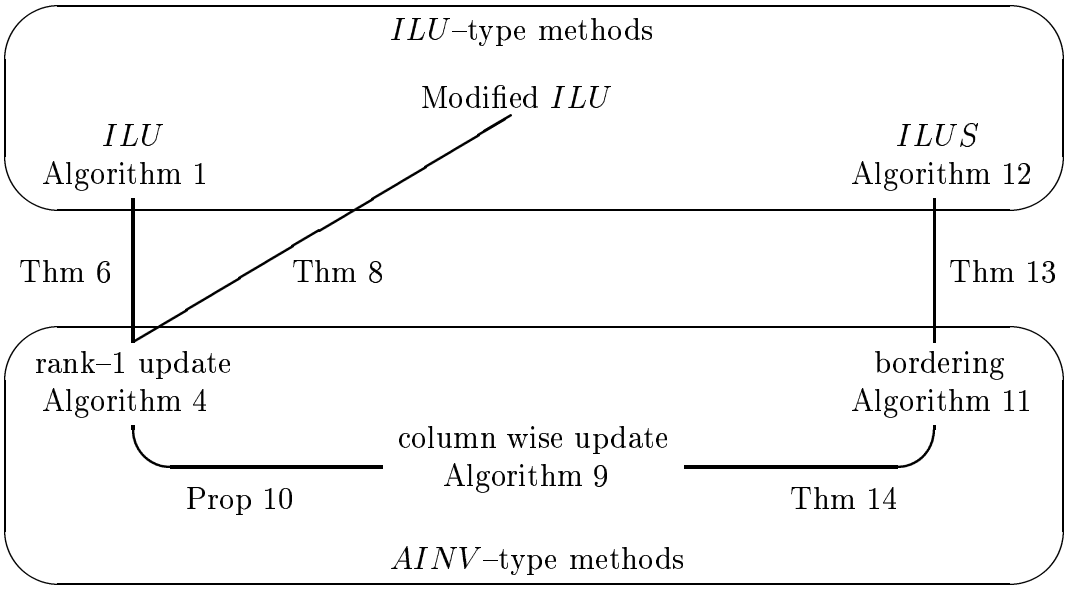
The purpose of this paper is to take an in-depth look at the relationships between these different preconditionings, using the incomplete LU decomposition as a reference point. In particular, we will show that these methods generate factors which can be viewed as approximations of the inverses of the triangular factors obtained by adapting incomplete LU decompositions. With a slight modification of the strategies to drop entries we will also show that matrices resulting from these methods can be viewed as the exact inverses of triangular factors obtained via an incomplete LU decomposition. Specifically, what is required is to suitably modify or construct modified approximate Schur–complements such that the inverse factors are those (or at least close to those) obtained by factorized approximate inverse techniques. On the other hand most of the methods which directly compute a factorized approximate inverse of A can essentially be viewed as variations differing in the way the approximate Schur–complement in the incomplete LU decomposition is defined. The connection between ILU–type algorithms and approximate inverse techniques discussed in this paper is summarized by the illustration in Figure 1.

2 Incomplete LU factorizations

Incomplete LU factorizations approximately construct a factorization

$$A \approx LDU$$

Figure 1: **Relation ILUs — factorized sparse approximate inverses**



where $L, U^\top$ are lower triangular matrices with unit diagonal. A partial LU factorization, when it exists, can be expressed by the equation

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \begin{bmatrix} L_B & 0 \\ L_E & I \end{bmatrix} \begin{bmatrix} D_B & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} U_B & U_F \\ 0 & I \end{bmatrix}, \tag{2}$$

where $L_B, U_B^\top \in \mathbb{R}^{k,k}$ are lower triangular matrices with unit diagonal and $D_B \in \mathbb{R}^{k,k}$ is diagonal. In the simplest case when $k = 1$, this means that $L_B = U_B = 1, D_B = B = A_{11}$. Otherwise L_B, D_B, U_B refer to an already computed LU–decomposition of B. We obtain $L_E D_B U_B = E \in \mathbb{R}^{n-k,k}$, $L_B D_B U_F = F \in \mathbb{R}^{k,n-k}$ and finally

$$S = C - L_E D_B U_F \in \mathbb{R}^{n-k,n-k} \tag{3}$$

denotes the so–called Schur–complement and the exact LU decomposition is obtained by applying (2) successively to the Schur–complement, using $k = 1$, at each step. The successive application of this process ends up with the factorization $S = L_S D_S U_S$. If this decomposition exists it can be re–substituted in (2) to obtain

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \begin{bmatrix} L_B & 0 \\ L_E & L_S \end{bmatrix} \begin{bmatrix} D_B & 0 \\ 0 & D_S \end{bmatrix} \begin{bmatrix} U_B & U_F \\ 0 & U_S \end{bmatrix}, \tag{4}$$

which completes the LU decomposition.

In incomplete factorizations, entries are dropped during this procedure in the L, U factors and the Schur–complement. A strategy which could be applied is the following. Consider sets $\mathcal{R}, \mathcal{C} \subseteq \{2, \ldots, n\}$ and drop entries in the first column of L belonging to $\mathcal{C}$. An analogous strategy is applied for U with respect to $\mathcal{R}$. By this procedure the matrices $U_F = B^{-1}F$, $L_E = EB^{-1}$ are replaced by approximations $\tilde{U}_F, \tilde{L}_E$, i.e.,

$$\begin{bmatrix} 1 \\ \tilde{L}_E \end{bmatrix} \approx \begin{bmatrix} 1 \\ L_E \end{bmatrix}; \quad \begin{bmatrix} 1 & \tilde{U}_F \end{bmatrix} \approx \begin{bmatrix} 1 & U_F \end{bmatrix}. \tag{5}$$

Resulting choices for the approximate Schur–complement are

$$\tilde{S} = C - \tilde{L}_E D_B \tilde{U}_F \tag{6}$$

$$\tilde{S} = C - \tilde{L}_E L_B^{-1} F - \left(E - \tilde{L}_E L_B^{-1} B\right) U_B^{-1} \tilde{U}_F \tag{7}$$

$$\tilde{S} = C - \tilde{L}_E L_B^{-1} F \tag{8}$$

$$\tilde{S} = C - E U_B^{-1} \tilde{U}_F \tag{9}$$

These approximations are obtained from various different expressions of the Schur–complement that are derived from equation (2). For example, upon multiplying both sides of (2) to the right by the inverse of the last factor of the right-hand-side, and equation the $(2,2)$ blocks of the resulting matrices leads to (9). Other ways of defining an approximate Schur–complement can be derived from other equivalent expressions of the Schur–complement. In practice, (6) is the most common scheme for defining Incomplete LU factorizations, see, e.g. [19] or [20]. Typically, (6) produces the smallest amount of fill–in compared with the other formulas.

2.1 K,I,J implementations

A sample routine for performing an incomplete LU decomposition is given by the following abstract algorithm. The MATLAB notation [1] is used for convenience. For two integers k, l, $k : l$ denotes the sequence $(k, k+1, \ldots, l)$ with the convention that whenever $k > l$ the set is empty. For a matrix $A = (A_{ij})_{i=1,\ldots,m,\, j=1,\ldots,n}$, we define

$$A_{k:l,q:r} := (A_{ij})_{i=k,\ldots,l,\, j=q,\ldots,r}.$$

The notation : as a subscript indicates that all columns/rows entries are taken. Thus, $A_{:,2}$ denotes the second column of A and $A_{2,:}$ denotes its second row.

