On the relations between ILUs and factored approximate inverses

*Matthias Bollhöfer and †Yousef Saad

June 21, 2001

Abstract

This paper discusses some relationships between Incomplete LU (ILU) factorization techniques and factored sparse approximate inverse (AINV) techniques. While ILU factorizations compute approximate LU factors of the coefficient matrix A, AINV techniques aim at building triangular matrices Z and W such that $W^\top AZ$ is approximately diagonal. The paper shows that certain forms of approximate inverse techniques amount to approximately inverting the triangular factors obtained from some variants of incomplete LU factorization of the original matrix. A few useful, and already known, applications of these relationships will be overviewed.
Keywords: sparse matrices, Incomplete LU (ILU), variants of ILU, sparse approximate inverse, AINV.
AMS subject classification: 65F05, 65F10, 65F50.

1 Introduction

Preconditioned Krylov–subspace iterations are among the most efficient techniques for solving linear systems of the form:

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

where $A \in \mathbb{R}^{n,n}$ is nonsingular and $b \in \mathbb{R}^n$ is a given right hand side, see e.g., [23, 13, 1, 15]. Among the most popular preconditioners are those based on approximate factorizations obtained from direct solution methods, such as the LU factorization [12],

*Institute of Mathematics, MA 4–5, Berlin University of Technology, D–10623 Berlin, Germany. Work 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@math.tu-berlin.de, URL: http://www.math.tu-berlin.de/∼bolle/.

†Department of Computer Science and Engineering, University of Minnesota, 4–192 EE/CSci Building, 200 Union St., SE, Minneapolis, MN 55455–0154. Work supported by the U.S. Army Research Office under contract DAAD19-00-1-0485, and by the Minnesota Supercomputing Institute. email: saad@cs.umn.edu, URL: http://www.cs.umn.edu/∼saad/

pp. 92ff. Alternative techniques appeared in recent years which compute approximate solutions of (1) via an approximate inverse of A, instead of a factorization. One of the main motivations for using preconditioners of this type is parallelism. Another important reason is that ILU preconditioners, which have been developed for M matrices [20], often fail for indefinite matrices.

A few of the approximate inverse techniques are based on minimizing $\|I - AM\|$ in some appropriate norm [18, 16, 14, 9]. Others compute the approximate inverse in factored form by seeking two sparse unit upper triangular matrices W and Z, and a diagonal D, such that $W^\top AZ \approx D$, see e.g. [3, 5, 2, 17, 23]. As it turns out, the latter class of preconditioners show an algebraic behavior that is similar to that of the well-known incomplete LU decompositions. For example, they are stable for M– and H–matrices, in perfect analogy with known results on incomplete LU decompositions in [20, 19].

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 [11]. However, our paper does not consider these methods.

The purpose of this paper is to take an in-depth look at the relationships between factored approximate inverse preconditioners (AINV) and incomplete LU decomposition methods. In particular, it will be shown that AINV methods generate factors which can be viewed as approximations of the inverses of the triangular factors obtained by certain variants of incomplete LU. Using 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 factored approximate inverse techniques.

2 Incomplete LU factorizations

Incomplete LU factorizations construct approximate L, D, U factors of A such that

$$A \approx LDU$$

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

$$\begin{bmatrix} a_{11} & f \\ e & C \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ g & I \end{bmatrix} \begin{bmatrix} \delta & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} 1 & h \\ 0 & I \end{bmatrix}, \tag{2}$$

with $\delta = a_{11}$. The terms δ, g, h and S satisfy $g\delta = e \in \mathbb{R}^{n-1,1}$, $\delta h = f \in \mathbb{R}^{1,n-1}$ and

$$S = C - g\ \delta\ h \in \mathbb{R}^{n-1,n-1}\ . \tag{3}$$

The matrix S denotes the so–called Schur–complement. An exact LU decomposition is obtained by applying (2) recursively on the resulting Schur–complement. The process is completed by substituting the factorization $S = L_S D_S U_S$, when it exists, into (2) to obtain

$$\begin{bmatrix} a_{11} & f \\ e & C \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ g & L_S \end{bmatrix} \begin{bmatrix} \delta & 0 \\ 0 & D_S \end{bmatrix} \begin{bmatrix} 1 & h \\ 0 & U_S \end{bmatrix}, \tag{4}$$

which is the final LU factorization.

In incomplete factorizations, entries are dropped during this procedure in the L, U factors and in the Schur–complement. A common strategy is to drop entries in the first column of L according to a certain "dropping rule" and apply a similar dropping rule to the first row of U. As a result of this procedure, the row $h = \delta^{-1} f$, and column $g = e\delta^{-1}$ are replaced by sparsified approximations

$$\tilde{h} \approx h \qquad \tilde{g} \approx g$$

leading to the approximate Schur complement

$$\tilde{S} = C - \tilde{g}\ \delta\ \tilde{h} \tag{5}$$

which is a sparsified version of (3). However, there are several other ways of defining approximate Schur complements from approximations to g and h. For example, we can multiply both sides of (2) to the left by the inverse of the approximate L factor obtained by replacing g by $\tilde{g}$. Equating the resulting $(2,2)$ blocks leads to

$$\tilde{S} = C - \tilde{g}\ f\ . \tag{6}$$

From an algorithmic point of view, the process amounts to multiplying the current matrix, i.e., the matrix on the left-hand-side of (2), to the left by

$$\begin{bmatrix} 1 & 0 \\ -\tilde{g} & I \end{bmatrix}.$$

In other words the next Schur complement is obtained by performing the usual row operations in Gaussian elimination using a sparsified version of L, obtained by dropping some elements.

Similarly, a column-based version of this process consists of multiplying both sides of (2) to the right by the inverse of of the approximate U factor obtained by replacing h by $\tilde{h}$. This leads to the approximation

$$\tilde{S} = C - e\ \tilde{h}. \tag{7}$$

A fourth option we mention consists of a combination of these two operations. First, operate with the approximation to the inverse of L to the left of the matrix A, and then

operate with the approximation to the inverse of U to the right of the resulting matrix. The $(2,2)$ block of the resulting matrix is the Schur complement

$$\tilde{S} = C - \tilde{g}\ f - (e - \tilde{g}\ \delta)\,\tilde{h}. \tag{8}$$

Other ways of defining an approximate Schur–complement can be derived from other equivalent expressions of the Schur–complement. In the case of an exact factorization (no dropping) the update formulas (5), (6), (7), and (8), will all lead to the same S. In practice, (5) is the most common scheme for defining Incomplete LU factorizations, see, e.g. [20] or [22]. Typically, (5) produces the smallest amount of fill–in compared with the other formulas. The update (8) has also been used in a number of papers [24, 2, 6, 8]. In the symmetric positive definite case, it is guaranteed to produce a stable ILU factorization, see [24].

