The University of Manchester

A max-plus approach to incomplete Cholesky factorization preconditioners

Jonathan Hogg, James Hook, Jennifer Scott and Francoise Tisseur

December 2016

MIMS EPrint: **2016.59**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: `http://www.manchester.ac.uk/mims/eprints`

And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

A MAX-PLUS APPROACH TO INCOMPLETE CHOLESKY FACTORIZATION PRECONDITIONERS

JONATHAN HOGG*, JAMES HOOK†, JENNIFER SCOTT*, AND FRANÇOISE TISSEUR‡

Abstract. We present a new method for constructing incomplete Cholesky factorization preconditioners for use in solving large sparse symmetric positive-definite linear systems. This method uses max-plus algebra to predict the positions of the largest entries in the Cholesky factor and then uses these positions as the sparsity pattern for the preconditioner. Our method builds on the max-plus incomplete LU factorization preconditioner recently proposed in [J. Hook and F. Tisseur, *Incomplete LU preconditioner based on max-plus approximation of LU factorization*, MIMS Eprint 2016.47, Manchester, 2016] but applied to symmetric positive-definite matrices, which comprise an important special case for the method and its application. A attractive feature of our approach is that the sparsity pattern of each column of the preconditioner can be computed in parallel. Numerical comparisons are made with other incomplete Cholesky factorization preconditioners using problems from a range of practical applications. We demonstrate that the new preconditioner can outperform traditional level-based preconditioners and offer a parallel alternative to a serial limited-memory based approach.

Key words. Sparse symmetric linear systems, incomplete factorizations, preconditioners, Hungarian scaling, max-plus algebra, sparsity pattern.

AMS subject classifications. 65F08, 65F30, 65F50, 15A80.

1. Introduction.

Incomplete Cholesky (IC) factorizations are an important tool in the solution of large sparse symmetric positive-definite linear systems of equations $Ax = b$. Preconditioners based on an incomplete factorization of A (that is, a factorization in which some of the fill entries that would occur in a complete factorization and possibly some of the entries of A are ignored) have been in widespread use for more than 50 years (see, for example, [33] for a brief historical overview that highlights some of the most significant developments). For general nonsymmetric systems, incomplete LU (ILU) factorization preconditioners are frequently used as they work well for a wide range of problems and, again, many variants have been proposed (see [30] for an introduction). The basic idea is to compute a factorization $A \approx LU$ (or $A \approx LL^T$ in the positive-definite case) with L and U sparse triangular matrices with the fill-in (that is, the entries in L and U that lie outside the sparsity pattern of A) restricted to some sparsity pattern S. Recently, Hook and Tisseur [19] have shown how max-plus algebra can be used to approximate the order of magnitude of the moduli of the entries in the LU factors of A and have used this to construct the sparsity pattern of ILU preconditioners. Max-plus algebra is the analog of linear algebra developed for the binary operations max and plus over the real numbers together with $-\infty$, the latter playing the role of additive identity; an introduction and a large number of references to the literature may be found in [10, Chap. 35]. In the past few years, max-plus algebra has been used to examine a number of numerical linear algebra problems [2, 13, 16, 27]. While the numerical experiments reported on in [19] were limited to modest-sized sparse problems (of order up to 10^3), they did indicate the potential of the approach to compute ILU preconditioners that

*STFC Rutherford Appleton Laboratory, Harwell Campus, Oxfordshire, OX11 0QX, UK. Supported by EPSRC grant EP/M025179/1.

†Institute for Mathematical Innovation, University of Bath, Claverton Down, Bath BA2 7AY, UK.

‡School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK. Supported by EPSRC grant EP/I005293 and by a Royal Society-Wolfson Research Merit Award.

can outperform traditional level of fill methods and be competitive with a threshold-based method.

One drawback of the max-plus method is that the sparsity pattern S cannot be updated to account for any pivoting during the factorization of A, so that the pattern chosen by the max-plus analysis is only useful when the factorization does not require row or column interchanges. An attractive feature of symmetric positive-definite matrices is that they will always (in exact precision) admit a Cholesky factorization without pivoting. However, if A is close to being indefinite or if an incomplete factorization is computed, then breakdown can occur (that is, a zero or negative pivot is encountered). In this case, we follow Manteuffel [26] and add a small multiple of the identity, that is, we factorize $DAD + \alpha I$ for some shift $\alpha > 0$ and diagonal scaling D. This avoids the need for pivoting and preserves the chosen sparsity structure S of the factors and, as shown recently by Scott and Tůma [34, 35], the resulting IC factors generally still provide an effective preconditioner for A. We observe that prescaling of A is essential to limit the size of the shift; preordering is also normally needed to limit fill in the factors (and hence the number of entries that are dropped during the incomplete factorization).

The aim of this paper is to present an algorithm for constructing IC preconditioners for large sparse positive-definite problems using max-plus algebra to predict the positions of the largest entries in the Cholesky factor. The sparsity pattern S_j of each column j can be determined in parallel. Once S_j is found, the IC factorization can be computed using a conventional serial procedure or using the novel approach of Chow and Patel [3] who propose using an iterative method to compute the entries of the factors. All the nonzero entries in the incomplete factors can be computed in parallel and asynchronously, using a number of sweeps of an iterative method. A key issue with their approach is that the pattern S must be chosen a priori.

The remainder of this paper is organized as follows. In Section 2, we briefly recall max-plus incomplete LU factorizations and then, in Section 3, we focus on positive-definite matrices. We present algorithms for computing each column of the max-plus factor independently. Results for a wide range of problems are presented in Section 4 and comparisons are given between our new max-plus IC preconditioner and both level-based and memory-based IC preconditioners used with the preconditioned conjugate gradient method. In Section 5, we present alternative approaches to the max-plus computation. Finally, some concluding remarks and possible future directions are given in Section 6. Note that we do not assume that the reader is familiar with the use of max-plus algebra but define the concepts that we need as necessary.

Throughout this paper, matrices are denoted by capital letters with their entries given by the corresponding lower case letter in the usual way, that is, $A = (a_{ij}) \in \mathbb{R}^{n\times n}$. Max-plus matrices are denoted by calligraphic capital letters and their entries by the corresponding lower case calligraphic letter, that is, $\mathcal{A} = (\mathpzc{a}_{ij}) \in \mathbb{R}_{\max}^{n\times n}$, where $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$.

2. Max-plus approximation of LU factorization. In [19], the authors use max-plus algebra to introduce the following method for approximating the moduli of the entries of the L and U factors of sparse matrices.

Consider the map $\mathcal{V}$ defined as

$$\begin{aligned} \mathcal{V}: \quad & \mathbb{R} \to \mathbb{R}_{\max} \\ & x \mapsto \log|x|, \end{aligned} \tag{2.1}$$

with the convention that $\log 0 = -\infty$. For $x, y \in \mathbb{R}$, $\mathcal{V}(xy) = \mathcal{V}(x) + \mathcal{V}(y)$, and when $|x| \gg |y|$ or $|x| \ll |y|$, $\mathcal{V}(x+y) \approx \max\{\mathcal{V}(x), \mathcal{V}(y)\}$, which suggests using the operations max and plus in place of the classical addition and multiplication once we have applied the map $\mathcal{V}$. The set $\mathbb{R}_{\max}$ endowed with the addition $x \oplus y = \max\{x, y\}$ and the multiplication $x \otimes y = x + y$ is called the *max-plus semiring*. It is not a ring as there is no additive inverse and hence there is no max-plus subtraction. The identity elements are $-\infty$ for the addition and 0 for the multiplication. When applied componentwise to a matrix, the map (2.1) allows us to transform the matrix $A \in \mathbb{R}^{n\times n}$ into a *max-plus matrix*, i.e., $\mathcal{V}(A) = \mathcal{A}$ is a matrix with entries $a_{ij} = \log|a_{ij}|$ in $\mathbb{R}_{\max}$. The max-plus matrix $\mathcal{V}(A)$ is termed the *valuation* of A.

It is well-known that the entries in the lower triangle of L and the upper triangle of U of a LU factorization of $A \in \mathbb{R}^{n\times n}$ can be expressed explicitly in terms of determinants of submatrices A (see [12, p. 35]). Using this fact and the heuristic that $\mathcal{V}\big(\det(A)\big) \approx \operatorname{perm}\big(\mathcal{V}(A)\big)$, where

$$\operatorname{perm}(\mathcal{A}) = \max_{\pi\in\Pi(n)} \sum_{i=1}^{n} a_{i,\pi(i)} \in \mathbb{R}_{\max}, \tag{2.2}$$

is the max-plus permanent of $\mathcal{A} \in \mathbb{R}_{\max}^{n\times n}$, $\Pi(n)$ being the set of all permutations on $\{1, \dots, n\}$, Hook and Tisseur [19] define the max-plus LU factors of $\mathcal{A} \in \mathbb{R}_{\max}^{n\times n}$ as the lower triangular max-plus matrix $\mathcal{L}$ and upper triangular max-plus matrix $\mathcal{U}$ with entries

$$\ell_{ik} = \operatorname{perm}\big(\mathcal{A}([1\colon k-1, i], 1\colon k)\big) - \operatorname{perm}\big(\mathcal{A}(1\colon k, 1\colon k)\big), \quad i \geq k, \tag{2.3}$$

$$u_{kj} = \operatorname{perm}\big(\mathcal{A}(1\colon k, [1\colon k-1, j]) - \operatorname{perm}\big(\mathcal{A}(1\colon k-1, 1\colon k-1)\big), \quad j \geq k, \tag{2.4}$$

and $\ell_{ik} = u_{kj} = -\infty$ if $i, j < k$. If the two terms on the right hand side of (2.3) or (2.4) are $-\infty$ then the convention $-\infty - (-\infty) = -\infty$ is used. If the second term is $-\infty$ but the first is not, then $\mathcal{A}$ does not admit max-plus LU factors. Hook and Tisseur show that $\mathcal{L}$ and $\mathcal{U}$ are such that

$$\mathcal{V}(L) \approx \mathcal{L}, \qquad \mathcal{V}(U) \approx \mathcal{U}, \tag{2.5}$$

where the symbol "$\approx$" should be interpreted componentwise as "offers an order of magnitude approximation".