Algorithm 1 (Incomplete LU factorization (ILU))
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute an ILU factorization $A \approx LDU$.

$L = U = I, S = A, D_{11} = A_{11}, \mathcal{C} = \mathcal{R} = \emptyset.$

for $k = 1 : n-1$

$L_{k:n,k} = \begin{bmatrix} 1 \\ S_{k+1:n,k} D_{kk}^{-1} \end{bmatrix}$, $U_{k,k:n} = \begin{bmatrix} 1 & D_{kk}^{-1} S_{k,k+1:n} \end{bmatrix}$.

Apply a dropping rule to $L_{k+1:n,k}, U_{k,k+1:n}$ *and include all* (l,k) *in* $\mathcal{C}$, (k,m) *in* $\mathcal{R}$ *for which of* $L_{l,k}, U_{k,m}$ *have been dropped*

Compute $S_{k+1:n,k+1:n}$ *according to one of the possible choice from (6)–(9), i.e. compute* $S_{k+1:n,k+1:n}$ *from one of the following options.*

$$S_{k+1:n,k+1:n} = \begin{cases} S_{k+1:n,k+1:n} - L_{k+1:n,k} D_{kk} U_{k,k+1:n} \\ S_{k+1:n,k+1:n} - L_{k+1:n,k} S_{k,k+1:n} - (S_{k+1:n,k} - L_{k+1:n,k} D_{kk})\, U_{k,k+1:n} \\ S_{k+1:n,k+1:n} - L_{k+1:n,k} S_{k,k+1:n} \\ S_{k+1:n,k+1:n} - S_{k+1:n,k} U_{k,k+1:n} \end{cases}$$

$D_{k+1,k+1} = S_{k+1,k+1}$

end

The above algorithm is based on the so-called K, I, J version (or 'rank-one' update version) of Gaussian elimination. One of its drawbacks lies in its practical implementation. At every step of the process rows $i+1$ to n of the matrix S (which typically would be represented by a single data structure) are altered, leading to a need for expensive linked lists, or the use of elbow room. In spite of these drawbacks the algorithm is attractive for several reasons, and it has been used by a few authors as to develop incomplete factorizations. One of its advantages is the ease with which powerful pivoting and reordering strategies can be implemented.

At this point we point out an important observation on the approximate Schur–complement – and this may cause some confusion. Instead of applying (2),(3) successively for the case $k = 1$ and deriving the Schur–complement step by step from equations (6)–(9) as it is done in Algorithm 1, we can alternatively obtain the approximate Schur–complement after, say l steps by immediately taking the same equations from (6)–(9) with the already computed blocks L_B, L_E, U_B, U_F. As it turns out, this leads to precisely the same approximate Schur–complement.

Lemma 2 *Suppose that* l *steps of Algorithm 1 have been performed with one of the four options based on (6) – (9). Then the same matrix* $\tilde{S} \equiv S_{l+1:n,l+1:n}$ *can be obtained by applying formulas (6)–(9) to the initial matrix* A *partitioned as* $A = \begin{bmatrix} B & F \\ E & C \end{bmatrix}$ *with* $B \in \mathbb{R}^{l,l}, C \in \mathbb{R}^{n-l,n-l}$ *and the other blocks of* A, L, U *analogously partitioned.*

Proof. For (6) the result is easy to show. We will only show (8) since the proof for (7) and (9) is analogous. The proof is based on observing the effect of the successive

transformations (8) on the global original matrix, and follows a standard argument used in Gaussian elimination. At step i,

$$A^{new} := \begin{pmatrix} I & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -L_{i+1:n,i} & I \end{pmatrix}^{-1} A^{old} \equiv \left(I - l_i e_i^\top\right)^{-1} A^{old}$$

where l_i denotes the n-dimensional vector consisting of the vector $L_{i+1:n,i}$ completed by i zeros above it. Indeed, the (2,2) block of size $(n-i) \times (n-i)$ in the above matrix is the same as the one obtained from the transformation (8). Consider now the result of accumulating these transformations for k consecutive steps. The inverse of each elementary factor $I - l_i e_i^\top$ is $I + l_i e_i^\top$. The inverse of the accumulated transformations is therefore,

$$(I + l_1 e_1^\top)(I + l_2 e_2^\top) \cdots (I + l_k e_k^\top) = I + l_1 e_1^\top + l_2 e_2^\top + \cdots + l_k e_k^\top$$

The above equality follows from a simple inductive argument. This is precisely the matrix

$$\begin{pmatrix} L_{1:k,1:k} & 0 \\ L_{k+1:n,1:k} & I \end{pmatrix}$$

The corresponding (2,2) block of the matrix A^{new} gives precisely the desired result (8) with a block size of k. □

2.2 I, K, J variants of ILU

An alternative implementation of ILU, and a common one used in the context of preconditioners, is based on the I, K, J version of Gaussian elimination. This is sketched in the next algorithm.

Algorithm 3 (Incomplete LU factorization (ILU))
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute an ILU factorization $A \approx LDU$.

$L = U = I, S = A, \mathcal{C} = \mathcal{R} = \emptyset$.
for $i = 1 : n$
  $w = A_{i,:}$
  for $j = 1, \ldots, i-1$ *and when* $w_k \neq 0$
    $w_j = w_j / D_{jj}$
    Apply a rule to drop w_j. If w_j is dropped, $\mathcal{C} := \mathcal{C} \cup \{(i,j)\}$
    if $w_j \neq 0$, $w_{j+1:n} := w_{j+1:n} - w_j U_{j,j+1:n}$
  end

Apply a dropping rule to w_j/w_i for all $j > i$ and include all (i,j) to $\mathcal{R}$ for which w_j has been dropped

Define $D_{ii} = w_i, \quad U_{i,i:n} = w_{i:n}/D_{ii}, \quad L_{i,1:i-1} = w_{1:i-1}$

end

Note that this algorithm is again expressed in terms of dropping sets $\mathcal{C}, \mathcal{R}$ for row i of L and U. These sets can be selected statically in the level–of–fill strategy or dynamically as in the $ILUT$ strategy. In the case of static dropping strategies it is known that Algorithm 3 and Algorithm 1 with S defined by (6) will deliver the same factors when the patterns $\mathcal{C}, \mathcal{R}$ are the same for both algorithms. However this relation is still true, if a dropping rule according to some fixed drop tolerance τ is applied.

In practice, incomplete factorization algorithms are typically organized such that the L, D, U factors are stored in one single data structure. The attraction of the above implementation is clear: the rows of L and U are determined one a time and are easily added to the existing data-structure.

2.3 Dropping Strategies

As was mentioned earlier, there are two broad classes of dropping strategies. First there are strategies which drop elements based on the pattern of the matrix only. This includes the level–of–fill strategy [19]. Another class of methods is based on dropping elements dynamically, based on their magnitude. A number of other strategies combine graph based methods with threshold dropping.