2.1 Update variants

In order to simplify the description of the algorithms to be considered we make the following observation which allows us to express all four types of updates just described in a concise manner. Consider, for example, the update (5). The update for entry (i,j) of C is performed only when $\tilde{g}_i$ and $\tilde{h}_j$ are both nonzero, i.e., when their original terms in g and h have both not been dropped. We now notice that if we call S the current Schur complement matrix, i.e., the matrix on the left-hand-side of (2), then (5) is equivalent to performing the following update for each pair (i,j) such that $s_{ik} \cdot s_{kj} \neq 0$:

$$s_{ij} = s_{ij} - \frac{s_{ik} s_{kj}}{d_{kk}} \tag{9}$$

but to restrict this update to the cases when g_i and h_j have both not been dropped. Thus, (5) can be expressed as "Perform (9) when $\tilde{g}_i \neq 0$ and $\tilde{h}_j \neq 0$". Interestingly, each of (5), (6), (7), and (8), can be expressed in this manner.

- Update (5): Perform (9) when $\tilde{g}_i \neq 0$ and $\tilde{h}_j \neq 0$
- Update (6): Perform (9) when $\tilde{g}_i \neq 0$
- Update (7): Perform (9) when $\tilde{h}_j \neq 0$
- Update (8): Perform (9) when $\tilde{g}_i \neq 0$ or $\tilde{h}_j \neq 0$.

A little explanation is required for the last case. If $\tilde{g}_i \neq 0$ and $\tilde{h}_j = 0$, then the formula will coincide with (6), which is the same as (9) for this particular situation. Similarly for the opposite case when $\tilde{g}_i = 0$ and $\tilde{h}_j \neq 0$, which leads to Formula (7). When both $\tilde{g}_i$ and $\tilde{h}_j$ are nonzero, then the term $c_{ij} - \tilde{g}_i f_j - e_i \tilde{h}_j$ can be viewed as a the term s_{ij} which has undergone two updates, one of which is extraneous. Therefore we need to correct this update by adding $\tilde{g}_i \delta \tilde{h}_j$.

Throughout the paper we will use the above formalism, i.e., all updates (5 – 8) will be expressed in the form "*if* version$(\tilde{g}_i, \tilde{h}_j)$ *then perform update (9)*", in which version$(\tilde{g}_i, \tilde{h}_j)$ is a boolean function which takes the following values for the four different cases under consideration:

- Update (5): version$(\tilde{g}_i, \tilde{h}_j) = \{\ \tilde{g}_i \neq 0$ and $\tilde{h}_j \neq 0\ \}$
- Update (6): version$(\tilde{g}_i, \tilde{h}_j) = \{\ \tilde{g}_i \neq 0\ \}$
- Update (7): version$(\tilde{g}_i, \tilde{h}_j) = \{\ \tilde{h}_j \neq 0\ \}$
- Update (8): version$(\tilde{g}_i, \tilde{h}_j) = \{\ \tilde{g}_i \neq 0$ or $\tilde{h}_j \neq 0\ \}$.

2.2 Block versions

It is useful to generalize the above arguments to the case when the (1,1) term a_{11} in (2) is replaced by a block B of size $k \times k$ of the matrix A, with $k > 1$. The partial LU factorization, when it exists, is now expressed by

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \begin{bmatrix} L_B & 0 \\ G & I \end{bmatrix} \begin{bmatrix} D_B & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} U_B & H \\ 0 & I \end{bmatrix}, \tag{10}$$

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. Here, L_B, D_B, U_B refer to an already computed LU–decomposition of B. The matrices D_B, G, H and S satisfy $GD_BU_B = E \in \mathbb{R}^{n-k,k}$, $L_BD_BH = F \in \mathbb{R}^{k,n-k}$ and the Schur complement now becomes

$$S = C - GD_BH \in \mathbb{R}^{n-k,n-k} \tag{11}$$

The same four versions of the approximate Schur complement as those defined by (5), $\cdots$, (8) can be defined similarly. We list them all below for future reference.

$$\begin{aligned} \tilde{S} &= C - \tilde{G}D_B\tilde{H} && (12)\\ \tilde{S} &= C - \tilde{G}L_B^{-1}F && (13)\\ \tilde{S} &= C - EU_B^{-1}\tilde{H} && (14)\\ \tilde{S} &= C - \tilde{G}L_B^{-1}F - \left(E - \tilde{G}L_B^{-1}B\right)U_B^{-1}\tilde{H} && (15) \end{aligned}$$

At this point we make an important observation regarding the approximate Schur–complement. For convenience we call the p-th Schur complement, the Schur complement obtained by eliminating unknowns $i = 1, \ldots, p$. The zero-th Schur complement is, by definition, the original matrix and the $(i+1)$-st Schur complement can be obtained by applying (10),(11) to the i-th Schur complement. When dropping is applied, the p-th Schur complement, a matrix of size $n - p$, will vary depending on which of the four

formulas (12),$\cdots$, (15) is used. Instead of this p-step procedure, we could alternatively obtain an approximate Schur–complement directly by using one step of the above process with $k = p$, taking the same equations from (12)–(15). The important property which we point out is that these two methods would lead to the same approximate Schur–complement.

Property 1 *The p-th (approximate) Schur complement S obtained from applying p consecutive steps of one of the four formulas (12 – 15) with $k = 1$, is identical with the (approximate) p-th Schur complement obtained from 1 step of the same formula among (12 – 15), with $k = p$.*

2.3 Dropping Strategies

There are two broad classes of dropping strategies. In the first category there are strategies, which drop elements based only on the pattern of the matrix. This includes the level–of–fill strategy [20]. A second category of methods drops elements dynamically, based on their magnitude [21, 22] Other strategies combine graph based methods with threshold dropping.

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 a dropping rule applied to the entries $g_{i,k}$ of G uses only information on $g_{i,k}$ and the related coordinates (i, k). Possible dropping rules of this type could be for example

- drop $g_{i,k}$ if $|g_{i,k}| \leqslant \tau$,
- drop $g_{i,k}$ if (i, k) is outside a specific pattern,
- drop $g_{i,k}$ if $|g_{i,k}| \leqslant \tau \|e_i^\top A\|$,

where τ is a fixed drop tolerance.

For more complex dynamic dropping, different versions of Gaussian elimination may produce different ILU factors even if the corresponding exact Gaussian elimination versions would produce the same factors. This is because the dropping strategies may yield different patterns. In general, threshold-based methods are harder to analyze than pattern-based algorithms.