EXAMPLE 2.1. *Consider*

$$A = \begin{bmatrix} -10 & 10 & -10^3 \\ 0 & 1 & -1 \\ 10^3 & 1 & 10^{-2} \end{bmatrix}, \quad \mathcal{V}(A) = \begin{bmatrix} 1 & 1 & 3 \\ -\infty & 0 & 0 \\ 3 & 0 & -2 \end{bmatrix}.$$

We compute the max-plus LU factors of $\mathcal{V}(A)$ using (2.3) *and* (2.4)*. For such small examples we can evaluate the permanent of a matrix or submatrix by simply evaluating all possible permutations and recording the maximum. E.g.*

$$\begin{aligned} u_{33} &= \operatorname{perm}\big(\mathcal{A}([1\colon 3], [1\colon 3])\big) - \operatorname{perm}\big(\mathcal{A}(1\colon 2, 1\colon 2)\big) \\ &= \max\{-1, 1, -\infty, 4, -\infty, 6\} - \max\{1, -\infty\} = 5. \end{aligned}$$

We obtain

$$\mathcal{L} = \begin{bmatrix} 0 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ 2 & 3 & 0 \end{bmatrix}, \quad \mathcal{U} = \begin{bmatrix} 1 & 1 & 3 \\ -\infty & 0 & 0 \\ -\infty & -\infty & 5 \end{bmatrix},$$

which provides a good approximation of the order of magnitude of the entries in the LU factors of A

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -100 & 1001 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} -10 & 10 & -1000 \\ 0 & 1 & -1 \\ 0 & 0 & -98999 \end{bmatrix},$$

where u_{33} is only provided to five significant digits.

2.1. Paths through bipartite graphs. In practice, we need a more efficient way to compute the max-plus LU factors of a matrix. The following technique enables us to compute all the entries in a row of the upper factor $\mathcal{U}$ or column of the lower factor $\mathcal{L}$ by solving a single maximally weighted path problem. This is many times more efficient than computing each entry individually by evaluating the permanents in (2.3) and (2.4).

For $\mathcal{A} \in \mathbb{R}_{\max}^{n\times n}$, let $B(\mathcal{A})$ denote the bipartite graph with left vertices $X = \{x(1), \ldots, x(n)\}$, right vertices $Y = \{y(1), \ldots, y(n)\}$ and a directed edge $e : x(i) \mapsto y(j)$ of weight a_{ij}, whenever $a_{ij} \neq -\infty$. We say that a subset of edges $M \subseteq B(\mathcal{A})$ is a *matching* between $X_\ell = \{x(1), \ldots, x(\ell)\}$ and $Y_\ell = \{y(1), \ldots, y(\ell)\}$ if the edges in M only visit the vertices in $X_\ell \cup Y_\ell$ and if each vertex in $X_\ell \cup Y_\ell$ is incident to exactly one edge in M. The *weight* of a matching is the sum of its constituent edge weights. Note that the edges in a maximally weighted matching between X_ℓ and Y_ℓ will correspond to the entries in a permutation that attains the maximum in the expression (2.2) applied to the ℓth principal submatrix $\operatorname{perm}\big(\mathcal{A}[1\colon \ell, 1\colon \ell]\big)$. Given a bipartite graph $B(\mathcal{A})$ and a matching M between X_ℓ and Y_ℓ we form the *residual bipartite graph* $R(M)$ from $B(\mathcal{A})$ by reversing the direction of and minimising the weight of every edge in M. Hook and Tisseur's max-plus LU algorithm relies on the following result.

PROPOSITION 2.2. *Let $\mathcal{A} \in \mathbb{R}_{\max}^{n\times n}$ have max-plus LU factors $\mathcal{L} = (l_{ij})$ and $\mathcal{U} = (u_{ij})$, and let $B(\mathcal{A})$ be the bipartite graph associated with $\mathcal{A}$. Let M_ℓ be a maximally weighted matching between the left vertices $\{x(1), \ldots, x(\ell)\}$ and right vertices $\{y(1), \ldots, y(\ell)\}$ of $B(\mathcal{A})$. Then*

(i) *for $j \geq k$, u_{kj} is either the weight of the maximally weighted path through $R(M_{k-1})$ from $x(k)$ to $y(j)$, or $-\infty$ if there is no such path,*

(ii) *for $i > k$, l_{ik} is either the weight of the maximally weighted path through $R^T(M_k)$ from $x(k)$ to $x(i)$, or $-\infty$ if there is no such path.*

We refer to [19, Appendix A] for a proof of this result.

2.2. Hungarian matrices and fill paths. A matrix $H \in \mathbb{R}^{n\times n}$ is said to be *Hungarian* if its entries satisfy $|h_{ij}| \leq 1$ and $|h_{ii}| = 1$, for all $i, j = 1, \ldots, n$. It is well-known that for any matrix $A \in \mathbb{R}^{n\times n}$ of full structural rank there exists a permutation matrix P and diagonal matrices D_1, D_2 such that PD_1AD_2 is a Hungarian matrix and that such a scaling is a highly effective preprocessing step both for sparse direct solvers and for incomplete factorizations. The idea was originally introduced by Olschowka and Neumaier [28] in the mid 1990s. They proposed an optimal assignment problem to compute an ordering and scaling to reduce the need for pivoting within Gaussian elimination; their work was further developed by Duff and Koster [8] (see also Gupta and Ying [14]). The idea was subsequently extended to symmetric systems [7, 9] and over the last fifteen years or so, it has been adopted by the sparse linear algebra community for both nonsymmetric and symmetric problems (see, for example, [1, 15, 17, 18, 24, 31, 32]).

A max-plus matrix $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ is said to be *Hungarian* if its entries satisfy $\hbar_{ij} \leq 0$ and $\hbar_{ii} = 0$, for all $i, j = 1, \ldots, n$. Note that $H \in \mathbb{R}^{n\times n}$ is Hungarian if and only if $\mathcal{V}(H)$ is Hungarian. It is shown in [19, Thm. 3.8] that max-plus Hungarian matrices always admit max-plus LU factors.

Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be a Hungarian matrix and let $G(\mathcal{H})$ be the *precedence* graph of $\mathcal{H}$, that is the graph with vertices $\{1, \ldots, n\}$ and a directed edge $e : i \to j$ of weight $\hbar_{ij}$, whenever $\hbar_{ij} \neq -\infty$. A *path* σ *of length* ℓ from i to j in $G(\mathcal{H})$ is a sequence of $\ell + 1$ distinct vertices $i = \sigma(1), \sigma(2), \ldots, \sigma(\ell), \sigma(\ell+1) = j$, such that $\hbar_{\sigma(k),\sigma(k+1)} \neq -\infty$ for all $k = 1, \ldots, \ell$. The *weight* of a path $W(\sigma)$ is given by the sum of its edge weights. We allow paths of zero length that consist of a single vertex and have zero weight but we do not allow paths to visit the same vertex more than once. Let $\Sigma(i, j, \mathcal{H})$ be the set of all paths in $G(\mathcal{H})$ from i to j. A path σ of length ℓ from i to j is a *fill path* if $\sigma(1) = i$, $\sigma(k) < \min\{i, j\}$ for $k = 2, \ldots, \ell$ and $\sigma(\ell + 1) = j$. Let $\Sigma^F(i, j, \mathcal{H})$ be the set of all fill paths from i to j in the graph $G(\mathcal{H})$. Clearly, $\Sigma^F(i, j, \mathcal{H}) \subseteq \Sigma(i, j, \mathcal{H})$. The Hungarian property allows us a neater description of the max-plus LU factors in terms of fill paths through $G(\mathcal{H})$ as follows.

PROPOSITION 2.3. *Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be Hungarian and have max-plus LU factors $\mathcal{L} = (\ell_{ij})$ and $\mathcal{U} = (u_{ij})$. Then*

$$\ell_{ij} = \begin{cases} \max_{\sigma\in\Sigma^F(i,j,\mathcal{H}^T)} W(\sigma) & \text{for } i \geq j, \\ -\infty & \text{otherwise,} \end{cases} \tag{2.6}$$

$$u_{ij} = \begin{cases} \max_{\sigma\in\Sigma^F(i,j,\mathcal{H})} W(\sigma) & \text{for } i \leq j, \\ -\infty & \text{otherwise,} \end{cases} \tag{2.7}$$

where we use the convention $\max_{\sigma\in\Sigma^F(i,j,\mathcal{H})} W(\sigma) = -\infty$, *whenever* $\Sigma^F(i, j, \mathcal{H}) = \emptyset$, *i.e., whenever there is no fill path from i to j.*

Proof. Since $\mathcal{H}$ is Hungarian we have $\hbar_{ij} \leq 0$ for all $i, j = 1, \ldots, n$ and any matching between vertices in $B(\mathcal{H})$ will have non-positive weight. Therefore any weight zero matching will automatically be a maximally weighted matching. A weight zero matching between the left vertices $\{x(1), \ldots, x(k)\}$ and the right vertices $\{y(1), \ldots, y(k)\}$ is given by $M_k = \{x(1) \mapsto y(1), \ldots, x(k) \mapsto y(k)\}$. We can therefore use this matching to compute the max-plus LU factors as in Proposition 2.2.