A well–known class of strategies for discarding elements in the LU factorization is one that is based on using the graph (i.e., pattern) of the matrix. For example the $ILU(0)$, the simplest of these techniques, discards any fill–in. This means that Gaussian elimination is essentially performed only on the pattern of A and any fill–in generated is simply ignored. Various generalizations of this method have been developed, see e.g., [21] for an overview. In addition to graph–based criteria there are a number of strategies that determine the sparsity pattern $\mathcal{C}, \mathcal{R}$ throughout the process by monitoring the growth of the elements. In this case only small entries are dropped. Strategies of this type are also discussed in [21].

It is important to point out here that the results we show concern not only the 'static' dropping strategies but also some dynamic dropping, e.g. with respect to a prescribed drop tolerance τ, similar to the threshold based ILUT preconditioning. To be more specific, throughout the paper we assume that any dropping rule we use for sparsifying a vector has information about its numerical values and its associated coordinates. For example, in Algorithm 1 the dropping rule for $L_{k+1:n,k}$ the coordinates $(k+1,k), \ldots, (n,k)$

in addition to the numerical values $L_{k+1:n,k}$. In the sequel, whenever we successfully apply a dropping rule the coordinates are added to the dropping sets (in Algorithm 1, $\mathcal{C}, \mathcal{R}$).

Indeed with some more complicated dynamic dropping, different versions of Gaussian elimination and different procedures that would otherwise deliver the same results in exact Gaussian elimination, may lead to different ILU factorizations because the dropping strategies may yield different patterns. In general threshold based methods are harder to analyze than pattern-based algorithms.

3 Relations between $AINV$ and ILUs

In recent years new preconditioners were proposed, that belong to the broad class of approximate inverse techniques. Among these methods, are those that calculate directly an approximate inverse M to A, see, e.g., [7, 11]. Another subclass of methods obtain this approximate inverse in a factored form. Specifically, unit lower triangular matrices W and Z are found such that $W^\top AZ \approx D$, where D is a diagonal matrix. A proposed method in this category, called AINV, was proposed in [4, 5].

3.1 $AINV$

The method in [4, 5] computes a decomposition of the form $W^\top AZ = D$, where W, Z are unit lower triangular matrices, and D is a diagonal. In the exact factorization case, the matrices W and Z are the inverses of $L^\top$ and U, respectively, in the standard LDU decomposition $A = LDU$, when this decomposition exists. The matrices W and Z can be directly computed by biorthogonalization. Indeed, since

$$W^\top A = DU$$

is upper triangular, we immediately get $(W_{:,i})^\top A_{:,j} = 0$ for any $j < i$, which means that column i of W is orthogonal to the first $i-1$ columns of A. Alternatively, we can also try to make columns $i+1, \ldots, n$ of W orthogonal to the first i columns of A. This makes it possible to successively orthogonalize all columns of W against each of the columns of A. During this procedure one can drop small entries or entries outside a certain sparsity pattern leading to an incomplete biorthogonalization process and finally to a factored approximate inverse. There are several advantages of having the inverses of a triangular matrix available – see the introduction. One of the most important reasons for the current interest in approximate inverse type preconditioners is their appeal for parallel processing. It is worth mentioning that there has been some work

on methods for inverting triangular matrices which are computed from a standard LU factorization, based on the same motivations, see [8]. However these do not compute factored approximate inverses and will not be considered here since they are based on using a standard incomplete LU decomposition and inverting the resulting triangular factors.

Algorithm 4 (Factorized Approximate INVerse, rank–1 update version)
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute $A^{-1} \approx ZD^{-1}W^\top$.

$p = q = \underbrace{(0, \ldots, 0)}_{n}, Z = W = I_n, \mathcal{C} = \mathcal{R} = \emptyset.$

for $i = 1 : n$

Define p, q either by (10) or by (11)

$$p_{i:n} = A_{:,i}^\top W_{:,i:n}, \qquad q_{i:n} = A_{i,:} Z_{:,i:n} \tag{10}$$

$$p_{i:n} = Z_{:,i}^\top A^\top W_{:,i:n}, \; q_{i:n} = W_{:,i}^\top A Z_{:,i:n} \tag{11}$$

Set $p_{i+1:n} := p_{i+1:n}/p_i$, $q_{i+1:n} := q_{i+1:n}/q_i$ and apply a dropping rule to $p_{i+1:n}, q_{i+1:n}$
include all (j, i) to $\mathcal{C}$, (i, k) to $\mathcal{R}$ for which p_j, q_k have been dropped
$W_{:,i+1:n} = W_{:,i+1:n} - W_{:,i} p_{i+1:n}$, $Z_{:,i+1:n} = Z_{:,i+1:n} - Z_{:,i} q_{i+1:n}$
apply a dropping rule to $W_{1:i,i+1:n}$, define $\mathcal{C}_i$ by those (l, k) for which W_{kl} is dropped
apply a dropping rule to $Z_{1:i,i+1:n}$, define $\mathcal{R}_i$ by those (k, l) for which Z_{kl} is dropped

end

Choose diagonal entries of D as the components of p or q.

Clearly, if no entry is dropped and if there exists an LDU decomposition of A, then $W = L^{-\top}$, $Z = U^{-1}$. In this case it can be immediately seen by induction that after step i of the algorithm, columns $i + 1, \ldots, n$ of W are orthogonal to column $1, \ldots, i$ of A and likewise columns $i + 1, \ldots, n$ of Z are orthogonal to rows $1, \ldots, i$ of A. It is remarkable that the computation of Z and W can be performed independently for (10). In other words, the sets $\mathcal{R}, \mathcal{R}_i$ do not affect the computation of W and the sets $\mathcal{C}, \mathcal{C}_i$ do not affect the computation of of Z .

It is important to note that the $AINV$ method uses $\mathcal{R} = \mathcal{C} = \emptyset$ and only $\mathcal{R}_i$, $\mathcal{C}_i$ are chosen by discarding entries in Z, W that are less than a certain drop tolerance. Moreover it has been pointed out in [4], that dropping entries of p, q produces poor results. We will still consider this variant for generality and because it has an interesting direct connection with ILU. In [4, 5] p, q were defined via (10), while in [14, 3] for the case of symmetric positive definite matrices (11) is used (In this situation, $W \equiv Z$).

Clearly, the strict biorthogonality property of the exact factorized inverse does not hold anymore if a drop tolerance is introduced. Interestingly, however, stability can be proved

for H–matrices, in the case incomplete LU factorizations as well as for $AINV$. So, there must be a close connection between both classes of algorithms and the next algorithm will provide a first bridge between both approaches. Essentially, incomplete LDU factorization can be computed, and its factors L, U can be inverted on the fly, leading to a progressive form of a factored inverse. Specifically, if at step $i-1$ we have a matrix U of the form,

$$U = \begin{bmatrix} U_{1:i-1,1:i-1} & U_{1:i-1,i:n} \\ \mathrm{O} & I \end{bmatrix}$$

the i-th step will compute the entries $U_{i,i+1:n}$ and add them to the current U to get U_{new}. Let $q^\top$ be the row vector $q^\top = U_{i,:} - e_i^\top$. Note that the 'diagonal' element q_i of q is zero. Then,

$$U_{new} = U + e_i q^\top$$

Because of the structure of U and q it is easy to see that $q^\top U = q^\top$, and so

$$U_{new} = (I + e_i q^\top) U$$

Therefore,

$$U_{new}^{-1} = U^{-1}(I + e_i q^\top)^{-1} = U^{-1}(I - e_i q^\top) \ .$$

Of course analogous arguments will hold for L. This provides a formula for progressively computing $L^{-\top}, U^{-1}$ throughout the algorithm. We call the inverse factors Z, W as in Algorithm 4.