2.4 K,I,J implementations

A sample routine for performing an incomplete LU decomposition is given by Algorithm 2. Algorithm 2 is based on the so-called K, I, J version (or 'rank-one' update version)

of Gaussian elimination. We make use of our earlier observation on a unified way to handle the approximate Schur–complements (12), (15), (13) and (14) in Algorithm 2. The different updates $s_{ij} = s_{ij} - \frac{s_{ik}s_{kj}}{d_{kk}}$ of the approximate Schur–complement can be expressed in terms of a logical value version$(\tilde{g}_i, \tilde{h}_j)$ which were defined earlier. The notation changes in the algorithm and the variables $\tilde{g}_i$ and $\tilde{h}_j$ are now called p_i and q_j.

Algorithm 2 (Incomplete LU factorization (ILU))

Input: $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$. *Output: ILU factorization* $A \approx LDU$.

0. $p = q = 0 \in \mathbb{R}^n$, $L = U = I$, $S = A$.
1. ***for*** $k = 1, \ldots, n$
2. $d_{kk} = s_{kk}$
3. ***for*** $i = k+1, \ldots, n$ *and when* $s_{ik} \neq 0$ *or* $s_{ki} \neq 0$
4. $p_i = s_{ik}/d_{kk}, \quad q_i = s_{ki}/d_{kk}$
5. *Apply a dropping rule to* p_i *and* q_i
6. $l_{ik} = p_i, \quad u_{ki} = q_i$
7. ***for*** $j = k+1, \ldots, n$ *and when* $s_{ik} \neq 0$ *and* $s_{kj} \neq 0$
8. ***if*** version(p_i, q_j) *then:* $s_{ij} = s_{ij} - \frac{s_{ik}s_{kj}}{d_{kk}}$
9. ***end***
10. ***end***
11. ***end***

A significant drawback of Algorithm 2 lies in its practical implementation. Each step of the procedure alters rows $k+1$ to n of the matrix S, which is typically held in a single data structure. This leads to the use of expensive linked lists, or elbow room. In spite of these drawbacks the algorithm is attractive for several reasons, and it has been used by a few authors to develop incomplete factorizations [10, 25]. One of its advantages is the ease with which powerful pivoting and reordering strategies can be implemented. The next section describes a different implementation which consists of swapping the k and i loops in Algorithm 2.

2.5 I,K,J variants of ILU

A more common alternative to implement incomplete LU factorizations is based on the I, K, J version of Gaussian elimination. This is sketched in Algorithm 3.

When the same static dropping strategy is used, e.g., one that is based on level-of-fill, it is known that Algorithm 3 and Algorithm 2, with S defined by (12), will deliver the same factors. However this relation is still true for dynamic dropping strategies, if the dropping rule is applied in the same way. Recall that Algorithms 3 and 2 perform the same sequence of operations in a different order. If an element is dropped in one it will also dropped in the other if the exact same criterion is applied. For this to be true one should be careful that the same rule is applied for partial results in the factorizations.

Algorithm 3 (Incomplete LU factorization (ILU))

Input: $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$. *Output: ILU factorization* $A \approx LDU$.

0. $L = D = U = I$.
1. ***for*** $i = 1, \ldots, n$
2. $w = e_i^\top A$
3. ***for*** $k = 1, \ldots, i-1$ *and when* $w_k \neq 0$
4. $w_k = w_k / d_{kk}$
5. *Apply a rule to drop* w_k.
6. ***for*** $j = k+1, \ldots, n$ *and if* ($w_k \neq 0$ *and* $u_{kj} \neq 0$)
7. $w_j := w_j - w_k u_{kj}$
8. ***end***
9. ***end***
10. $d_{ii} = w_i$
11. ***for*** *all* $j < i$: $l_{ij} = w_j$
12. ***for*** *all* $j > i$: $u_{ij} = w_j / w_i$. *Apply a dropping rule to* u_{ij}
13. ***end***

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 implementation in Algorithm 3 is clear: the rows of L and U are determined one at a time and are easily added to the existing data-structure.

3 Relations between AINV and ILUs

There are two broad classes of approximate inverse methods. The first includes methods which compute directly an approximate inverse M to A, see, e.g., [9, 14]. The second includes those methods which obtain this approximate inverse in the form of a product of two triangular factors. A method in this category, called AINV, was proposed in [3, 5]. It is briefly outlined next.

3.1 Factored Approximate Inverse (AINV)

The method in [3, 5] computes a decomposition of the form $W^\top AZ = D$, where W, Z are unit upper triangular matrices, and D is a diagonal. In the exact factorization case, the matrices W and Z are the inverses of the factors $L^\top$ and U, respectively, of the standard LDU decomposition $A = LDU$, when this decomposition exists. The matrices W and Z can be directly computed by a biorthogonalization procedure. Indeed, since

$$W^\top A = DU$$

is upper triangular, we immediately get $e_i^\top W^\top A e_j = 0$, for any $j < i$, which means that column i of W is orthogonal to the first $i-1$ columns of A. A procedure can be devised to make the i-th column of W orthogonal to the columns $1, \ldots, i-1$ of A, via linear combinations with first $i-1$ columns of W. Alternatively, columns $i+1, \ldots, n$ of W can be made 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. The resulting incomplete biorthogonalization process, which is sketched next, produces an approximate factored inverse.

Algorithm 4 (Factored Approximate INVerse (right–looking AINV))

0. *Input:* $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. *Output:* Z, D, W *such that* $A^{-1} \approx Z D^{-1} W^\top$.
1. *Let* $p = q = (0, \ldots, 0) \in \mathbb{R}^n$, $Z = [z_1, \ldots, z_n] = I_n$, $W = [w_1, \ldots, w_n] = I_n$.
2. ***for*** $k = 1, \ldots, n$
3a. $p_k = w_k^\top A e_k, \quad q_k = e_k^\top A z_k$
4. ***for*** $i = k+1, \ldots, n$
5a. $p_i = \left(w_i^\top A e_k\right)/p_k, \quad q_i = \left(e_k^\top A z_i\right)/q_k$
6. *Apply a dropping rule to* p_i, q_i
7. $w_i = w_i - w_k p_i, \quad z_i = z_i - z_k q_i$
8. *Apply a dropping rule to* $w_{j,i}$ *and* $z_{j,i}$, *for* $j = 1, \ldots, i$.
9. ***end***
10. ***end***
11. *Choose diagonal entries of* D *as the components of* p *or* q.