For $j \geq k$, consider a path ς through $R(M_{k-1})$ from $x(k)$ to $y(j)$. Since $R(M_{k-1})$ is a bipartite graph, ς must alternate between the left and right vertices. Because the only right to left edges in $R(M_{k-1})$ come from the reversed edges in M_{k-1}, if σ visits $y(i)$ then it must visit $x(i)$ on its next step for $i = 1, \ldots, k-1$, and σ may not visit any $y(i)$ with $i > k - 1$ until its final step as otherwise it would have no edges available to continue moving along. Therefore, $\varsigma = \big(x(i_1), y(i_2), x(i_2), y(i_3), \ldots, x(i_\ell), y(i_{\ell+1})\big)$, for some sequence of distinct indices $i_1, \ldots, i_{\ell+1}$ with $i_1 = k$, $i_1 \ldots, i_\ell \leq k$ and $i_{\ell+1} = j$. Notice that the sequence $i_1, \ldots, i_{\ell+1}$ gives us a path $\sigma = (i_1, \ldots, i_{\ell+1})$ in $G(\mathcal{H})$, which satisfies the conditions to be a fill path from k to j. The weight of the path in the bipartite graph ς is equal to the sum of its left to right edge weights and its right to left edge weights. But because every edge in M_{k-1} has weight zero, the right to left edges make zero contribution to this sum. The weight of the path is thus given by

$$W(\varsigma) = \sum_{t=1}^{\ell} \hbar_{i(t),i(t+1)},$$

which is exactly the same as the weight of the fill path σ through the precedence graph. Therefore, for every path ς through $R(M_{k-1})$ from $x(k)$ to $y(j)$ there is

a fill path σ through $G(\mathcal{H})$ from k to j with the same weight. The converse is also true. If $\sigma = (i_1, \ldots, i_{\ell+1})$ is a fill path through $G(\mathcal{H})$ from k to j then $\varsigma = \big(x(i_1), y(i_2), x(i_2), y(i_3), \ldots, x(i_\ell), y(i_{\ell+1})\big)$ is a path through $R(M_{k-1})$ from $x(k)$ to $y(j)$, with the same weight as σ. Thus the characterization of the entries in the upper triangle of $\mathcal{U}$ in Proposition 2.2(i) and that in (2.6) are identical. The proof for the lower triangular factors is the same only we work with the graphs generated from $\mathcal{H}^T$. □

3. The max-plus IC factorization. We now focus on symmetric positive-definite matrices $A \in \mathbb{R}^{n\times n}$. For such matrices, there exists a unique lower triangular matrix $L \in \mathbb{R}^{n\times n}$ such that $A = LL^T$; this is the Cholesky factorization of A. The following result is from Olschowka and Neumaier [28].

LEMMA 3.1. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric positive-definite matrix and let $D \in \mathbb{R}^{n\times n}$ be the diagonal matrix with diagonal entries $d_{ii} = 1/\sqrt{a_{ii}}$, $i = 1, \ldots, n$, then*

$$H = DAD,$$

is a symmetric positive-definite Hungarian matrix.

Let $H \in \mathbb{R}^{n\times n}$ be a symmetric positive-definite Hungarian matrix with the Cholesky factorization $H = LL^T$. If $\widetilde{D} \in \mathbb{R}^{n\times n}$ is the diagonal matrix with diagonal entries $\widetilde{d}_{ii} = l_{ii} > 0$ then

$$\widetilde{L} = L\widetilde{D}^{-1}, \quad \widetilde{U} = \widetilde{D}L^T$$

provides an LU factorization of H, $H = \widetilde{L}\widetilde{U}$, with $\widetilde{l}_{ii} = 1$ for all $i = 1, \ldots, n$. We can therefore use (2.5) to approximate the moduli of the entries of $\widetilde{L}$ and $\widetilde{U}$ in terms of the max-plus LU factors $\mathcal{L}, \mathcal{U} \in \mathbb{R}_{\max}^{n\times n}$ of the valuation $\mathcal{H} = \mathcal{V}(H)$ and since $\mathcal{H}$ is Hungarian, these factors always exist. Also, since H is symmetric, $\mathcal{H}$ is also symmetric and (2.6) becomes

$$\mathcal{l}_{ij} = \mathcal{u}_{ji} = \begin{cases} \max_{\sigma\in\Sigma^F(i,j,\mathcal{H})} W(\sigma) & \text{for } i \geq j, \\ -\infty & \text{otherwise,} \end{cases} \tag{3.1}$$

which gives the max-plus approximation

$$\mathcal{V}(\widetilde{L}) \approx \mathcal{L}, \quad \mathcal{V}(\widetilde{U}) \approx \mathcal{U}.$$

Now, by construction, $\widetilde{u}_{ii} = l_{ii}^2$ so that $\widetilde{d}_{ii} = l_{ii} = \big(\widetilde{u}_{ii}\big)^{1/2}$ for all i. Note from (2.6) that the diagonal entries of $\mathcal{U}$ are all equal to zero, i.e., $\mathcal{u}_{ii} = 0$ for all $i = 1, \ldots, n$. This is because the length zero path from i to itself is a fill path of length zero and there are no edges or paths with weight greater than zero. Thus the max-plus approximation gives $\widetilde{d}_{ii} \approx 1$ for all i, or equivalently, $\widetilde{D} \approx I$. We therefore have $L = \widetilde{L}\widetilde{D} \approx \widetilde{L}$, which gives the following max-plus approximation of the modulus of the entries in the Cholesky factor

$$\mathcal{V}(L) \approx \mathcal{L} \iff \log|l_{ij}| \approx \mathcal{l}_{ij},\ 1 \leq i, j \leq n. \tag{3.2}$$

Given the valuation $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ of a symmetric positive-definite Hungarian matrix $H \in \mathbb{R}^{n\times n}$, we say that $\mathcal{L}$ with entries given by (3.1) is the *max-plus Cholesky factor* of $\mathcal{H}$.

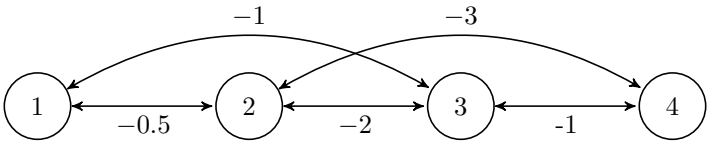


FIG. 3.1. *Precedence graph $G(\mathcal{H})$ for the matrix of Example 3.2.*

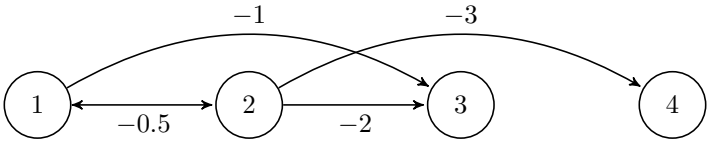


FIG. 3.2. *Precedence graph $G\big(\mathcal{H}(2)\big)$ for the matrix of Example 3.2.*

EXAMPLE 3.2. *Consider*

$$H = \begin{bmatrix} 1 & 10^{-0.5} & 10^{-1} & 0 \\ 10^{-0.5} & 1 & 10^{-2} & 10^{-3} \\ 10^{-1} & 10^{-2} & 1 & 10^{-1} \\ 0 & 10^{-3} & 10^{-1} & 1 \end{bmatrix}, \quad \mathcal{H} = \mathcal{V}(H) = \begin{bmatrix} 0 & -0.5 & -1 & -\infty \\ -0.5 & 0 & -2 & -3 \\ -1 & -2 & 0 & -1 \\ -\infty & -3 & -1 & 0 \end{bmatrix},$$

where H is Hungarian and symmetric positive definite.

Figure 3.1 displays the precedence graph $G(\mathcal{H})$. Suppose that we want to compute the 2nd column of $\mathcal{L}$. From (3.1) we have $\ell_{12} = -\infty$, $\ell_{22} = 0$, $\ell_{32} = \max_{\sigma\in\Sigma^F(2,3,\mathcal{H})} W(\sigma)$, where $\Sigma^F(2,3,\mathcal{H}) = \{(2,3),(2,1,3)\}$. Since $W(2,3) = -2$ and $W(2,1,3) = -1.5$ we have $\ell_{32} = -1.5$. Similarly, $\ell_{42} = \max_{\sigma\in\Sigma^F(2,4,\mathcal{H})} W(\sigma)$, where $\Sigma^F(2,4,\mathcal{H}) = \{(2,4)\}$ and since $W(2,4) = -3$, we have $\ell_{42} = -3$. Note that $\Sigma(2,4,\mathcal{H})$ also contains the paths $(2,3,4)$ and $(2,1,3,4)$. Note also that $W(2,1,3,4) = -2.5$, which is greater than $W(2,4)$, but we do not count the weight of the path $(2,1,3,4)$ as it is not a fill path.

The remaining columns of $\mathcal{L}$ can be computed in the same way to give the max-plus Cholesky factor of $\mathcal{H}$,

$$\mathcal{L} = \begin{bmatrix} 0 & \infty & \infty & -\infty \\ -0.5 & 0 & \infty & \infty \\ -1 & -1.5 & 0 & \infty \\ -\infty & -3 & -1 & 0 \end{bmatrix}.$$

This provides a good approximation of the order of magnitude of the moduli of the entries in the Cholesky factor of H

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.316 & 0.949 & 0 & 0 \\ 0.1 & -0.023 & 0.995 & 0 \\ 0 & 0.001 & 0.101 & 0.995 \end{bmatrix}, \quad \mathcal{V}(L) = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -0.5 & -0.023 & -\infty & -\infty \\ -1 & -1.642 & -0.002 & -\infty \\ -\infty & -2.977 & -0.998 & -0.002 \end{bmatrix}.$$

We now define $\mathcal{H}(k) \in \mathbb{R}_{\max}^{n\times n}$, $1 \le k \le n$ to be the matrix with entries given by

$$\mathcal{H}(k)_{ij} = \begin{cases} \mathcal{H}_{ij} & \text{for } i \le k, \\ -\infty & \text{otherwise.} \end{cases}$$

Thus $G\big(\mathcal{H}(k)\big)$ is the graph with vertices $\{1,\dots,n\}$ that contains all edges from $\{1,\dots,k\}$ to itself and all edges from $\{1,\dots,k\}$ to $\{k+1,\dots,n\}$ but no edges from $\{k+1,\dots,n\}$ to itself or from $\{k+1,\dots,n\}$ to $\{1,\dots,k\}$. This construction is illustrated for the matrix of Example 3.2 in Figure 3.2.