Algorithm 5 (ILU with progressive inversion of L, U)
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute $A \approx LDU$, $A^{-1} \approx ZD^{-1}W^\top$.

$L = U = I, S = A, Z = W = I$, $D_{11} = A_{11}$, $\mathcal{C} = \mathcal{R} = \emptyset$

for $i = 1 : n-1$

Set: $p_j = \begin{cases} 0 & \text{for } j < i \\ S_{ji}/S_{ii} & \text{otherwise} \end{cases}$, $\quad q_k = \begin{cases} 0 & \text{for } k < i \\ S_{ik}/S_{ii} & \text{otherwise} \end{cases}$

Apply a dropping rule to $p_{i+1:n}, q_{i+1:n}$ and include those (j,i) to $\mathcal{C}$, (i,k) to $\mathcal{R}$ for which p_j, q_k have been dropped

Set: $L_{:,i} = e_i + p$; $U_{i,:} = e_i^\top + q^\top$; $W = W(I - e_i p^\top)$; $Z = Z(I - e_i q^\top)$

Apply a dropping rule to $W_{1:i,i+1:n}$, define $\mathcal{C}_i$ by those (l,k) for which W_{kl} is dropped

apply a dropping rule to $Z_{1:i,i+1:n}$, define $\mathcal{R}_i$ by those (k,l) for which Z_{kl} is dropped

Define $S_{i+1:n,i+1:n}$ from one of the options in Algorithm 1

Set: $D_{i+1,i+1} = S_{i+1,i+1}$

end

The notation of Algorithm 5 already suggests that Z, W coincide with those of Algorithm 4. This is confirmed by Theorem 6.

Theorem 6 *Suppose that in Algorithm 4 and Algorithm 5 the same dropping rules are applied to p, q (same $\mathcal{C}, \mathcal{R}$) and that there is no dropping rule applied to W, Z (empty sets $\mathcal{C}_i, \mathcal{R}_i$, $i = 1, \ldots, n$). Then certain choices of S in Algorithm 5, and p, q in Algorithm 4 will imply the following identities.*

Choice of S / p, q	*Alg. 5* (8)	*Alg. 4* (10)	*Alg. 5* (7)	*Alg. 4* (11)	*Alg. 5* (9)	*Alg. 4* (10)
	$\Downarrow$		$\Downarrow$		$\Downarrow$	
Identities	$L^{-\top} = W_{Alg.5/4}$		$L^{-\top} = W_{Alg.5/4}$ $U^{-1} = Z_{Alg.5/4}$		$U^{-1} = Z_{Alg.5/4}$	
	$\operatorname{diag}(D_{Alg.5}) = p_{Alg.4}$		$\operatorname{diag}(D_{Alg.5}) = \begin{cases} p_{Alg.4} \\ q_{Alg.4} \end{cases}$		$\operatorname{diag}(D_{Alg.5}) = q_{Alg.4}$	

Proof. We will only prove the first result for W and p, since the proof for the other cases is analogous. We will show by induction on i, that W is identical in both methods after any step i, that the first i diagonal entries of D coincide with $p_{1:i}$ and that

$$S_{i+1:n,i+1:n} = \begin{bmatrix} 0 & I_{n-i} \end{bmatrix} W^\top A \begin{bmatrix} 0 \\ I_{n-i} \end{bmatrix}. \tag{12}$$

Initially, for $i = 0$ there is nothing to show since obviously $W = I$ in both algorithms and $S = A$. Now suppose that W is identical before we enter step i of each algorithm. Suppose that the first $i-1$ diagonal entries of D coincide with the first $i-1$ components of p and that

$$S_{i:n,i:n} = \begin{bmatrix} 0 & I_{n-i+1} \end{bmatrix} W^\top A \begin{bmatrix} 0 \\ I_{n-i+1} \end{bmatrix}.$$

We immediately obtain

$$p_{i:n}^\top = W_{:,i:n}^\top A_{:,i} = S_{i:n,i}.$$

From this it follows that $p_i = s_{ii}$ and $p_{i+1:n}^{(new)} = \frac{p_{i+1:n}^{(old)}}{p_i} = L_{i+1:n,i}$. Since we choose the same dropping rule for both algorithms, this equality still holds after sparsifying these entries.

Obviously the update procedure

$$W_{:,i+1:n}^{(new)} = W_{:,i+1:n}^{(old)} - W_{:,i}^{(old)} p_{i+1:n}^{(new)}$$

from Algorithm 4 is the nontrivial update part on

$$W^{(new)} = W^{(old)} \left(I - e_i p^\top\right)$$

in Algorithm 5. Now for $S_{i+1:n,i+1:n}$ we have the update procedure

$$\begin{aligned}
S^{(new)}_{i+1:n,i+1:n} &:= S^{(old)}_{i+1:n,i+1:n} - L_{i+1:n,i} S^{(old)}_{i,i+1:n} \\
&= \begin{bmatrix} 0 & I_{n-i} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -p^{(new)}_{i+1:n} & I \end{bmatrix} \begin{bmatrix} S^{(old)}_{i,i+1:n} \\ S^{(old)}_{i+1:n,i+1:n} \end{bmatrix} \\
&= \begin{bmatrix} 0 & I_{n-i} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -p^{(new)}_{i+1:n} & I \end{bmatrix} \begin{bmatrix} 0 & I_{n-i+1} \end{bmatrix} (W^{(old)})^\top A \begin{bmatrix} 0 \\ 0 \\ I_{n-i} \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0 & I_{n-i} \end{bmatrix} (W^{(new)})^\top A \begin{bmatrix} 0 \\ 0 \\ I_{n-i} \end{bmatrix}.
\end{aligned}$$

This completes the proof. □

This surprising result is not too unexpected. As pointed out earlier, in Algorithm 4 (and as well in the $AINV$ method) one can compute Z, W independently of each other for (10). However, in an ILU method where dropping strategies are applied to the L and U factors at the same time, we have an interaction between the computation of the inverses.

3.2 Consequences

An important consequence of the above result is that with the introduction of factorized sparse approximate inverse in Algorithm 4 we can apply the whole theory available for incomplete ILU decompositions. As an example consider the stability of the ILU for H–matrices. It has well–known, that for any choice of $\mathcal{C}$, $\mathcal{R}$ the ILU decomposition of an H–matrix exists. See [19, 18] for details. It immediately follows that Z, W of the Algorithm 4 exist for this case. Likewise for M–matrices we know that the computed L, U are again M–matrices. Consequently Z, W have to be nonnegative in this case.

We obtain one immediate conclusion for the symmetric positive definite case.