Line (3) and (5) are labeled with an (a) because they represent only one of two available options. An alternative way of computing W and Z is based on the fact that $W^\top A Z$ should become approximately diagonal. Instead of orthogonalizing W (resp. Z) with respect to the columns of A, we can apply a biconjugation process that enforces the bi-orthogonality of the columns of W and Z. For this we must enforce $e_k^\top W^\top A Z e_j = 0$ for all $k \neq j$, $1 \leqslant k, j \leqslant i$. This will result in simple changes to Algorithm 11. Specifically, the second option which we label with a (b) consists in changing lines (3a) and (5a) into the the following lines:

3b. $p_k = w_k^\top A z_k, \quad q_k = w_k^\top A z_k$
5b. $p_i = \left(w_i^\top A z_k\right)/p_k, \quad q_i = \left(w_k^\top A z_i\right)/q_k$

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. Remarkably, the computations of Z and W can be performed independently of each other for option (a).

It is important to note that in the original version of AINV [3, 5], no dropping is applied to p_i or q_i. One is only applied to w_i and z_i by discarding entries in W and Z that are less than a certain drop tolerance. Moreover, it has been pointed out in [3], that dropping entries of p and q produces poor results. The problem with dropping elements in p, q is that small entries $|p_j/p_i|$ may multiply large entries of $Z_{:,i}$ resulting in discarded entries in the approximate inverse that might not be small at all.

We still consider this variant because it shows very strong direct connections with various implementations of ILU. More general results, that concern practical variants, will be shown in Section 3.4. In [3, 5] p and q were defined using option (a), while option (b) was used in [17, 2] for symmetric positive definite matrices.

Note that the strict biorthogonality property of the exact factors no longer holds if dropping is introduced. Interestingly, however, stability can still be proved for H–matrices, in the case of incomplete LU factorizations as well as for AINV, see for example [3].

3.2 ILU with progressive factor inversion

In order to establish a bridge between the AINV and the ILU approaches, we introduce an intermediate algorithm that can be viewed as an ILU process with a simultaneous inversion of the factors which it produces. Specifically, if at step $k-1$ we have a matrix U of the form,

$$U = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ \mathrm{O} & 1 & \mathrm{O} \\ \mathrm{O} & \mathrm{O} & I \end{bmatrix}$$

the k–th step will compute the entries in position (2,3) of the above matrix and add them to the current U to get U_{new}. Consider the row vector $q^\top = e_k^\top U - e_k^\top$. Note that the 'diagonal' element q_k of q is zero. Then,

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

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

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

Hence the relation,

$$U_{new}^{-1} = U^{-1}(I + e_k q^\top)^{-1} = U^{-1}(I - e_k q^\top) = U^{-1} - U^{-1} e_k q^\top \ . \tag{16}$$

If we were to compute the inverse of U progressively, the columns $z_j, j = 1, \ldots, n$ of this corresponding progressive approximate inverse, could therefore be updated by the following formula at the k-th step:

$$z_i := z_i - z_k q_i, \qquad i = k+1, \ldots, n.$$

Analogous arguments hold for the L factor. This provides a formula for progressively computing $L^{-\top}$ and U^{-1} throughout the ILU factorization algorithm. We call the inverse factors W and Z as in Algorithm 4.

Algorithm 5 (ILU with progressive inversion of L and U)

Input: $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$. *Output: ILU factorization* $A \approx LDU$.

0. $p = q = 0 \in \mathbb{R}^n$, $L = U = I$, $W = [w_1, \ldots, w_n] = Z = [z_1, \ldots, z_n] = I$, $S = A$.
1. ***for*** $k = 1, \ldots, n$
2. $d_{kk} = s_{kk}$
3. ***for*** $i = k+1, \ldots, n$ *and when* $s_{ik} \neq 0$ *or* $s_{ki} \neq 0$
4. $p_i = s_{ik}/d_{kk}, \quad q_i = s_{ki}/d_{kk}$
5. *Apply a dropping rule to* p_i *and* q_i
6. $l_{ik} = p_i, \quad u_{ki} = q_i$
7. $w_i = w_i - w_k p_i, \quad z_i = z_i - z_k q_i$
8. *for all* $l \leqslant i$: *apply a dropping rule to* w_{li} *and to* z_{li}
9. ***for*** $j = k+1, \ldots, n$ *and when* $s_{ik} \neq 0$ *and* $s_{kj} \neq 0$
10. ***if*** $\mathrm{version}(p_i, q_j)$ *then:* $s_{ij} = s_{ij} - \frac{s_{ik} s_{kj}}{d_{kk}}$
11. ***end***
12. ***end***
13. ***end***

3.3 A first comparison theorem

The notation we used in Algorithm 5 already suggests that Z, W coincide with those of Algorithm 4. This is confirmed by the next theorem which establishes a few relations between various versions of AINV and ILU.

Theorem 6 *Assume that in Algorithm 4 and Algorithm 5 the same dropping rules are applied to p and q and that no dropping rule is applied to W, Z. Then certain choices of S in Algorithm 5, and the choice of options (a) or (b) in Algorithm 4 will imply the following identities.*

Choice of S / *(a), (b)*	*Alg. 5* *(13)*	*Alg. 4* *(a)*	*Alg. 5* *(15)*	*Alg. 4* *(b)*	*Alg. 5* *(14)*	*Alg. 4* *(a)*
	$\Downarrow$		$\Downarrow$		$\Downarrow$	
Identities	$L^{-\top} = W_{Alg.5/4}$		$L^{-\top} = W_{Alg.5/4}$			
			$U^{-1} = Z_{Alg.5/4}$		$U^{-1} = Z_{Alg.5/4}$	
	$\mathrm{diag}\,(D_{Alg.5}) = p_{Alg.4}$		$\mathrm{diag}\,(D_{Alg.5}) = \begin{cases} p_{Alg.4} \\ q_{Alg.4} \end{cases}$		$\mathrm{diag}\,(D_{Alg.5}) = q_{Alg.4}$	

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 k, that W is identical in both methods after any step k, that the first k diagonal entries of D coincide with $p_1, \ldots, p_k$ and that

$$s_{ij} = w_i^\top A e_j, \text{ for all } i, j > k. \tag{17}$$

Initially, for $k = 0$ 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 k of each algorithm. Suppose that the first $k-1$ diagonal entries of D coincide with the first $k-1$ components of p and that

$$s_{ij} = w_i^\top A e_j, \text{ for all } i, j \geqslant k.$$

We immediately obtain $p_k = w_k^\top A e_k = s_{kk} = d_{kk}$ and

$$p_i = w_i^\top A e_k / d_{kk} = s_{ik}/d_{kk}, \text{ for all } i = k+1, \ldots, n.$$

From this it follows that $p_i^{(new)}$ from both algorithms is identical and satisfies $p_i^{(new)} = l_{ik}$, for any $i > k$. Since we choose the same dropping rule for both algorithms, this equality still holds after sparsifying these entries.