LEMMA 3.3. *Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be the valuation of a symmetric positive-definite Hungarian matrix then*

$$\Sigma^F(k,i,\mathcal{H}) = \Sigma\big(k,i,\mathcal{H}(k)\big),$$

for $k = 1,\dots,n$ and $i = k,\dots,n$.

Proof. The set $\Sigma^F(k,i,\mathcal{H})$ contains the zero length path $\sigma = (k)$ if and only if $i = k$. Likewise for $\Sigma\big(k,i,\mathcal{H}(k)\big)$. Now suppose that $\sigma \in \Sigma^F(k,i,\mathcal{H})$ is a path of length $\ell > 0$. Since σ is a fill path from k to i it must satisfy $\sigma(1) = k$, $\sigma(j) < k$ for $j = 2,\dots,\ell$ and $\sigma(\ell+1) = i$. Therefore σ traverses $\ell - 1$ edges between the vertices $\{1,\dots,k\}$ then traverses an edge from $\{1,\dots,k\}$ to i, since all of these edges are also contained in $G\big(\mathcal{H}(k)\big)$ we have $\sigma \in \Sigma\big(k,i,\mathcal{H}(k)\big)$. Conversely, suppose that $\sigma \in \Sigma\big(k,i,\mathcal{H}(k)\big)$ is a path of length $\ell > 0$. Since σ is a path from k to i it must satisfy $\sigma(1) = k$ and $\sigma(\ell+1) = i$. However, since there are no edges from vertices $\{k+1,\dots,n\}$ to $\{1,\dots,n\}$ in $G\big(\mathcal{H}(k)\big)$, the path σ can only visit a vertex in $\{k+1,\dots,n\}$ as its final vertex so that $\sigma(j) \le k$ for $j = 2,\dots,\ell$. Moreover, since $\sigma(1) = k$ and σ is a path and, as such, must consist of a sequence of distinct vertices, we have $\sigma(j) < k$ for $j = 2,\dots,\ell$ and therefore $\sigma \in \Sigma^F(k,i,\mathcal{H})$. ∎

COROLLARY 3.4. *Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be the valuation of a symmetric positive-definite Hungarian matrix and let $\mathcal{L}$ be the Cholesky factor of $\mathcal{H}$, then for $i \ge k$*

$$\mathcal{l}_{ik} = \max_{\sigma\in\Sigma^F(k,i,\mathcal{H})} W(\sigma) = \max_{\sigma\in\Sigma(k,i,\mathcal{H}(k))} W(\sigma).$$

To compute the kth column of $\mathcal{L}$ it is therefore sufficient to compute the weight of the maximally weighted path from k to i for all $i = k+1,\dots,n$ in $\mathcal{H}(k)$. Since the entries of $\mathcal{H}$ are non-positive this can be done using Dijkstra's algorithm [5] with worst case cost $O\big(E_k + n\log(n)\big)$, where E_k is the number of nonzero entries in $\mathcal{H}(k)$. The algorithm is given as Algorithm 1. For efficiency it uses a priority heap data structure that stores indices with an associated priority. The operation *push(heap, index, priority)* adds or updates an index-priority pair in the heap, whilst the operation *(index, priority) = pop_max(heap)* returns and removes the index-priority pair with maximum priority.

3.1. Max-plus IC preconditioner pattern. We can use the max-plus Cholesky factor $\mathcal{L} \in \mathbb{R}_{\max}^{n\times n}$ of $\mathcal{V}(H)$ to construct a sparsity pattern for an IC preconditioner for a symmetric positive-definite Hungarian matrix $H \in \mathbb{R}^{n\times n}$ as follows. If we want the IC pattern to include the positions of all of the entries in the exact Cholesky factor L of H that are greater in modulus than some drop tolerance $\epsilon > 0$, then (3.2) suggests using the pattern $S \in \{0,1\}^{n\times n}$ given by

$$s_{ij} = \begin{cases} 1 & \text{if } \mathcal{l}_{ij} \ge \log\epsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

The total number of nonzero entries per column can also be restricted to an integer m, by setting $s_{ij} = 1$ for only the m largest positions in the jth column, as predicted by (3.2).

Algorithm 1 Given the valuation $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ of a real symmetric positive-definite Hungarian matrix and an integer k, $1 \le k \le n$, this algorithm computes the kth column of the max-plus Cholesky factor $\mathcal{L}$ of $\mathcal{H}$.

```
1: set d_i = −∞ and checked_i =false, 1 ≤ i ≤ n
2: initialize heap; push(heap, k, 0)
3: while heap is non empty do
4:     (i, d_i) = pop_max(heap)
5:     set checked_i =true
6:     if i ≤ k then
7:         for all j such that ℏ_ij ≠ −∞ and checked_j =false do
8:             d_cand = d_i + ℏ_ij
9:             if d_cand > d_j then
10:                set d_j = d_cand
11:                push(heap, j, d_cand)
12:            end if
13:        end for
14:    end if
15:    if i ≥ k then
16:        ℓ_ik = d_i
17:    end if
18: end while
```

If we are computing the max-plus Cholesky factor of $\mathcal{H}$ in order to predict the positions of large entries in the Cholesky factor L of H then we can speed up Algorithm 1 by terminating it early. Algorithm 2 calculates the positions of the m largest entries in the kth column of $\mathcal{L}$. If there are fewer than m entries greater than some chosen threshold $\log\epsilon$ then it will instead return the positions of the $r < m$ entries that are greater than $\log\epsilon$. The worst-case cost of this algorithm is $O\big(E_{k,m,\epsilon} + V_{k,m,\epsilon}\log(n)\big)$, where $E_{k,m,\epsilon}$ is the total number of edges explored in line 6 of Algorithm 2 and $V_{k,m,\epsilon}$ is the number of times the algorithm passes round the while loop that begins on line 3. A possible approximation of these quantities for the case $\epsilon = 0$ is given by

$$E_{k,m,0} = E_k\frac{m}{n-k}, \qquad V_{k,m,0} = \frac{km}{n-k} + m.$$

Here the assumption is that vertices are examined and checked in a uniform random order and that the number of nonzero entries per row in H is constant.

It is interesting to compare the max-plus pattern (3.3) with the level of fill IC(k) pattern. The IC(k) pattern $P \in \{0,1\}^{n\times n}$ can be expressed as

$$p_{ij} = \begin{cases} 1 & \text{if there is a fill path of length} \le k \text{ from } i \text{ to } j \text{ through } G(\mathcal{H}),\\ 0 & \text{otherwise,}\end{cases}$$

whereas, using (3.1), the max-plus IC pattern $S \in \{0,1\}^{n\times n}$ in (3.3) can be expressed as

$$s_{ij} = \begin{cases} 1 & \text{if there is a fill path of weight} \ge \log\epsilon \text{ from } i \text{ to } j \text{ through } G(\mathcal{H}),\\ 0 & \text{otherwise.}\end{cases}$$

The level of fill approach drops entries that correspond to longer paths, while the max-plus approach drops entries that correspond to paths with less weight. By taking this extra information into account the max-plus approach has the ability to produce more effective preconditioners, by dropping some smaller entries with lower level of fill and including some larger entries with higher level of fill. Note that in the special case that $H \in \mathbb{R}^{n\times n}$ has $h_{ii} = 1$, $1 \le i \le n$ and $h_{ij} = \gamma < 1$ for all other nonzero positions, then the IC(k) pattern will be identical to the max-plus pattern chosen using $\epsilon = \gamma^{k+1}$.

Algorithm 2 Given the valuation $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ of a real symmetric positive-definite Hungarian matrix, a tolerance $\epsilon > 0$, and two integers m, k, $1 \le m, k \le n$, this algorithm computes the kth column of a pattern matrix with 0 and 1 entries such that there are 1's in the $r \le m$ entries corresponding the largest entries in the kth column of the max-plus Cholesky factor $\mathcal{L}$ of $\mathcal{H}$ that are greater than $\log \epsilon$.

```
for i = 1, ..., n set d_i = -∞ and checked_i = false
initialize heap; push(heap, k, 0)
set p_ik = 0 for i = 1, 2, ..., n; set r = 0
while r < m do
    (i, d_i) = pop_max(heap)
    if d_i < log ε then exit
    set checked_i = true
    if i ≤ k then
        for all j such that ℋ_ij ≠ -∞ and checked_j = false do
            d_cand = d_i + ℋ_ij
            if d_cand > d_j then
                set d_j = d_cand
                push(heap, j, d_cand)
            end if
        end for
    end if
    if i ≥ k then
        p_ik = 1
        r = r + 1
    end if
end while
```

4. Numerical Results. We present numerical results for problems taken from the UFL Sparse Matrix Collection [4]. We select all symmetric positive-definite matrices of order $n > 5000$ except those that are diagonal or represent minor variations on other matrices. This gives a set of 132 problems. In each test, the matrix A is reordered, scaled and, if necessary, shifted to avoid breakdown of the factorization so that the incomplete factorization of

$$\widehat{A} = DQ^T AQD + \alpha I$$

is computed, where Q is a permutation matrix, D is a diagonal scaling matrix with entries $d_{ii} = 1/\sqrt{(Q^T AQ)_{ii}}$ and α is a non-negative shift. The permutation matrix Q is computed using the Sloan profile reduction ordering algorithm [29, 36, 37]. This ordering is used since, in our experience, it frequently leads to a reduction in the number of conjugate gradient iterations [34, 35]. Preconditioned conjugate gradients

(PCG) is applied to the original matrix A (so that the incomplete preconditioner is $(\widehat{L}\widehat{L}^T)^{-1}$ with $\widehat{L} = QD^{-1}L$). The strategy for choosing α is as described in [34] (see also [25]). The max-plus IC factorization is started with $\alpha = 0$ but if a zero (or negative) pivot is encountered, a nonzero α is employed (the initial value used in our experiments is 0.001) and the factorization restarted. This process may need to be repeated more than once, with α increased (normally by a factor of 2) each time the factorization breaks down. For our test set, we found that the largest shift needed by the max-plus IC factorization was 0.064.