Corollary 7 *Let A be symmetric positive definite. Suppose that Algorithm 4 and Algorithm 5 apply the same dropping rule to p, q and that no dropping is applied to W, Z. If p, q in Algorithm 4 are defined via (11) and if S in Algorithm 5 is defined via (7), then both Algorithms do not break down. In addition both methods compute the same W, Z and $W = Z$. The diagonal entries of S in Algorithm 5 are positive and coincide with the entries of $p = q$ in Algorithm 4.*

Proof. This follows immediately from Theorem 6 and Lemma 2. □

As mentioned earlier, Algorithm 4 is more general than the original $AINV$ algorithm [4, 5] which does not drop entries in the update factors from p, q but only to the updated matrices Z, W. The problem is that small entries $|p_j/p_i|$ may multiply large entries of $Z_{:,i}$ discarding entries in the approximate inverse that might be not small at all. To interpret $AINV$ as a form of ILU, the definition of the approximate Schur–complement must be adapted. So far the computation of the Schur–complement in Algorithm 1 corresponds to the definition in (7), (8). The key to get the connection between $AINV$ and an ILU or modified ILU is equation (12). Once we construct our Schur–complement such that this relation holds between $AINV$ and a modified ILU, we can immediately conclude that both algorithms compute comparable W, Z. By construction of the $AINV$ we cannot expect that W, Z will be inverses of some triangular factors L, U. But what we might expect is that $\|L^{-1} - W^\top\|$ is small, i.e. that W can be viewed as sparse approximation to the inverse of L. For this purpose consider an artificial algorithm, in which we just define the Schur–complement via (12). Due to this special definition of the Schur–complement we can only expect W to be close to $L^{-\top}$.

Theorem 8 *Suppose that in Algorithm 5 and Algorithm 4 dropping is performed according to some drop tolerance $\varepsilon \in (0, 1)$. To be more precise, suppose that in step i of Algorithm 5 an entry L_{ji} is discarded only if $|L_{ji}| \max\{1\} \cup \{|W_{ik}| : k < i\} \leqslant \varepsilon$ while no dropping is applied to p in Algorithm 4, $i = 1, \ldots, n$. Suppose that in both Algorithms $\mathcal{C}_i$ is chosen such that W_{kl} is dropped from $W_{1:i,i+1:n}$ if $|W_{kl}| \leqslant \varepsilon$. If the (modified) Schur–complement $S_{i+1:n,i+1:n}$ is defined via*

$$S_{i+1:n,i+1:n} = W_{:,i+1:n}^\top A_{:,i+1:n},$$

then for any $k > l$:

$$|(L^{-\top})_{kl} - W_{kl}| \leqslant \varepsilon(2(k-l)-1)$$

and the diagonal entries of D are those of p.

Proof. It is clear by construction that the diagonal entries of D at the end coincide with the components of p in Algorithm 4. Denote by $L_{i+1:n,i}$ the ith column of L before dropping is applied in step i of Algorithm 5. Then we have

$$\begin{aligned}(L^{-\top})^{(new)}_{1:i-1,i+1:n} &= (L^{-\top})^{(old)}_{1:i-1,i+1:n} - (L^{-\top})^{(old)}_{1:i-1,i}(L^\top_{i+1:n,i} + f^\top)\\ (L^{-\top})^{(new)}_{i,i+1:n} &= -L^\top_{i+1:n,i} - f^\top,\end{aligned}$$

where f denotes the error vector obtained by dropping entries of $L_{i+1:n,i}$ according to our dropping strategy. The matrix W in Algorithm 4 will be

$$W^{(new)}_{1:i-1,i+1:n} = W^{(old)}_{1:i-1,i+1:n} - W^{(old)}_{1:i-1,i} p^{(new)}_{i+1:n} + E$$

$$W_{i,i+1:n}^{(new)} \;=\; -p_{i+1:n}^{(new)} + e,$$

where E, e refer to the related error matrices. By construction we have $L_{i+1:n,i} = p_{i+1:n}^{(new)}$, since both algorithms compute the same W. It follows that

$$\left| W_{il}^{(new)} - (L^{-\top})_{il}^{(new)} \right| \leqslant \varepsilon, l > i$$

and

$$\left| W_{kl}^{(new)} - (L^{-\top})_{kl}^{(new)} \right| \leqslant \left| W_{kl}^{(old)} - (L^{-\top})_{kl}^{(old)} \right| + 2\varepsilon, k < i, l > i.$$

Continuing this estimate for $i = 1, 2, \ldots, n$ yields the desired result. □

Clearly one gets an analogous theorem for the relation between U^{-1} and Z.

An equivalent alternative to Algorithm 4, at least without dropping was suggested in [5] and was referred to as the SDS version of $AINV$. The method consists essentially of computing the approximate inverses Z, W column–wise instead of using rank–1 updates as in Algorithm 4.

Algorithm 9 (Factorized Approximate INVerse, column-wise update version)
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute $A^{-1} \approx ZD^{-1}W^{\top}$.

$p = q = \underbrace{(0, \ldots, 0)}_{n}, Z = W = D = I_n, p_1 = q_1 = A_{11}, \mathfrak{C} = \mathfrak{R} = \mathfrak{C}_i = \mathfrak{R}_i = \emptyset, i = 1, \ldots, n$

for $i = 2 : n$
  for $j = 1 : i-1$
    Define P_j, Q_j either by (13) or by (14)

$$P_j = A_{:,j}^{\top} W_{:,i}, \qquad Q_j = A_{j,:} Z_{:,i} \tag{13}$$
$$P_j = Z_{:,j}^{\top} A^{\top} W_{:,i}, Q_j = W_{:,j}^{\top} A Z_{:,i} \tag{14}$$

    Set $P_j = P_j/p_j, Q_j = Q_j/q_j$ and apply a dropping rule to P_j, Q_j.
    If P_j is discarded, $\mathfrak{R} := \mathfrak{R} \cup \{(i,j)\}$. If Q_j is discarded, $\mathfrak{C} := \mathfrak{C} \cup \{(j,i)\}$.
    $W_{:,i} = W_{:,i} - W_{:,j} P_j$, $Z_{:,i} = Z_{:,i} - Z_{:,j} Q_j$ and apply a dropping rule to $W_{1:j,i}, Z_{1:j,i}$.
    If W_{ki} is discarded, $\mathfrak{R}_i := \mathfrak{R}_i \cup \{(i,k)\}$. If Z_{li} is discarded, $\mathfrak{C}_i := \mathfrak{C}_i \cup \{(l,i)\}$.
  end
  Define $P_i = p_i, Q_i = q_i$ either by (13) or by (14) for $j = i$.
end
Choose diagonal entries of D as the components of p or q.

This algorithm is almost identical to Algorithm 4 except that the updates in Z, W are now performed subsequently, column by column while in Algorithm 4 the updates are performed simultaneously for all columns. The simple relation between both algorithms is stated in the following Proposition which can be verified in a straightforward fashion by adding the statement "If W_{ki} is dropped, $\mathcal{C}_k := \mathcal{C}_k \cup \{(i,k)\}$, if Z_{li} is dropped, $\mathcal{R}_l := \mathcal{R}_l \cup \{(l,i)\}$" to the inner loop of Algorithm 9 to get $\mathcal{C}_k, \mathcal{R}_l$ from Algorithm 4.