Obviously the update procedure

$$w_i^{(new)} = w_i^{(old)} - w_k^{(old)} p_i^{(new)}, \text{ for all } i > k$$

from Algorithm 4 is the nontrivial update part on

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

in Algorithm 5. Now for the new entries s_{ij}, $i, j > k$ we have the update procedure

$$\begin{aligned}
s_{ij}^{(new)} &:= s_{ij}^{(old)} - p_i^{(new)} s_{kj}^{(old)} \\
&= e_i^\top \begin{bmatrix} I_{k-1} & & & & \\ & 1 & & & \\ & -p_{k+1}^{(new)} & 1 & & \\ & \vdots & & \ddots & \\ & -p_n^{(new)} & & & 1 \end{bmatrix} S^{(old)} e_j \\
&= e_i^\top \begin{bmatrix} I_{k-1} & & & & \\ & 1 & & & \\ & -p_{k+1}^{(new)} & 1 & & \\ & \vdots & & \ddots & \\ & -p_n^{(new)} & & & 1 \end{bmatrix} (W^{(old)})^\top A e_j = e_i^\top (W^{(new)})^\top A e_j.
\end{aligned}$$

This completes the proof. □

3.4 Dropping elements in W and Z

As mentioned earlier, Algorithm 4 is more general than the original AINV algorithm [3, 5] which does not allow dropping entries in the update factors from p, q but only in the updated matrices Z, W. The previous theorem does not address this case since its assumptions do not allow dropping in W and Z. The key to getting a connection between AINV and ILU-type factorizations lies is equation (17). If a Schur–complement is constructed so that this relation holds between AINV and an ILU factorization, it is easy to see that both algorithms will result in comparable Ws and Zs. The results to be proved next concerns such update versions, i.e., they are valid for the Schur complements defined by either of the following three expressions,

$$S = (W^\top A)_\square \ , \quad \text{or} \quad S = (AZ)_\square \ , \quad \text{or} \quad S = (W^\top AZ)_\square \ , \tag{18}$$

where the square subscripts indicate that an appropriate submatrix is extracted. In these situations, we may expect for example $W^\top$ to be close to L^{-1} in some sense, i.e., that $W^\top$ can be viewed as an approximation to the inverse of L.

We will need two simple lemmas before establishing the general result. We begin with some required additional notation. The matrix W at the k–th step of the Algorithm 5 is denoted by $W^{(k)}$, starting with $W^{(0)} = I$. It is obtained from $W^{(k-1)}$ by the relation,

$$W^{(k)} = W^{(k-1)} \left[I - e_k (p^{(k)})^\top \right] - G_k \tag{19}$$

where G_k is the matrix of elements that have been dropped in the process and $p^{(k)}$ is the vector denoted by p in the algorithm, as step k. The vector $p^{(k)}$ has zero elements in positions 1 through k, i.e., $e_j^\top p^{(k)} = 0$ for all $j \leqslant k$.

Lemma 7 *Denote by Q_k the matrix*

$$Q_k = I - e_k (p^{(k)})^\top \tag{20}$$

and let G_k the matrix of elements dropped in the matrix $W^{(k)}$ at step k. Then,

$$G_k \ Q_{k-l} = G_k \quad 0 \leq l \leq k-1 \ . \tag{21}$$

Proof. Note that $(G_k)_{ij} = 0$ for $j \leqslant k$ or $i > k$. Therefore, we can write

$$G_k = \sum_{i \leqslant k, \ j > k} g_{ij} e_i e_j^\top$$

and so

$$G_k Q_{k-l} = \sum_{i \leqslant k, \ j > k} g_{ij} e_i e_j^\top \left(I - e_{k-l} (p^{(k-l)})^\top \right) = \sum_{i \leqslant k, \ j > k} g_{ij} e_i e_j^\top = G_k$$

□

This relation is key to establishing the next lemma.

Lemma 8 *Let $W^{(k)}, G_k$ defined by (19) and (20) and let*

$$L_k^{-\top} = Q_1 \times Q_2 \cdots Q_k$$

then

$$I - W^{(k)} L_k^\top = \sum_{i=1}^{k} G_i \tag{22}$$

Proof. Exploiting the result of Lemma 7 we can write

$$\begin{aligned} W^{(k)} &= W^{(k-1)} Q_k - G_k = (W^{(k-1)} - G_k) Q_k \\ &= [(W^{(k-2)} - G_{k-1}) Q_{k-1} - G_k] Q_k = [W^{(k-2)} - G_{k-1} - G_k] Q_{k-1} Q_k \\ &= [W^{(k-3)} - G_{k-2} - G_{k-1} - G_k] Q_{k-2} Q_{k-1} Q_k \\ &= \cdots \\ &= [W^{(0)} - G_1 - \cdots - G_k] Q_1 Q_2 \cdots Q_k \end{aligned}$$

This essentially gives the result by recalling that $W^{(0)} \equiv I$. □

We now need to link the AINV algorithm (Algorithm 4) with Algorithm 5. To interpret AINV as a form of ILU, the definition of the approximate Schur–complement must be adapted. Standard computations of the Schur–complement in Algorithm 2 correspond to the definition in (15), (12). We now consider a hypothetical version of Algorithm 5, in which the Schur–complement is defined via one of the options in (18).

An important observation is that we will obtain the same W matrices in algorithms 4 and 5, if the same dropping rule is used for p in both algorithms, and if the Schur complement is defined from (18) in Algorithm 5.

Lemma 8 indicates that $W^{(k)}$ is an approximate inverse of $L_k^\top$ if the sum of the matrices G_i remains small, a statement which can be made more precise if a drop tolerance strategy is invoked. Putting these observations together leads to the following result.

Theorem 9 *Assume that in Algorithm 5 w_{ij} is dropped if $|w_{ji}| \leqslant \varepsilon$, $i \leqslant k, j > k$. Then, the L–factor and the matrix W produced by Algorithm 5 are such that,*

$$|(I - W L^\top)_{ij}| \leqslant (j - i)\varepsilon, \quad 1 \leqslant i \leqslant j \leqslant n \tag{23}$$

If in addition the Schur–complement in Algorithm (5) is defined through (18) and if the related version of Algorithm 4 uses the same dropping rules for W as Algorithm 5, whereas no dropping is applied to p, q, then the matrices W produced by both algorithms are identical.

Proof. The first part of the theorem follows by applying the previous lemma with $k \equiv n$, and noting that in position (i, j) of W, dropping occurs at most $(j - i)$ times since at step k dropping takes place only in the rectangle of pairs (i, j) such that $i < k < j$.