We use the implementation `MI21` of PCG provided by the HSL mathematical software library [20]. For each problem, we terminate the computation if a limit of either 10,000 iterations or 10 minutes is reached. The PCG algorithm is considered to have converged on the ith step if

$$\frac{\|r_i\|_2}{\|r_0\|_2} \leq 10^{-10},$$

where r_i is the current residual vector and $r_0 = b - Ax_0$ is the initial residual. In all our experiments, we take the initial solution guess to be $x_0 = 0$ and choose the right-hand side b so that the solution is the vectors of 1's. If PCG fails to converge within our chosen limits, the result is recorded as a failure. All runs are performed on a dual socket E5-2695 v3 machine using the Intel Compiler Suite v16.0.1. We use the MKL's sparse triangular solve and matrix-vector routines to apply the preconditioner and calculate matrix-vector products Ax.

To measure the performance of PCG, we use the following statistics:

nitr: the number of iterations required for PCG to converge.

ma_{pcg}: the total number of memory accesses to perform PCG, given by

$$ma_{pcg} = nitr \times (nnz(A) + 2\, nnz(L)), \tag{4.1}$$

where $nnz(A)$ and $nnz(L)$ are the number of entries in the lower triangle of A and in the (incomplete) Cholesky factor, respectively. This represents a matrix-vector multiplication, and a forwards and a backwards solve with L at each iteration. In an ideal implementation, runtime would be proportional to ma_{pcg}.

We compare the new max-plus IC preconditioner with the following preconditioners.

Diagonal: Equivalent (in exact arithmetic) to no preconditioning as our pre-scaling results in all diagonal entries being 1.0.

IC(0): Incomplete Cholesky based on the sparsity pattern of A. A drop tolerance $\delta = 10^{-3}$ is applied in a post-factorization filtering step (so that all entries in the computed factor that are of absolute value less than δ are discarded).

IC(1): Incomplete Cholesky based on the pattern of A plus one level of fill. A drop tolerance $\delta = 10^{-3}$ is applied in a post-factorization filtering step.

`HSL_MI28`: A limited memory IC preconditioner developed by Scott and Tůma [34, 35]. We use the default drop tolerances of 10^{-3} and 10^{-4} and allow up to 10 fill entries in each column of the incomplete factor (the software's parameters *lsize* and *rsize* that control the memory usage and sparsity of the factor are both set to 10).

Note that, for each problem and each algorithm, a different shift α may be needed. A nonzero value is only used if, during the construction of the preconditioner, a zero or negative pivot is encountered.

We do not report detailed times to form the preconditioner as the purpose of this study is to evaluate the numerical quality of the preconditioner, rather than to develop an efficient implementation. However, to give an indication of runtimes for our current code, we comment that for each of our test problems, our basic max-plus serial implementation takes less than 7 minutes to construct the preconditioner. The slowest max-plus time that gives a preconditioner that leads to PCG converging within our chosen limits is for problem Janna/Bump_2911. In this case, our prototype code takes approximately 260 seconds to construct the preconditioner, followed by 58 seconds to run PCG, which compares to around 14 seconds for constructing IC(0), followed by 90 seconds to run PCG. For IC(1) (respectively, `HSL_MI28`) the corresponding times are 30 seconds (respectively, 55 seconds) for constructing the preconditioner and 95 seconds (respectively, 35 seconds) to run PCG. The max-plus implementation can potentially be accelerated through the use of parallel processing as the pattern of each column can be calculated independently but implementing this efficiently is non trivial. We observe that the patterns of the columns of IC(1) (and more generally, IC(k)) can also be computed in parallel [21]) but `HSL_MI28` is a serial approach.

4.1. Max-plus parameters. Our algorithm for determining the incomplete max-plus pattern has three parameters: m, the maximum number of entries per column, ϵ, the max-plus drop tolerance, and δ, a tolerance that is applied to filter the final L factor. To be consistent with the IC(0) and IC(1) preconditioners, the later is set to 10^{-3}.

To establish suitable settings for m and ϵ, we perform experiments on a subset of 15 matrices. This subset was chosen by ordering the test set in order of $nnz(A)$ and then choosing (approximately) every 10th example. We present results in Table 4.1 for $m = 10$ and 20 with $\epsilon = 10^{-4}$, 10^{-5} and 10^{-6}. For comparison, the results for the other IC approaches are shown in Table 4.2. As m increases and ϵ decreases, more entries are included in the factors. The results show that typically the relaxation of ϵ has little effect on the size of the factors, although for some problems using $\epsilon \leq 10^{-5}$ can significantly decrease the number of iterations required (e.g. AMD/G2_circuit, ND/nd6k, Janna/Bump_2911). We therefore choose to use $\epsilon = 10^{-6}$ in the rest of this paper. We observe that we also experimented with using $\epsilon = 0.0$. For a small number of examples, this can further reduce the number of iterations (e.g. for Williams/cant, the count is cut from 2016 to 1617) but the time to compute the preconditioner increases significantly (for many of our tests, compared to using $\epsilon = 10^{-6}$, the time for $\epsilon = 0.0$ increases by more than 50 percent and this is not fully offset by the reduction in the iteration count).

The effect of m is much more dramatic, both in terms of increased factor size and decreased number of iterations. When we consider the balance of these qualities in the number of memory accesses ma_{pcg}, the best result can go in either direction. As $m = 10$ always gives the sparsest factors, we choose $m = 10$ for the remainder of this paper. The combination $m = 10, \epsilon = 10^{-6}$ has the property that (on these 15 matrices) the size of the factors is always smaller or commensurate with those produced by `HSL_MI28` with the selected settings for its input parameters.

4.2. Comparison with other IC preconditioners. To assess the performance of the different preconditioners on our test set of 132 problems, we employ performance profiles [6]. The performance ratio for an algorithm on a particular problem is the performance measure for that algorithm divided by the smallest performance measure for the same problem over all the algorithms being tested (here we are assuming that the performance measure is one for which smaller is better,

TABLE 4.1

Results for various values of the max-plus parameters m and ϵ for a subset of 15 matrices. Entries in bold are within 10% of the best.

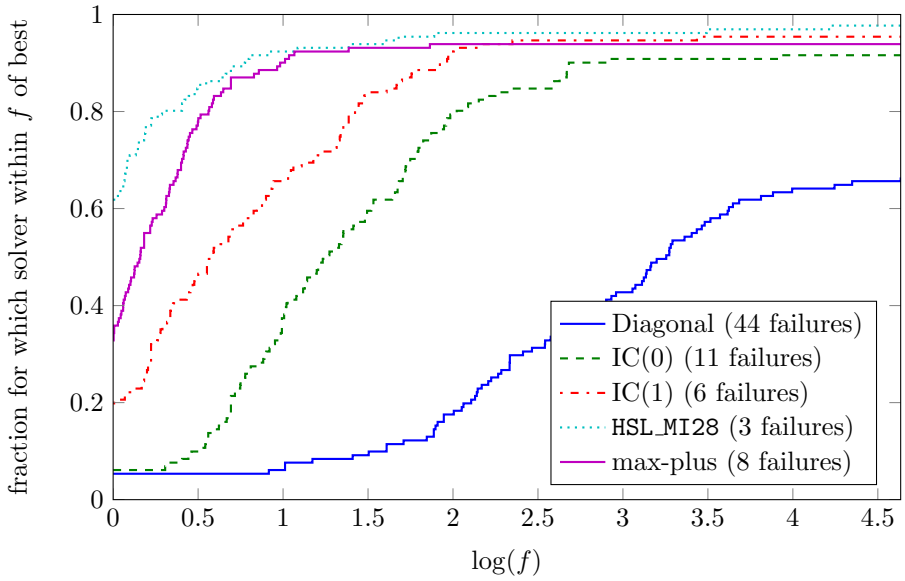
Problem		$nnz(L) \times 10^6$			$nitr$			$ma_{pcg} \times 10^9$		
	$\epsilon =$	10^{-4}	10^{-5}	10^{-6}	10^{-4}	10^{-5}	10^{-6}	10^{-4}	10^{-5}	10^{-6}
Pothen/bodyy5	$m = 10$	**0.08**	**0.08**	**0.08**	6	**5**	**5**	**0.001**	**0.001**	**0.001**
	$m = 20$	**0.08**	**0.08**	**0.08**	6	**5**	**5**	**0.001**	**0.001**	**0.001**
HB/bcsstk18	$m = 10$	**0.11**	**0.11**	**0.11**	80	80	80	0.025	0.025	0.025
	$m = 20$	0.13	0.13	0.13	63	**54**	**53**	0.021	**0.019**	**0.018**
GHS_psdef/minsurfo	$m = 10$	**0.27**	0.30	0.32	9	7	**6**	0.006	**0.005**	**0.005**
	$m = 20$	**0.27**	0.30	0.32	9	7	**6**	0.006	**0.005**	**0.005**
GHS_psdef/apache1	$m = 10$	**0.82**	**0.82**	**0.83**	140	141	140	**0.274**	**0.276**	**0.277**
	$m = 20$	0.96	0.99	1.03	**126**	**127**	**117**	**0.281**	**0.292**	**0.277**
AMD/G2_circuit	$m = 10$	**1.26**	1.49	1.61	123	89	84	0.364	**0.304**	**0.308**
	$m = 20$	**1.28**	1.56	1.78	123	89	**70**	0.370	0.317	**0.280**
Rothberg/cfd2	$m = 10$	**2.78**	**2.78**	**2.78**	598	599	599	**4.29**	**4.30**	**4.30**
	$m = 20$	3.83	3.83	3.83	**526**	**525**	**527**	4.88	4.87	4.88
Williams/cant	$m = 10$	**2.52**	**2.52**	**2.52**	**1899**	**1902**	**1899**	**13.4**	**13.5**	**13.4**
	$m = 20$	3.05	3.05	3.05	2115	2130	2106	17.2	17.3	17.1
DNVS/shipsec5	$m = 10$	**3.89**	**3.90**	**3.90**	**171**	**168**	**168**	**2.21**	**2.17**	**2.17**
	$m = 20$	4.61	4.66	4.67	189	189	189	2.72	2.74	2.74
Williams/consph	$m = 10$	**3.26**	**3.26**	**3.26**	182	183	182	**1.74**	**1.75**	**1.74**
	$m = 20$	3.95	3.95	3.95	**158**	**157**	**158**	**1.73**	**1.72**	**1.73**
ND/nd6k	$m = 10$	**0.67**	**0.70**	**0.71**	218	178	176	1.05	0.863	0.860
	$m = 20$	**0.67**	**0.70**	**0.73**	218	179	**156**	1.05	0.868	**0.766**
Boeing/pwtk	$m = 10$	**7.13**	**7.13**	**7.13**	3058	3258	3244	61.7	65.8	65.5
	$m = 20$	8.56	8.56	8.56	**2109**	**2103**	**2104**	**48.6**	**48.5**	**48.5**
Schenk_AFE/af_shell3	$m = 10$	**13.6**	**13.6**	**13.6**	434	434	434	**15.8**	**15.8**	**15.8**
	$m = 20$	18.3	18.3	18.3	**327**	**325**	**325**	**14.9**	**14.8**	**14.8**
Oberwolfach/bone010	$m = 10$	**43.8**	**43.8**	**43.8**	2054	2040	2039	**255.**	**253.**	**253.**
	$m = 20$	52.3	52.3	52.3	**1862**	**1853**	**1890**	262.	261.	266.
GHS_psdef/audikw_1	$m = 10$	**47.1**	**47.1**	**47.1**	**952**	**958**	**956**	**127.**	**128.**	**128.**
	$m = 20$	55.1	55.1	55.1	**899**	**896**	**896**	**134.**	**134.**	**134.**
Janna/Bump_2911	$m = 10$	**70.7**	**70.9**	**70.9**	190	189	189	**39.3**	**39.2**	**39.2**
	$m = 20$	82.0	82.8	83.0	213	**170**	**170**	48.9	**39.3**	**39.3**