Proposition 10 *Suppose that in Algorithm 4 and Algorithm 9 we have $\mathcal{C} = \mathfrak{R} = \mathcal{R} = \mathfrak{C} = \emptyset$. Suppose in addition that the same dropping rule is applied to W, Z in both algorithms. Then Algorithm 4 and Algorithm 9 compute the same Z, W, p, q.*

In fact the equality between both algorithms also includes the case that each column is sparsified only once. For Algorithm 9 this would be the natural dropping rule, i.e., entries of $Z_{:,i}, W_{:,i}$ would be discarded only if $j = i - 1$. For step i of Algorithm 4 the associated dropping rule would sparsify only column $i + 1$ of Z, W which might lead to a giant fill–in for Z, W.

4 Incomplete factorizations via bordering

In this section we discuss the relation between approximate inverses obtained by a bordering technique, suggested in [21]. The main idea is to successively invert the leading main principal matrices of a given nonsingular matrix $A = (A_{ij})_{i,j=1,\ldots,n}$ by applying inverses of an upper and lower triangular matrix from both sides. In other words for

$$\begin{pmatrix} A_{1:k,1:k} & A_{1:k,k+1} \\ A_{k+1,1:k} & A_{k+1,k+1} \end{pmatrix} = \begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

and $k = 1, \ldots, n-1$ a problem of the following form is considered:

$$\begin{pmatrix} W_B & W_E \\ 0 & 1 \end{pmatrix}^\top \begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} Z_B & Z_F \\ 0 & 1 \end{pmatrix} \approx \begin{pmatrix} D_B & 0 \\ 0 & S \end{pmatrix}$$

where $Z_B, W_B^\top$ are upper triangular matrices with unit diagonal and D_B is diagonal. Suppose that approximate solutions Z_B, D_B, W_B have already been computed, then we can obtain Z_F, S, W_E from the equations

$$\begin{aligned} B^\top W_E &= -E^\top \\ B Z_F &= -F \\ S &= C + W_E^\top F + E Z_F + W_E^\top B Z_F. \end{aligned} \tag{15}$$

Instead of solving a system with B, the relation $W_B^\top B Z_B \approx D_B$ is exploited to approximate Z_F, W_E by discarding some entries according to a dropping rule. To allow for more generality, we will consider four different approximate Schur–complements in analogy with Algorithm 12.

Algorithm 11 (Factorized Approximate Inverse Using Bordering)
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute a factorized approximate inverse $A^{-1} \approx Z D^{-1} W^\top$.

$Z = W = D = I_n, D_{11} = A_{11}, \mathfrak{R} = \mathfrak{C} = \emptyset.$

for $i = 2 : n$

$P_{1:i-1} = D^{-1}_{1:i-1,1:i-1} Z^\top_{1:i-1,1:i-1} A^\top_{i,1:i-1}$; $Q_{1:i-1} = D^{-1}_{1:i-1,1:i-1} W^\top_{1:i-1,1:i-1} A_{1:i-1,i}$

Apply a dropping rule to $P_{1:i-1}, Q_{1:i-1}$

If P_j is dropped, $\mathfrak{R} := \mathfrak{R} \cup \{(i,j)\}$. If Q_j is dropped, $\mathfrak{C} := \mathfrak{C} \cup \{(j,i)\}$

$W_{1:i-1,i} = -W_{1:i-1,1:i-1} P_{1:i-1}$, $Z_{1:i-1,i} = -Z_{1:i-1,1:i-1} Q_{1:i-1}$

Apply a dropping rule to $W_{1:i-1,i}, Z_{1:i-1,i}$

Let $\mathfrak{R}_i$ be the set of all (i,k) for which W_{ki} has been dropped.

Let $\mathfrak{C}_i$ be the set of all (l,i) for which Z_{li} has been dropped.

Compute D_{ii} from one of the following options.

$$D_{ii} = A_{ii} - P^\top_{1:i-1} A_{1:i-1,1:i-1} Q_{1:i-1} \tag{16}$$

$$D_{ii} = A_{ii} + W^\top_{1:i-1,i} A_{1:i-1,i} + A_{i,1:i-1} Z_{1:i-1,i} + W^\top_{1:i-1,i} A_{1:i-1,1:i-1} Z_{1:i-1,i} \tag{17}$$

$$D_{ii} = A_{ii} + W^\top_{1:i-1,i} A_{1:i-1,i} \tag{18}$$

$$D_{ii} = A_{ii} + A_{i,1:i-1} Z_{1:i-1,i} \tag{19}$$

end

Essentially the same technique to compute a banded factorized approximate inverse for a banded symmetric positive definite matrices can already be found in [16]. A generalization to the general case but without sparsifying is given in [17]. The main difference is that the roles of Z and W are interchanged and that the initial matrix is overwritten.

Like in the case of Section 3 we cannot expect to have an exact relation between Algorithm 11 as it stands now and an ILU. For this reason we have to simplify the dropping strategy, i.e. we first consider the case $\mathfrak{C}_i = \mathfrak{R}_i = \emptyset$. It is interesting that with this restriction we can immediately find an ILU such that the inverses correspond to those computed by Algorithm 11. The ILU we use for comparison slightly differs from the one in the introduction and we will refer to it as $ILUS$, since this algorithm [6, 21], was designed for matrices stored in a so–called skyline format.

The main idea is to compute an (approximate) LU decomposition of each leading main principal matrix of A, i.e. one considers equation (2) with A replaced by $A_{1:i,1:i}$ and $k = i - 1$. In other words, the Schur–complement in (2) is only a number.

Algorithm 12 (Incomplete Skyline LU factorization ($ILUS$))
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. Compute an $ILUS$ factorization $A \approx LDU$.

$L = D = U = I_n, D_{11} = A_{11}, \mathfrak{R} = \mathfrak{C} = \emptyset$

for $i = 2 : n$

$L_{i,1:i-1} = A_{i,1:i-1} U^{-1}_{1:i-1,1:i-1} D^{-1}_{1:i-1,1:i-1}$; $U_{1:i-1,i} = D^{-1}_{1:i-1,1:i-1} L^{-1}_{1:i-1,1:i-1} A_{1:i-1,i}$

Apply a dropping rule to $L_{i,1:i-1}, U_{1:i-1,i}$.

If L_{ij} is dropped, $\mathfrak{R} := \mathfrak{R} \cup \{(i,j)\}$. If U_{ji} is dropped, $\mathfrak{C} := \mathfrak{C} \cup \{(j,i)\}$

Compute D_{ii} according to one of the possible choices from (6)–(9), i.e. compute D_{ii} from one of the following options.

$$D_{ii} = \begin{cases} A_{ii} - L_{i,1:i-1} D_{1:i-1,1:i-1} U_{1:i-1,i} \\ \Big[A_{ii} - L_{i,1:i-1} L^{-1}_{1:i-1,1:i-1} A_{1:i-1,i} - \\ \qquad \left(A_{i,1:i-1} - L_{i,1:i-1} L^{-1}_{1:i-1,1:i-1} A_{1:i-1,1:i-1} \right) U^{-1}_{1:i-1,1:i-1} U_{1:i-1,i} \Big] \\ A_{ii} - L_{i,1:i-1} L^{-1}_{1:i-1,1:i-1} A_{1:i-1,i} \\ A_{ii} - A_{i,1:i-1} U^{-1}_{1:i-1,1:i-1} U_{1:i-1,i} \end{cases}$$

end

In [21] $D_{ii} = A_{ii} - L_{i,1:i-1} D_{1:i-1,1:i-1} U_{1:i-1,i}$ is used. In addition dropping is performed with respect to a different strategy, since solving systems with $L_{1:i-1,1:i-1}, U_{1:i-1,1:i-1}$ becomes very expensive as i increases. But for our investigation on the relation between approximate inverses and incomplete LU–decompositions we only need this strategy. It is obvious that the four options of Algorithm 12 correspond to (6)–(9) by considering only the leading $i \times i$ block of A.