The second part of the theorem was stated above without proof. A rigorous proof would be by induction. In short, both sequences satisfy the same recurrence relation:

$$W^{(k)} = W^{(k-1)}(I - e_k(p^{(k)})^\top) - G_k$$

because $p^{(k)}$ and G_k are the same in both algorithms due to the common dropping rules. This leads to the same sequence of W's for both algorithms. □

Though all the analysis has been made for the lower triangular factor L and the associated W, it is clear that analogous relationships can be established between U^{-1} and Z (apply Theorem 9 to $A^\top$). We mention that an algorithm of this type was recently presented in [4] for the symmetric positive definite case where the Cholesky factor was taken as a by–product from the AINV factor.

We now consider the more general situation when no dropping is applied to p and q in Algorithm 4 while Algorithm 5 does perform dropping. In this case the W matrices obtained by both algorithms are no longer (easily) comparable. This is because the vectors p, q in the recurrence (19) are no longer the same. We could modify Algorithm 5 so that dropping is also not done in p and q but only in L after p, q have been used to update W and Z. This amounts to simply moving Line 7 of the algorithm to behind line 4. Specifically, only Lines 5-7 change in the algorithm and they become:

5a. $w_i = w_i - w_k p_i, \quad z_i = z_i - z_k q_i$
6a. Apply a dropping rule to p_i and q_i
7a. $l_{ik} = p_i, \quad u_{ki} = q_i$

We will refer to this algorithm, as the a-version of Algorithm 5. If the goal is to mimic the behavior of the actual AINV (no dropping in p, q), then clearly this version is more suitable and practical.

There are now two sequences of L matrices produced by this version of the algorithm. One is the sequence L_k seen before which uses the vectors $p^{(k)}$ before dropping. The second is a sequence $\tilde{L}_k$ which corresponds to the actual L-factors produced by the factorization and which uses the vectors p, q after dropping is applied. Therefore, we define the elementary factors corresponding to this second sequence:

$$\tilde{Q}_k = I - e_k(\tilde{p}^{(k)})^\top = I - e_k(p^{(k)} - f_k)^\top \tag{24}$$

in which f_k is the column vector of elements that have been dropped in $p^{(k)}$, and

$$\tilde{L}_k^{-\top} = \tilde{Q}_1 \times \tilde{Q}_2 \cdots \tilde{Q}_k$$

which is the transpose of the inverse L-factor produced at the end of step k of algorithm 5. A standard result of LU factorizations is that $\tilde{L}_k$ is simply the matrix with column

vectors $\tilde{p}^{(i)}$, $i = 1, \ldots, k$, to which we add the identity. Similarly for L_k. Therefore, it is clear that:

$$L_k^\top - \tilde{L}_k^\top = \sum_{i=1}^{k} e_i f_i^\top \tag{25}$$

We define,

$$F_k = \sum_{i=1}^{k} e_i f_i^\top. \tag{26}$$

Putting Equation (25) into (22) gives the following generalization of Theorem 9.

Theorem 10 *Assume that the a-version of Algorithm 5 is used and let $W^{(k)}, G_k, F_k$ defined by (19) and (26), and $\tilde{L}_k$ the L–factor obtained at step k of the same algorithm. Then the following equality holds*

$$I - W^{(k)} \tilde{L}_k^\top = \sum_{l=1}^{k} G_l + W^{(k)} F_k \ . \tag{27}$$

Furthermore, assume that at step k of Algorithm 5 an entry l_{ik} is discarded at most if

$$|l_{ik}| \ \times \ \max_{j=k,\ldots,n} |w_{jk}| \leqslant \varepsilon$$

whereas no dropping is applied for p, q in Algorithm 4. In both algorithms it is assumed that w_{ij} is dropped if

$$|w_{ij}| \leqslant \varepsilon, \quad i \leqslant k, j > k \ . \tag{28}$$

Then for Algorithm 5 the following holds for any $j > i$:

$$\mid \ (I - W \tilde{L}^\top)_{ij} \ \mid \leqslant 2(j-i)\varepsilon. \tag{29}$$

If in addition the Schur–complement in Algorithm (5) is defined through (18), then the matrices W produced by Algorithm 5 and the related version of Algorithm 4 are identical.

Proof. Relation (27) follows immediately from (25) and (22). Denote $W^{(n)}$ by W and, similarly F_n by F. For the rest of the theorem, we write W as

$$W = \sum_{k=1}^{n} w_k e_k^\top$$

from which we infer that

$$WF = \sum_{k=1}^{n} w_k e_k^\top \sum_{k=1}^{n} e_k f_k^\top = \sum_{l=1}^{n} w_l f_l^\top$$

We now consider the entry (i,j) on both sides of (27)

$$|e_i^\top (I - W\tilde{L}^\top)e_j| \leqslant \left|\sum_{l=1}^{n} e_i^\top G_l e_j\right| + \left|e_i^\top W F e_j\right| \ . \tag{30}$$

From Theorem 9, we already have a bound for the first term on the right-hand-side

$$\left|\sum_{l=1}^{n} e_i^\top G_l e_j\right| \leqslant (j-i)\varepsilon \quad 1 \leqslant i \leqslant j \leqslant n \tag{31}$$

For the second term, we write

$$|e_i^\top W F e_j| = \left|\sum_{k=1}^{n} e_i^\top w_k f_k^\top e_j\right| \leqslant \sum_{k=1}^{n} |e_i^\top w_k| \ |f_k^\top e_j|$$

Notice that $e_i^\top w_k = 0$ for $k < i$ and similarly $f_k^\top e_j = 0$ for $k \geqslant j$ so the above inequality becomes

$$|e_i^\top W F e_j| \leqslant \sum_{k<j,\ k\geqslant i} |e_i^\top w_k| \ |f_k^\top e_j| \leqslant \sum_{k<j,\ k\geqslant i} \max_i |w_{ki}| \ |f_k^\top e_j|$$

According to the dropping strategy each term in the sum does not exceed ε. Therefore,

$$|e_i^\top W F e_j| \leqslant \sum_{k<j,\ k\geqslant i} \varepsilon = (j-i)\varepsilon \tag{32}$$

Substituting (31) and (32) into (30) yields the desired result (29). □

As in Theorem 9 all the analysis can be carried over to establish analogous relationships between U^{-1} and Z.

3.5 Left-looking AINV

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

Algorithm 11 (Factored Approximate INVerse (left–looking AINV))

Input: $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$. *Output:* Z, D, W *such that* $A^{-1} \approx ZD^{-1}W^\top$.