TABLE 4.2

Results for other IC preconditioner for our subset of 15 matrices. - indicates failure to converge within our set limits.

Problem	$nnz(L) \times 10^6$				$nitr$				$ma_{pcg} \times 10^9$			
	Diag.	IC(0)	IC(1)	MI28	diag	IC(0)	IC(1)	MI28	Diag.	IC(0)	IC(1)	MI28
Pothen/bodyy5	0.02	0.06	0.08	0.08	186	67	29	5	0.021	0.014	0.007	0.001
HB/bcsstk18	0.01	0.07	0.11	0.13	1343	332	153	35	0.140	0.073	0.046	0.012
GHS_psdef/minsurfo	0.04	0.12	0.16	0.32	103	31	20	6	0.021	0.011	0.009	0.005
GHS_psdef/apache1	0.08	0.29	0.45	0.92	479	285	286	127	0.227	0.255	0.348	0.274
AMD/G2_circuit	0.15	0.44	0.58	1.77	1524	471	274	66	1.13	0.619	0.438	0.262
Rothberg/cfd2	0.12	1.60	3.45	2.84	5824	539	390	430	10.8	2.60	3.32	3.13
Williams/cant	0.06	2.03	4.37	2.66	4133	2703	1561	1279	8.93	16.5	16.8	9.40
DNVS/shipsec5	0.18	3.39	5.12	5.15	3259	511	279	72	17.9	6.09	4.29	1.11
Williams/consph	0.08	2.85	5.86	3.84	1307	242	134	103	4.20	2.12	1.98	1.11
ND/nd6k	0.02	1.16	2.26	0.69	-	551	100	194	-	3.18	0.798	0.938
Boeing/pwtk	0.22	5.77	8.37	8.00	-	6276	2249	1195	-	110.	51.0	26.2
Schenk_AFE/af_shell3	0.50	9.04	11.5	14.1	3330	872	561	323	33.5	23.7	18.0	12.0
Oberwolfach/bone010	0.99	36.3	77.2	46.1	9978	1847	1186	1438	382.	201.	226.	185.
GHS_psdef/audikw_1	0.94	39.2	76.5	48.7	7009	1317	541	475	289.	155.	104.	64.9
Janna/Bump_2911	2.91	62.8	101.	78.4	8821	357	247	114	628.	68.2	66.2	25.3

FIG. 4.1. *Performance profile comparing nitr across various preconditioners.*



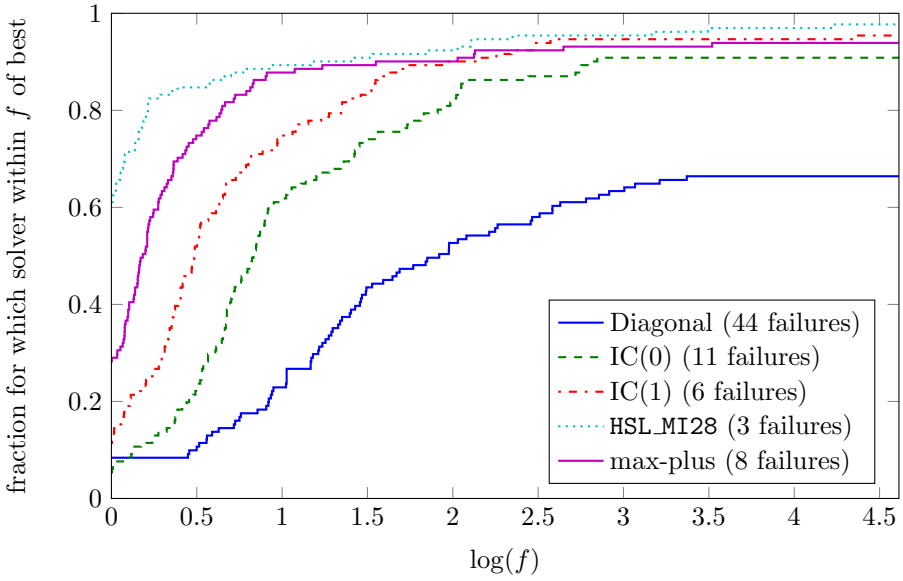
for example, the iteration count). The performance profile is the set of functions $\{p_i(f) : f \in [1, \infty)\}$, where $p_i(f)$ is the proportion of problems where the performance ratio of the i-th algorithm is at most f. Thus $p_i(f)$ is a monotonically increasing function taking values in the interval $[0, 1]$. In particular, $p_i(1)$ gives the fraction of the examples for which algorithm i is the winner (that is, the best according to the performance measure), while if we assume failure to solve a problem (for example, through the maximum iteration count or time limit being exceeded) is signaled by a performance measure of infinity, $p_i^* := \lim_{f\to\infty} p_i(f)$ gives the fraction for which algorithm i is successful.

Figures 4.1 and 4.2 present performance profiles for the iteration counts *nitr* and memory accesses ma_{pcg}, respectively. We use a logarithmic scale in order to observe the performance of the algorithms over a large range of f while still being able to discern in some detail what happens for small f. The highest number of failures (a third of the examples) results from using diagonal preconditioning while `HSL_MI28` has only 3 failures. In terms of both iteration counts and memory accesses, `HSL_MI28` has the best performance but the max-plus preconditioner also performs well and outperforms the level-based preconditioners.

5. Alternative approaches to the max-plus computation. As already observed, a potentially attractive feature of Algorithm 1 is that each column of $\mathcal{L}$ can be computed independently. However, because each of these independent computations requires access to all, or at least a large part of, the matrix $\mathcal{H}$, it may be difficult to fully leverage this parallelism in practice. Further, on many-core architectures, as memory bandwidth is a limiting factor for Algorithm 1, achieving a high level of parallel efficiency may be difficult. A possible alternative is the following serial method, which is analogous to classical Gaussian elimination.

For a Hungarian max-plus matrix $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ with precedence graph $G(\mathcal{H})$, we say that a path $\sigma = \sigma(1), \ldots, \sigma(\ell)$ is a *below k path* if $\sigma(t) \le k$ for $t = 2, \ldots, \ell - 1$.

FIG. 4.2. *Performance profile comparing ma_{pcg} in (4.1) across various preconditioners.*



Let $\Sigma_k(i,j,\mathcal{H})$ denote the set of all below k paths from i to j through $G(\mathcal{H})$. Below zero paths are given by paths of length one, that is,

$$\Sigma_0(i,j,\mathcal{H}) = \{(i,j) : \mathcal{H}_{ij} \neq -\infty\}.$$

Recall that a path σ from i to j is a fill path if $\sigma(t) \leq \min\{i,j\}$ for $t = 2,\ldots,\ell-1$. We denote by $\Sigma^F(i,j,\mathcal{H})$ the set of all fill paths in $G(\mathcal{H})$ from i to j. Note that if σ is a below $\min\{i,j\}$ path from i to j then it is also a fill path. We can therefore use (3.1) to express the entries of the max-plus Cholesky factor $\mathcal{L}$ of $\mathcal{H}$ in terms of below k paths as

$$\mathcal{L}_{ij} = \max_{\sigma\in\Sigma^F(i,j,\mathcal{H})} W(\sigma) = \max_{\sigma\in\Sigma_j(i,j,\mathcal{H})} W(\sigma), \quad i \geq j.$$

Now define $\mathcal{W}(0),\ldots,\mathcal{W}(n) \in \mathbb{R}_{\max}^{n\times n}$ to be weight matrices with entries

$$w(k)_{ij} = \max_{\sigma\in\Sigma_k(i,j,\mathcal{H})} W(\sigma), \tag{5.1}$$

that is, $w(k)_{ij}$ is equal to the weight of the maximally weighted below k path in $G(\mathcal{H})$ from i to j and $\mathcal{L}_{ij} = w(j)_{ij}$. To compute these weights we set $\mathcal{W}(0) = \mathcal{H}$ and then compute the remaining weight matrices iteratively using the following result.