Theorem 13 *Assume that for any $i = 2, \ldots, n$, $\mathfrak{R}_i = \mathfrak{C}_i = \emptyset$. Suppose that Algorithm 12 and Algorithm 11 use the same dropping rule to L, U and P, Q, i.e. the same sets $\mathfrak{C}, \mathfrak{R}$ are generated. Assume that in step i of both algorithms, D_{ii} is chosen the same way, i.e. in step i we have for D_{ii} the choices*

Algorithm 12	*(6)*	*(7)*	*(8)*	*(9)*
Algorithm 11	*(16)*	*(17)*	*(18)*	*(19)*

Then Algorithm 5 and Algorithm 11 compute the same matrix D and

$$L^{-\top} = W, \quad U^{-1} = Z.$$

Proof. We use induction on i. Suppose that $L^{-\top}_{1:i,1:i} = W_{1:i,1:i}$, $U^{-1}_{1:i,1:i} = Z_{1:i,1:i}$ and that both algorithms have computed the same D_{ii}. Obviously this is true for $i = 1$. Now row

$i+1$ of L in Algorithm 12 will be $p^\top := A_{i+1,1:i}U^{-1}_{1:i,1:i}D^{-1}_{1:i,1:i}$ after sparsifying p with respect to some dropping rule. In the same way we obtain $q := D^{-1}_{1:i,1:i}L^{-1}_{1:i,1:i}A_{1:i,i+1}$ for $U_{1:i,i+1}$.

On the other hand we will compute precisely the same values $P_{1:i} = p, Q_{1:i} = q$ in Algorithm 11. Moreover dropping is applied to the same vectors with the same dropping rules. Let us call $\hat{p}, \hat{q}$ the sparsified vectors. It follows that

$$L_{1:i+1,1:i+1} = \begin{bmatrix} L_{1:i,1:i} & 0 \\ \hat{p}^\top & 1 \end{bmatrix}, \; U_{1:i+1,1:i+1} = \begin{bmatrix} U_{1:i,1:i} & \hat{q} \\ 0 & 1 \end{bmatrix}$$

and

$$L^{-\top}_{1:i+1,1:i+1} = \begin{bmatrix} W_{1:i,1:i} & -W_{1:i,1:i}\hat{p} \\ 0 & 1 \end{bmatrix}, \; U^{-1}_{1:i+1,1:i+1} = \begin{bmatrix} Z_{1:i,1:i} & -Z_{1:i,1:i}\hat{q} \\ 0 & 1 \end{bmatrix}.$$

But this is precisely the way Z, W are defined in Algorithm 11 and since there is no additional sparsification done we have that $L^{-\top}_{1:i+1,1:i+1} = W_{1:i+1,1:i+1}$, $U^{-1}_{1:i+1,1:i+1} = Z_{1:i+1,1:i+1}$. Finally, for any choice of $D_{i+1,i+1}$ mentioned above we immediately obtain that both algorithms compute the same value. □

This result between Algorithm 11 and Algorithm 12 is the perfect analogy to the relation between Algorithm 4 and Algorithm 1. This analogy seems to be reasonable since Algorithm 4 and Algorithm 11 combined with a simplified dropping strategy are set up in an analogous way to Algorithm 1 and Algorithm 12.

There is also a surprisingly simple connection between Algorithm 11 and Algorithm 9. Clearly we should require that $\mathfrak{R} = \mathfrak{C} = \emptyset$ in both cases since otherwise this leads to a connection between Algorithm 11 and Algorithm 12 but also between Algorithm 9 and Algorithm 1 (i.e., in case both algorithms would refer to different incomplete LU–decompositions). Since Algorithm 9 can in principle compute Z, W independently of each other, at least for the two specific cases (8),(9) it is clear that we only have to consider the case when one of $\mathfrak{R}_i, \mathfrak{C}_i$ is empty.

Theorem 14 *Consider Algorithm 9, and Algorithm 11 where no sparsifying is applied to P, Q ($\mathfrak{R} = \mathfrak{C} = \emptyset$). Assume that in Algorithm 9, $\mathfrak{R}_i = \mathfrak{C}_i = \emptyset$ for any step $j < i-1$, i.e. sparsifying is done only once per column in the final step $j = i-1$.*
Assume, in addition, that Algorithm 9 applies the same dropping rule to $W_{1:i-1,i}, Z_{1:i-1,i}$ in step $j = i-1$ as Algorithm 11.

If Algorithm 9 computes P by (13) and if Algorithm 11 computes D_{ii} by (18), then both algorithms compute the same W, D, provided that no dropping is applied to Z (same $\mathfrak{R}_i, \; \mathfrak{C}_i = \emptyset, i = 2, \ldots, n$).

If Algorithm 9 computes Q by (13) and if Algorithm 11 computes D_{ii} by (19), then both algorithms compute the same Z, D, provided that no dropping is applied to W (same $\mathfrak{C}_i$, $\mathfrak{R}_i = \emptyset, i = 2, \ldots, n$).

Proof. It suffices to show only the relation for the matrix W. To avoid confusion, we note that during this proof we always refer to P as the choice of P in Algorithm 9. In Algorithm 9 we introduce an auxiliary matrix $\hat{Z}$ which will be initialized as $\hat{Z} = I_n$ during each step of Algorithm 9 we compute beside P_j the auxiliary value

$$\hat{P}_j = A_{:,i}^\top W_{:,j}.$$

As for P_j we divide $\hat{P}_j$ by p_j, i.e. $\hat{P}_j := \hat{P}_j / p_j$. Then we update $\hat{Z}$ in any step by

$$\hat{Z}_{:,i} = \hat{Z}_{:,i} - \hat{Z}_{:,j} \hat{P}_j$$

and no sparsifying is done. We will show that $\hat{Z}$ of Algorithm 9 corresponds to Z in Algorithm 11. Suppose that at some step i, both matrices have computed the same $W_{1:i,1:i}$, that the diagonal entries of $D_{1:i,1:i}$ of Algorithm 11 coincide with first i components of p. Finally assume that $\hat{Z}_{1:i,1:i}$ of Algorithm 9 is associated with $Z_{1:i,1:i}$ from Algorithm 11. This is obviously true for $i = 1$. Algorithm 9 updates column i of W, Z. Initially we have $W_{:,i} = e_i, Z_{:,i} = e_i$. Denote by $P_{1:j}, \hat{P}_{1:j}$ the column vectors with components $P_1, \ldots, P_j$, $\hat{P}_1, \ldots, \hat{P}_j$. When we refer to $P_j, \hat{P}_j$ we mean these variables after the division by p_j.