0. $p = q = 0 \in \mathbb{R}^n$, $p_1 = q_1 = a_{11}$; $W = Z = D = I_n$

1. ***for*** $i = 2, \ldots, n$

2. ***for*** $j = 1, \ldots, i-1$

3a. $P_j = \left(w_i^\top A e_j\right)/p_j \qquad Q_j = \left(e_j^\top A z_i\right)/q_j$

4. *apply a dropping rule to* P_j *and* Q_j.

5. $w_i = w_i - w_j \ P_j, \qquad z_i = z_i - z_j \ Q_j$

6. *for all $l \leqslant i$: apply a dropping rule to w_{li}, z_{li}.*
7. ***end***
8a. $p_i = w_i^\top A e_j, \quad q_i = e_j^\top A z_i$
9. ***end***
10. *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 in sequence, column by column, while in Algorithm 4 the updates are performed simultaneously for all columns. This difference corresponds to the difference between certain versions of the Gram-Schmidt orthogonalization algorithm. Similarly to Algorithm 4, Algorithm 11 also has a (b) option which consists of the following changes to lines (3a) and (8a):

3b. $P_j = \left(w_i^\top A z_j\right)/p_j \qquad Q_j = \left(w_j^\top A z_i\right)/q_j$
8b. $p_i = w_i^\top A z_j, \quad q_i = w_j^\top A z_i$

The simple relation between Algorithms 4 and 11 is stated in the following proposition which is straightforward to verify.

Proposition 12 *Assume that the same dropping rule is applied to p, q and P, Q and that the same dropping rule is also applied to W and Z in Algorithm 4 and Algorithm 11. Then both algorithms will compute the same W, Z. They also compute the same D if the same choice is made for the D entries in their lines 11 and 10 respectively.*

In fact the equality between both algorithms also includes the case when each column is sparsified only once. For Algorithm 11 this would be a more natural dropping rule, i.e., entries of z_{li}, w_{li} would be discarded only if $j = i - 1$. For step k of Algorithm 4 the associated dropping rule would sparsify only column $k+1$ of W and Z which might lead to extreme fill–in for W and Z.

3.6 Bordering methods

An analysis similar to the one developed in the previous sections was discussed in the earlier report [8] which established links between ILU and approximate inverse methods based on "bordering". An approximate inverse method of this type was discussed in [23]. The main idea is to partition the (k, k) principal submatrix of A as

$$A_k = \begin{pmatrix} B & f \\ e^\top & c \end{pmatrix}$$

where $B \equiv A_{k-1}$ is of dimension $k-1$. Assume that we already know the factorized approximate inverse of B, in the form $W^\top B Z = D$ where W, Z are unit upper triangular

and D is diagonal. Then the factored inverse of A_k can be obtained by writing

$$W_{new}^\top A_k Z_{new} \equiv \begin{pmatrix} W & g \\ 0 & 1 \end{pmatrix}^\top \begin{pmatrix} B & f \\ e^\top & c \end{pmatrix} \begin{pmatrix} Z & h \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} D & 0 \\ 0 & s \end{pmatrix}$$

The above relation immediately shows that h, s, g must satisfy the equations

$$B^\top g = -e \tag{33}$$

$$Bh = -f \tag{34}$$

$$s = c + g^\top f + e^\top h + g^\top Bh. \tag{35}$$

To develop approximate inverse methods, we can simply use vectors g, h provided from approximately solving systems (33) and (34) and then computing s. In fact, we could simply utilize the relation $W^\top BZ \approx D$ to approximate h, g and discard some entries according to a dropping rule. This means that we compute g and h from

$$g := -WD^{-1}Z^\top e \; ; \qquad h := -ZD^{-1}W^\top f$$

and apply a dropping rule to g and h. The algorithm now becomes clear. Start with the (1,1) matrix which has a trivial factorized approximate inverse and then build, recursively, the approximate inverses of the (k, k) principal submatrix of A_k from that of the $(k-1, k-1)$ principal submatrix, for $k = 2, \ldots, n$. It is also possible to develop additional variants to the algorithm depending on how the diagonal elements of D are selected. Denote the columns of the final W and Z matrices by w_j and z_j, $j = 1, \cdots, n$. Then, formula (35) corresponds to the choice $s = w_k^\top A_k z_k$. Two other choices are obtained by taking $s = c + g^\top f \equiv w_k A_k e_k$ which corresponds to (13) or $s = c + e^\top h \equiv e_k^\top A_k z_k$ which is analogous with (14).

In [8] a result similar to Theorem 6 was mentioned, though this applies again to non–practical versions. However, it was also shown that there is a practical relation between bordered factored inverse methods and Algorithm 11. Specifically, a first result is that both algorithms compute the same W and D when (1) the choice $s = w_k^\top A_k e_k$ is used in the bordered approximate inverse algorithm; (2) the same dropping rule is used for W in both algorithms; and (3) no dropping is applied to Z in the bordering method. A second result is that both algorithms compute the same D and Z under analogous conditions.

4 Consequences

The comparison results established in the previous sections can provide theoretical insight into known algorithms by exploiting the body of existing literature on ILU and AINV. On the practical side, they can also help develop improved variants of both ILU and AINV. In fact new algorithms have already been developed by exploiting these relationships in both directions. In the following we briefly discuss a few of these known results and point to other potential applications yet to be explored.

4.1 AINV with pivoting

In [7] we applied what is known from ILU algorithms to devise pivoting techniques for approximate inverse methods. This technique can be easily inferred from the following relation which holds at step k,

$$W^\top AZ \approx \begin{pmatrix} D & 0 \\ 0 & S \end{pmatrix}$$

where $W^\top$ and Z are the inverses of the matrices L and U respectively of the LU factorization. As was seen in earlier sections, these are also close to the W and Z matrices obtained at step k of the AINV procedure. If we apply permutations $\Pi^\top$, Σ to S, on the left and right respectively, it is easy to determine how this permutation must be also applied to W and Z for consistency:

$$\begin{pmatrix} I & 0 \\ 0 & \Pi^\top \end{pmatrix} W^\top AZ \begin{pmatrix} I & 0 \\ 0 & \Sigma \end{pmatrix} \approx \begin{pmatrix} D & 0 \\ 0 & \Pi^\top S \Sigma \end{pmatrix}$$

This means that the corresponding rows of W and Σ need to be permuting according to the permutation applied to S. As for which permutation to apply, we can use the parallel with ILU, since S is more or less the same matrix that is obtained from the ILU factorization. For example, we can simply do a column permutation as is done in ILUTP [22]. The strategy suggested in [7] is to use row and column pivoting successively a few times (in the same step) until the pivot satisfies a certain stability condition both for the k-th row and the k-th column of S. For details see [7]. Numerical experiments, do confirm that this procedure is much more robust than a non-pivoting AINV.