LEMMA 5.1. *Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be a Hungarian max-plus matrix. Then for $k = 1,\ldots,n-1$,*

$$w(k+1)_{ij} = \max\{w(k)_{ij}, w(k)_{i,k+1} + w(k)_{k+1,j}\}, \quad 1 \leq i,j \leq n. \tag{5.2}$$

Proof. Let $\sigma = \sigma(1),\ldots,\sigma(\ell)$ and suppose that $\sigma \in \Sigma_{k+1}(i,j,\mathcal{H})$. If $\ell = 2$ then $\sigma \in \Sigma_0(i,j,\mathcal{H}) \subseteq \Sigma_k(i,j,\mathcal{H})$. If $\ell > 2$ then consider $s = \max_{t=2}^{\ell-1}\sigma(t)$. If $s \leq k$ then $\sigma \in \Sigma_k(i,j,\mathcal{H})$, otherwise if $s = k+1$ then σ can be broken down into two

paths $\sigma_1 = \sigma(1), \ldots, \sigma(r)$ from i to $k+1$ and $\sigma_2 = \sigma(r), \ldots, \sigma(\ell)$ from $k+1$ to j, where $r = \arg\max_{t=2}^{\ell-1} \sigma(t)$ is unique since σ is an acyclic path. Clearly we have $W(\sigma) = W(\sigma_1) + W(\sigma_2)$ and since the only possible intermediate vertices on σ_1 and σ_2 have index less than $k+1$ we also have $\sigma_1 \in \Sigma_k(i, k+1, \mathcal{H})$ and $\sigma_2 \in \Sigma_k(k+1, j, \mathcal{H})$. Therefore every path in $\Sigma_{k+1}(i, j, \mathcal{H})$ is either a path in $\Sigma_k(i, j, \mathcal{H})$ or it corresponds to a pair of paths one from each of $\Sigma_k(i, k+1, \mathcal{H})$ and $\Sigma_k(k+1, j, \mathcal{H})$ so that

$$w(k+1)_{ij} \leq \max\{w(k)_{ij}, w(k)_{i,k+1} + w(k)_{k+1,j}\}, \quad 1 \leq i, j \leq n.$$

To prove the reverse inequality note that $\Sigma_k(i, j, \mathcal{H}) \subseteq \Sigma_{k+1}(i, j, \mathcal{H})$ and that any pair of paths $\sigma_1 \in \Sigma_k(i, k+1, \mathcal{H})$ and $\sigma_2 \in \Sigma_k(k+1, j, \mathcal{H})$ can concatenated to give a new path $\sigma = (\sigma_1, \sigma_2) \in \Sigma_{k+1}(i, j, \mathcal{H})$. □

Because we only ever need to work with the most recently computed set of k-path weights, it suffices to store them in a single matrix $\mathcal{W}$. This matrix $\mathcal{W}$ plays the role of the intermediate upper factor in classical Gaussian elimination.

Algorithm 3 Given the valuation $\mathcal{H} \in \mathbb{R}_{\max}^{n \times n}$ of a real symmetric positive-definite Hungarian matrix, this algorithm returns its max-plus Cholesky factor.

1: set $\mathcal{W} = \mathcal{H}$
2: **for** $k = 1, \ldots, n$ **do**
3: set $\mathcal{L}(k\colon n, k) = \mathcal{W}(k\colon n, k)$
4: **for** $i, j > k$ such that $w_{ik} \neq -\infty$ and $w_{jk} \neq -\infty$ **do**
5: set $w_{ij} = \max\{w_{ij}, w_{ik} + w_{jk}\}$
6: **end for**
7: **end for**

EXAMPLE 5.2. *We apply Algorithm 3 to*

$$\mathcal{H} = \begin{bmatrix} 0 & -1 & -1 & -\infty & -\infty \\ -1 & 0 & -3 & -\infty & -1 \\ -1 & -3 & 0 & -2 & -\infty \\ -\infty & -\infty & -2 & 0 & -6 \\ -\infty & -1 & -\infty & -6 & 0 \end{bmatrix}.$$

and record the matrix $\mathcal{W}$ at line 7 for each value of k until the algorithm terminates. This yields

$$\mathcal{W}(1) = \begin{bmatrix} 0 & -1 & -1 & -\infty & -\infty \\ -1 & 0 & -2 & -\infty & -1 \\ -1 & -2 & 0 & -2 & -\infty \\ -\infty & -\infty & -2 & 0 & -6 \\ -\infty & -1 & -\infty & -6 & 0 \end{bmatrix}, \quad \mathcal{W}(2) = \begin{bmatrix} 0 & -1 & -1 & -\infty & -\infty \\ -1 & 0 & -2 & -\infty & -1 \\ -1 & -2 & 0 & -2 & -3 \\ -\infty & -\infty & -2 & 0 & -6 \\ -\infty & -1 & -3 & -6 & 0 \end{bmatrix},$$

$$\mathcal{W}(3) = \begin{bmatrix} 0 & -1 & -1 & -\infty & -\infty \\ -1 & 0 & -2 & -\infty & -1 \\ -1 & -2 & 0 & -2 & -3 \\ -\infty & -\infty & -2 & 0 & -5 \\ -\infty & -1 & -3 & -5 & 0 \end{bmatrix}, \qquad \mathcal{L} = \begin{bmatrix} 0 & \infty & \infty & -\infty & -\infty \\ -1 & 0 & \infty & \infty & -\infty \\ -1 & -2 & 0 & \infty & -\infty \\ -\infty & -\infty & -2 & 0 & -\infty \\ -\infty & -1 & -3 & -5 & 0 \end{bmatrix}.$$

Let us focus on the $(5, 3)$ entry of the matrix $\mathcal{L}$. Looking at Figure 5.1, we see that there are three paths through $G(\mathcal{H})$ from vertex 5 to vertex 3,

$$\sigma_1 = (5, 2, 3), \quad \sigma_2 = (5, 2, 1, 3), \quad \sigma_3 = (5, 4, 3).$$

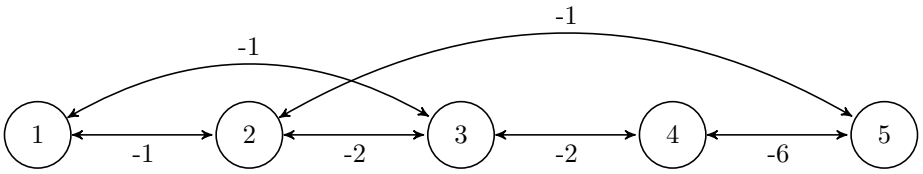


FIG. 5.1. *Precedence graph $G(\mathcal{H})$ for the matrix of Example 5.2.*

None of these paths is a below 1 *path but σ_1 and σ_2 are below* 2 *paths and σ_3 is a below* 4 *path. Since $W(\sigma_1) = W(\sigma_2) = -3$ and $W(\sigma_3) = -8$, the weight of the maximally weighted below k path from* 5 *to* 3 *is equal to -3 for $k \geq 2$ and $-\infty$ for $k < 2$.*

On the face of it, Algorithm 3 has no performance advantage over taking the exact Cholesky factorization of the matrix $H \in \mathbb{R}^{n\times n}$. Its worst case cost is $O(n^3)$.

It is possible to speed up Algorithm 3 by using a drop tolerance to limit the fill in, but again this is no different to classical Gaussian elimination applied to H. However, unlike conventional Gaussian elimination, we can use Algorithm 1 to compute a specially chosen subset of the columns $\mathcal{L}$ and then use these columns to seed several independent instances of Algorithm 3 to compute the remaining columns in a technique that we call *cutting in*.

A k-cut is a cut through $G(\mathcal{H})$ that separates the vertices $\{1,\ldots,k\}$ from $\{k+1,\ldots,n\}$. Define the *k-cut seed set* by

$$C = \big\{ i \in \mathbb{N} \ : \ \min\{c : \mathcal{h}_{cj} \neq -\infty \text{ for some } j > k\} \leq i \leq k \big\}.$$

Thus C is a contiguous set of vertices containing all of the vertices in $\{1,\ldots,k\}$ that are connected directly to a vertex in $\{k+1,\ldots,n\}$. Now if we use Algorithm 1 to compute the columns of $\mathcal{L}$ that correspond to vertices in C, that is we compute $\{\mathcal{L}_{\cdot,c} : c \in C\}$, then we can compute $w(k)_{ij}$ (as defined in (5.1)) for all $i,j > k$ from these columns using the following result. Once we have computed $w(k)_{ij}$ for $i,j > k$, we can use this data to initialize an instance of Algorithm 3 at step k.

LEMMA 5.3. *Let $\mathcal{H} \in \mathbb{R}^{n\times n}_{\max}$ be a Hungarian max-plus matrix and C be the k-cut seed set then*

$$\mathcal{w}(k)_{ij} = \max\big\{ \mathcal{h}_{ij}, \max_{c\in C}\{\mathcal{l}_{ic} + \mathcal{l}_{jc}\}\big\}, \quad i,j > k, \tag{5.3}$$

with $\mathcal{w}(k)_{ij}$ as in (5.1).