Since $\hat{Z}_{:,i}$ is not referenced during the inner loop we immediately have

$$\hat{P}_{1:i-1} = D_{1:i-1,1:i-1}^{-1} W_{1:i-1,1:i-1}^\top A_{1:i-1,i}$$

and

$$\hat{Z}_{1:i-1,i} = -\hat{Z}_{1:i-1,1:i-1} \hat{P}_{1:i-1} = -\hat{Z}_{1:i-1,1:i-1} D_{1:i-1,1:i-1}^{-1} W_{1:i-1,1:i-1}^\top A_{1:i-1,i}$$

This is already precisely the way $Z_{1:i-1,i}$ is defined in Algorithm 11.

We will show that for any $j = 1, \ldots, i-1$ we will have

$$P_j = A_{i,:} \hat{Z}_{:,j} / p_j, \tag{20}$$

This is obviously true for $j = 1$. Now suppose that this formula is correct for $1, \ldots, j-1$. In this case we obtain

$$\hat{Z}_{1:j,j} \;=\; \begin{bmatrix} -\hat{Z}_{1:j-1,1:j-1} D_{1:j-1,1:j-1}^{-1} W_{1:j-1,1:j-1}^\top A_{1:j-1,j} \\ 1 \end{bmatrix}$$

and

$$A_{i,:} \hat{Z}_{:,j} = A_{ij} - A_{i,1:j-1} \hat{Z}_{1:j-1,1:j-1} D_{1:j-1,1:j-1}^{-1} W_{1:j-1,1:j-1}^\top A_{1:j-1,j}. \tag{21}$$

Denote by $W_{:,i}^{(j)}$ the updated version of $W_{:,i}$ after step j. Starting with $W_{:,i}^{(0)} = e_i$ we find that

$$W_{:,i}^{(j)} = e_i - W_{:,1:j-1} P_{1:j-1}.$$

We compute P_j by substituting the latest update $W_{:,i}^{(j)}$ of $W_{:,i}$ by $e_i - W_{:,1:j-1} P_{1:j-1}$ and get

$$\begin{aligned} A_{:,j}^\top W_{:,i}^{(j-1)} &= A_{:,j}^\top \left(e_i - W_{:,1:j-1} P_{1:j-1} \right) \\ &= A_{ij} - A_{1:j-1,j}^\top W_{1:j-1,1:j-1} P_{1:j-1} \end{aligned}$$

But since

$$P_{1:j-1}^\top = A_{i,1:j-1} \hat{Z}_{1:j-1,1:j-1} D_{1:j-1,1:j-1}^{-1}$$

we finally end up with

$$A_{:,j}^\top W_{:,i}^{(j-1)} = A_{ij} - A_{1:j-1,j}^\top W_{1:j-1,1:j-1} D_{1:j-1,1:j-1}^{-1} \hat{Z}_{1:j-1,1:j-1}^\top A_{i,1:j-1}^\top. \tag{22}$$

Fortunately formula (22) for $A_{:,j}^\top W_{:,i}^{(j-1)}$ matches with formula (21) for $A_{i,:} \hat{Z}_{:,j}$ which proves (20).

After we have established (20) we obtain that both algorithms compute the same $P_{1:i-1}$ and therefore both Algorithms compute the same

$$W_{1:i-1,i} = -W_{1:i-1,1:i-1} P_{1:i-1}.$$

Since we apply sparsification at the same place to the same vectors, these vectors will coincide after being sparsified. It is clear by construction that D_{ii} and p_i will also be the same. □

Roughly spoken we have to apply Algorithm 11 with sparsifying either only in W or only in Z to obtain the same W/Z as in Algorithm 9. In this case for both Algorithms, sparsifying is done only once at the end for each column. The key relation between both algorithms here is (20). It gives us a dual representation of the coefficients P during the update of $W_{:,i}$ that does not depend on the updated vectors of $W_{:,i}$.

5 Conclusions

We have shown a number of inter-relations between factorized approximate inverse and related incomplete factorizations of the ILU type. We also established relations between different approaches to compute factorized approximate inverses. It is an interesting fact

to see that approximate inverse methods are intimately related to ILU factorizations. They can be viewed as a process for obtaining the inverses of the L and U factors directly from the elementary subfactors that arise in Gaussian elimination. What is interesting, is that with an appropriate set of assumptions on the patterns used for dropping, many other relationships can be established. This equivalence permits to establish some results on existence and, more generally, to better understand the algorithms. For example, it is now clear that ILU and $AINV$ factorizations are two extremes where elementary factors are all inverted (in $AINV$) or kept are they are (in ILUs) – but it is clear that there is a sea of variation in between these extremes and it is quite conceivable that better methods would be adaptive algorithms that lie in between – where adaptivity here is understood in relation to stability.

References

[1] MATLAB – The language of technical computing. The MathWorks Inc., 1996.

[2] O. Axelsson. *Iterative Solution Methods.* Cambridge University Press, New York, 1994.

[3] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. Technical report LA–UR–99–2899, Los Alamos National Laboratory, Scientific Computing Group, 1999.

[4] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.

[5] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998.

[6] E. Chow and Y. Saad. ILUS: an incomplete LU factorization for matrices in sparse skyline format. *International Journal for Numerical Methods in Fluids*, 25:739–748, 1997.

[7] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Statist. Comput.*, 19:995–1023, 1998.

[8] A. C. V. Duin. Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices. Preprint, Rijksuniversiteit Leiden, Department of Computer Science, 1997.

[9] G. Golub and C. V. Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.

[10] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* Frontiers in Applied Mathematics. SIAM Publications, 1997.

[11] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3), 1997.

[12] W. Hackbusch. *Iterative Solution of Large Linear Systems of Equations.* Springer Verlag, New York, 1994.

[13] I. E. Kaporin. New convergence results and preconditioning strategies for conjugate gradient method. *Numer. Lin. Alg. w. Appl.*, 1(2):179–210, 1994.

[14] S. Kharchenko, L. Kolotilina, A. Nikishin, and A. Yeremin. A reliable AINV–type preconditioning method for constructing sparse approximate inverse preconditioners in factored form. Technical report, Russian Academy of Sciences, Moscow, 1999.

[15] Y. Kolotilina and Y. Yeremin. Factorized sparse approximate inverse preconditionings I. theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.

[16] J.-C. Luo. An incomplete inverse as a preconditioner for the conjugate gradient method. *Computer & Math. w. Appl.*, 25(2):73–79, 1993.

[17] J.-C. Luo. A new class of decomposition for inverting asymmetric and indefinite matrices. *Computer & Math. w. Appl.*, 25(4):95–104, 1993.

[18] T. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–490, 1980.

[19] J. Meijerink and H. A. V. der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m–matrix. *Math. Comp.*, 31:148–162, 1977.

[20] Y. Saad. ILUT: a dual treshold incomplete ILU factorization. *Numer. Lin. Alg. w. Appl.*, 1:387–402, 1994.

[21] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.