4.2 An ILU based on monitoring the growth factors

Proceeding in the reverse direction, the relationships established in this paper have also allowed to design more robust ILU techniques. Here, we cite two independent works [6, 4]. The paper [6] introduces dropping strategies in ILU that are more rigorous than simple-minded threshold techniques, by exploiting the parallel between ILU and AINV [6].

The fundamental relation which was exploited in [6] is (28). As shown by Theorem 10 this relation insures that the W matrix is close to the inverse of the factor L. Therefore the L–factor will clearly be stable, in the sense that its inverse will have a moderate norm. Similarly for the U factor.

In [4], an incomplete Cholesky factorization was extracted as a by–product of the AINV process for the symmetric positive definite case [2]. This can be seen as another way of exploiting the relationships between ILU and AINV. Numerical observation have shown that AINV preconditioning often outperforms the standard incomplete Cholesky factorization for the Conjugate Gradient. In [4], it was shown that only the by–product

incomplete Cholesky decomposition was able to obtain results comparable with those of AINV. However, the crucial dropping strategy (28) is not employed. We believe that such a dropping strategy may substantially enhance the quality of the factor produced by the method.

4.3 Theory: Results for SPD matrices and for H–matrices

From a theoretical point of view, some results on approximate inverse methods can be derived by exploiting the relationship with ILU – for which much is known. This line of argument was indeed exploited, for example, in [2] by transferring the related incomplete Cholesky decomposition [24].
An immediate corollary for the symmetric positive definite case is the following.

Corollary 13 *Let A be symmetric positive definite. Suppose that Algorithm 4 and Algorithm 5 apply the same dropping rule to p and q and that no dropping is applied to W and Z.*
If Option (b) is used in Algorithm 4 and if S in Algorithm 5 is defined via (15), then both Algorithms do not break down. In addition both methods compute the same W and 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 Property 1. □

It is well–known that the ILU decomposition of an H–matrix exists for any of the dropping strategies discussed in Section 2.3, see, e.g., [20, 19]. It immediately follows that W and Z of Algorithm 4 exist for this case. Likewise for M–matrices we know that the computed L and U are again M–matrices. Consequently W and Z have to be nonnegative in this case. However this argument only applies for the theoretic way of dropping in p and q. A proof for the natural way of dropping is given in [3].

4.4 Further applications

The few applications just described indicate that much can be gained by exploiting good qualities of a technique form one class to improve the corresponding algorithm from the other class. Another possible application which does not seem to have been explored is to exploit level-of-fill strategies used in ILU techniques, for developing pattern-based dropping strategies for AINV methods. Finding good patterns for dropping in AINV methods remains poorly understood. For matrices with good diagonal dominance properties, level-of-fill techniques work quite well, and, when combined with blocking they are often the preferred techniques for solving certain types of problems in fluid dynamics, for example.

In ILU(p) a level-of-fill *lev* is attributed to each element during factorization. Each element that is updated by formula such as (9) will have its `lev` value updated by the formula

$$\texttt{lev}(s_{ij}) = \min\{\texttt{lev}(s_{ij}) \ , \ \texttt{lev}(s_{ik}) + \texttt{lev}(s_{kj}) + 1\}$$

Initially, any nonzero element is assigned a `lev` value of 0, and any zero element is (implicitly) assigned an infinite `lev` value. It is typical to process the ILU factorization in two phases, a symbolic one and a numeric one. The pattern of ILU(p) is determined in the symbolic factorization. This pattern can now be used for obtaining a pattern for AINV. Consider, in Line 5 of Algorithm 4, the update to w_j the j-th column of W. This update is $w_j = w_j - w_k p_j$, or, component-wise $w_{ij} = w_{ij} - w_{ik} p_j$. Now recall that p_j is nothing but s_{jk}, so

$$w_{ij} = w_{ij} - w_{ik} s_{jk}$$

Using the same model for decrease of the elements in the factorization, we can easily see that a good way to define the level of fill of w_{ij} is

$$\texttt{lev}(w_{ij}) = \min\{\texttt{lev}(w_{ij}) \ , \ \texttt{lev}(w_{ik}) + \texttt{lev}(s_{jk}) + 1\}$$

Notice that computing the `lev` values for the L factors is inexpensive.

A hint at another potential class of applications is provided by the recent paper [6]. There, some information about W, Z is exploited to gain insight on suitable dropping strategies when building L and U. ILU and AINV can be viewed as some kind of optimization methods, which produce factors that approximate either A directly (for ILU) or its inverse (for AINV). A rule of thumb seems to be that ILU works better than AINV methods when it produces factors that are stable. In other words, *accuracy+stability* $\rightarrow$ *fast convergence*. If we find factors L from ILU, such that $L^{-1} \approx W$ and W is well-behaved then clearly both criteria of accuracy and stability are satisfied. This suggests that strategies which combine both criteria should be developed. In [6] a dropping strategy was found which ensured that $L^{-1} \approx W$ – using the result of Theorem 10. But other strategies may exist.

5 Conclusions

We have shown a number of inter-relations between factored approximate inverse and related incomplete factorizations of *ILU* type. We also established relations between different approaches to compute factored approximate inverses. It was shown that approximate techniques are intimately related to ILU factorizations. Indeed, 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). It is also clear, however, that there is a multitude of variation in between these two 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] O. Axelsson. *Iterative Solution Methods.* Cambridge University Press, New York, 1994.

[2] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22:1318–1332, 2000.

[3] 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.

[4] M. Benzi and M. Tůma. A robust incomplete factorization preconditioner for positive definite matrices. Presented at the Tahoe conference on preconditioning methods, April. 2001. Paper in preparation, 2001.

[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] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra Appl.* to appear.

[7] M. Bollhöfer and Y. Saad. A factored approximate inverse preconditioner with pivoting. *SIAM J. Matrix Anal. Appl.* to appear.

[8] M. Bollhöfer and Y. Saad. *ILU*s and factorized approximate inverses are strongly related. Part I: Overview of results. Technical Report umsi–2000-39, Minnesota Supercomputer Institue, University of Minnesota, 2000.

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

[10] E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang. Towards a cost-effective high order ilu preconditioner. *BIT*, 31:442–463, 1992.

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

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

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

[14] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, 1997.

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

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

[17] 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.

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

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

[20] 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.

[21] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM Trans. Math. Software*, 6:206–219, 1980.

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

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

[24] M. Tismenetsky. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra Appl.*, 154–156:331–353, 1991.

[25] Z. Zlatev. Use of iterative refinement in the solution of sparse linear systems. *SIAM Journal on Numerical Analysis*, 19:381–399, 1982.