Proof. Let $\sigma = \sigma(1),\ldots,\sigma(\ell)$ and suppose that $\sigma \in \Sigma_k(i,j,\mathcal{H})$. If $\ell = 2$ then $W(\sigma) = \mathcal{h}_{ij}$. Otherwise if $\ell > 2$ then consider $s = \max_{t=2}^{\ell-1} \sigma(t)$. By construction the seed set must contain s. Now σ can be broken down into two paths $\sigma_1 = \sigma(1),\ldots,\sigma(r)$ from i to s and $\sigma_2 = \sigma(r),\ldots,\sigma(\ell)$ from s to j, where $r = \arg\max_{t=2}^{\ell-1} \sigma(t)$ is unique since σ is an acyclic path. Clearly, we have $W(\sigma) = W(\sigma_1) + W(\sigma_2)$ and since c is the intermediate vertex of maximum index visited by σ we have $\sigma_1 \in \Sigma^F(i,c,\mathcal{H})$ and $\sigma_2 \in \Sigma^F(c,j,\mathcal{H})$. Therefore,

$$\mathcal{w}(k)_{ij} \leq \max\big\{ \mathcal{h}_{ij}, \max_{c\in C}\{\mathcal{l}_{ic} + \mathcal{l}_{jc}\}\big\}, \quad i,j > k.$$

To prove the reverse inequality note that $\Sigma_0(i,j,\mathcal{H}) \subseteq \Sigma_k(i,j,\mathcal{H})$ and that for any $c \in C$, any pair of paths $\sigma_1 \in \Sigma_k(i,c,\mathcal{H})$ and $\sigma_2 \in \Sigma_k(c,j,\mathcal{H})$ can be concatenated to give a new path $\sigma = (\sigma_1,\sigma_2) \in \Sigma_k(i,j,\mathcal{H})$. ∎

EXAMPLE 5.4. *We return to the matrix of Example 5.2. We can cut in at* $k = 3$. *The seed set is given by* $C = \{2, 3\}$. *Using Algorithm 1 we compute*

$$\mathcal{L}_{\cdot,2} = [-\infty, 0, -2, -\infty, -1]^T, \quad \mathcal{L}_{\cdot,3} = [-\infty, -\infty, 0, -2, -3]^T,$$

which we substitute into (5.3) *to obtain*

$$\mathcal{W}(k)_{([4,5],[4,5])} = \max\left\{ \begin{bmatrix} 0 & -6 \\ -6 & 0 \end{bmatrix}, \begin{bmatrix} -\infty & -\infty \\ -\infty & -2 \end{bmatrix}, \begin{bmatrix} -4 & -5 \\ -5 & -6 \end{bmatrix} \right\} = \begin{bmatrix} 0 & -5 \\ -5 & 0 \end{bmatrix},$$

which agrees with the calculation in Example 5.2. We can then apply Algorithm 3 to compute columns 4 *and* 5 *of* $\mathcal{L}$ *from this small matrix.*

The cost of computing (5.3) will be $O(\sum_{c\in C} n(c)^2)$, where $n(c)$ is the number of finite entries in the cth column of $\mathcal{L}$. As before, this technique is compatible with using a drop tolerance or fixing a maximum number of entries per column so that this cost should be around $O(tm^2)$, where m is the maximum allowed number of finite entries per column and t is the number of vertices in the seed set. Thus if we are able to find several cuts through $G(\mathcal{H})$, each of which has a small seed set, then this hybrid approach has the potential to be fast. Finding cuts through graphs associated with finite element problems should be a good case for this approach as cuts though the matrix naturally correspond to partitions of the mesh.

6. Concluding remarks. We have described a novel approach for computing the sparsity pattern of an IC preconditioner, which makes use of max-plus algebra. Our numerical results demonstrate that this approach is able to produce effective preconditioners for use with the PCG method. It is outside the scope of the present study to develop an efficient implementation and more work is needed to obtain a high-quality efficient parallel implementation that, in terms of the total solution time, can compete with simpler established IC preconditioners. The max-plus IC problem has some nice features that might lead one to think that this is possible. Importantly, each column can be computed independently using Algorithm 1 and our alternative algorithm based on Gaussian elimination that was discussed in Section 5 can also exploit some parallelism. Having computed the max-plus preconditioner sparsity pattern, the parallel approach of Chow and Patel [3] can be used to (approximately) perform the incomplete factorization.

Finally, we note that the Factorized Sparse Approximate Inverse (FSAI) preconditioner that was introduced more than 20 years by Kolotilina and Yeremin [23] requires a pattern for the nonzero entries of the factors. Originally, this had to be be set statically by the user (typically using small powers of A) although, more recently, dynamic schemes have been proposed (see, for example, [11, 22] for further details and references). Theory would suggest that the positions of the largest entries of L^{-1} would be the ideal choice for the sparsity pattern; it remains an open question whether we can use max-plus algebra in this case.

REFERENCES

[1] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. on Scientific Computing*, 22(4):1333–1353, 2000.

[2] D. A. Bini and V. Noferini. Solving polynomial eigenvalue problems by means of the Ehrlich-Aberth method. *Linear Algebra and its Applications*, 439(4):1130–1149, 2013.

[3] E. Chow and A. Patel. Fine-grained parallel incomplete LU factorization. *SIAM J. on Scientific Computing*, 37(2):C169–C193, 2015.

[4] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):Article 1, 2011.

[5] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[7] I. S. Duff and J. R. Gilbert. Maximum-weighted matching and block pivoting for symmetric indefinite systems. in Abstract book of Householder Symposium XV, June 17-21, 2002, pp.73–75.

[8] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. on Matrix Analysis and Applications*, 22(4):973–996, 2001.

[9] I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. on Matrix Analysis and Applications*, 27:313–340, 2005.

[10] L. Hogben (editor). *Handbook of Linear Algebra, Second Edition*. CRC Press, Taylor & Francis Group, 2014.

[11] FSAIPACK. User's guide, 2013. `http://www.dmsa.unipd.it/~janna/FSAIPACK/FSAIPACK_UG.pdf`.

[12] F. R. Gantmacher. *The Theory of Matrices*, volume one. Chelsea, New York, 1959.

[13] S. Gaubert and M. Sharify. Tropical scaling of polynomial matrices. In *Positive systems*, volume 389 of *Lecture Notes in Control and Information Sciences*, pages 291–303. Springer-Verlag, Berlin, 2009.

[14] A. Gupta and L. Ying. On algorithms for finding maximum matchings in bipartite graphs. Technical Report RC 21576, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1999.

[15] M. Hagemann and O. Schenk. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM J. on Scientific Computing*, 28(2):403–420, 2006.

[16] S. Hammarling, C. J. Munro, and F. Tisseur. An algorithm for the complete solution of quadratic eigenvalue problems. *ACM Transactions on Mathematical Software*, 39(3):18:1–18:19, April 2013.

[17] J. D. Hogg and J. A. Scott. Optimal weighted matchings for rank-deficient sparse matrices. *SIAM J. on Matrix Analysis and Applications*, 34:1431–1447, 2013. DOI 10.1137/120884262.

[18] J. D. Hogg and J. A. Scott. Pivoting strategies for tough sparse indefinite systems. *ACM Transactions on Mathematical Software*, 40(1):Article 4, 2013.

[19] J. Hook and F. Tisseur. Incomplete LU preconditioner based on max-plus approximation of LU factorization. MIMS Eprint 2016.47, School of Mathematics, The University of Manchester, 2016.

[20] HSL. A collection of Fortran codes for large-scale scientific computation, 2016. `http://www.hsl.rl.ac.uk/`.

[21] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. on Scientific Computing*, 22:2194–2215, 2001.

[22] C. Janna, M. Ferronato, F. Sartoretto, and G. Gambolati. FSAIPACK: a software package for high performance factored sparse approximate inverse preconditioning. *ACM Transactions on Mathematical Software*, 41(2):Article 10, 2015.

[23] L. Y. Kolotilina and A. Y. Yeremin. Factorized sparse approximate inverse preconditionings. I. Theory. *SIAM J. on Matrix Analysis and Applications*, 14(1):45–58, 1993.

[24] X. S. Li and J. W. Demmel. Making sparse Gaussian elimination scalable by static pivoting. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pages 1–17. IEEE Computer Society, 1998.

[25] C.-J. Lin and J. J. Moré. Incomplete Cholesky factorizations with limited memory. *SIAM J. on Scientific Computing*, 21(1):24–45, 1999.

[26] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34:473–497, 1980.

[27] V. Noferini, M. Sharify, and F. Tisseur. Tropical roots as approximations to eigenvalues of matrix polynomials. *SIAM J. on Matrix Analysis and Applications*, 36(1):138–157, 2015.

[28] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra and its Applications*, 240:131–151, 1996.

[29] J. K. Reid and J. A. Scott. Ordering symmetric sparse matrices for small profile and wavefront. *International J. of Numerical Methods in Engineering*, 45:1737–1755, 1999.

[30] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Co., Boston, 1996.

[31] O. Schenk, S. Röllin, and A. Gupta. The Effects of Unsymmetric Matrix Permutations and Scalings in Semiconductor Device and Circuit Simulation. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, 23(3):400–411, 2004.

[32] O. Schenk, A. Wächter, and M. Hagemann. Matching-based preprocessing algorithms to the solution of saddle-point problems in saddle-point problems in large-scale non-convex interior-point optimization. *Computational Optimization and Applications*, 36:321–341, 2007.

[33] J. A. Scott and M. Tůma. The importance of structure in incomplete factorization preconditioners. *BIT Numerical Mathematics*, 51:385–404, 2011.

[34] J. A. Scott and M. Tůma. `HSL_MI28`: an efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Transactions on Mathematical Software*, 40(4):Article 24, 2014.

[35] J. A. Scott and M. Tůma. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM J. on Scientific Computing*, 36(2):A609–A633, 2014.

[36] S. W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *International J. of Numerical Methods in Engineering*, 23:239–251, 1986.

[37] S. W. Sloan. A Fortran program for profile and wavefront reduction. *International J. of Numerical Methods in Engineering*, 28:2651–2679, 1989.