SIAM J. SCI. COMPUT.
Vol. 40, No. 4, pp. A1987–A2004



# A MAX-PLUS APPROACH TO INCOMPLETE CHOLESKY FACTORIZATION PRECONDITIONERS*

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

**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, *SIAM J. Matrix Anal. Appl.*, 38 (2017), pp. 1160–1189] but is applied to symmetric positive-definite matrices, which comprise an important special case for the method and its application. An 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

**DOI.** 10.1137/16M1107735



## 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, [34] 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 [31] 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 [20] 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 analogue of linear algebra developed

*Submitted to the journal's Methods and Algorithms for Scientific Computing section December 13, 2016; accepted for publication (in revised form) April 16, 2018; published electronically July 3, 2018.

http://www.siam.org/journals/sisc/40-4/M110773.html

**Funding:** The second author's work was supported by EPSRC grant EP/M025179/1. The third author's work was supported by EPSRC grant EP/I005293 and by a Royal Society-Wolfson Research Merit Award.

†Institute for Mathematical Innovation, University of Bath, Claverton Down, Bath, BA2 7AY, UK (j.l.hook@bath.ac.uk.)

‡STFC Rutherford Appleton Laboratory, Harwell Campus, Oxfordshire, OX11 0QX, UK (jennifer.scott@stfc.ac.uk, Jonathan.Hogg@stfc.ac.uk).

§School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (Francoise.Tisseur@manchester.ac.uk).

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 [16, Chap. 35]. In the past few years, max-plus algebra has been used to examine a number of numerical linear algebra problems [2, 12, 15, 28]. While the numerical experiments reported on in [20] 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 can outperform the 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 [27] 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 [35, 36], 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 made between our new max-plus IC preconditioner and both level-based and memory-based IC preconditioners used with the preconditioned conjugate gradient method. Finally, some concluding remarks and possible future directions are given in section 5. 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, $\mathcal{A} = (\mathcal{a}_{ij}) \in \mathbb{R}_{\max}^{n\times n}$, where $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$.

**2. Max-plus approximation of LU factorization.** In [20], 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

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

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 $\boldsymbol{a}_{ij} = \log|a_{ij}|$ in $\mathbb{R}_{\max}$. The max-plus matrix $\mathcal{V}(A)$ is termed the *valuation* of $A$.

For $\mathcal{A} \in \mathbb{R}_{\max}^{n\times n}$, define the max-plus permanent

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

where $\Pi(n)$ is the set of all permutations on $\{1, \ldots, n\}$. In [20], Hook and Tisseur make repeated use of the heuristic that $\mathcal{V}\big(\det(A)\big) \approx \operatorname{perm}\big(\mathcal{V}(A)\big)$. This approximation can be intuitively justified by expressing the determinant as a sum of terms coming from each permutation and the permanent as the maximum of those terms. If the matrix $A$ is sparse and has a wide range of entry sizes, then the heuristic approximation should be more accurate. It is well known that the entries in the lower triangle of $L$ and the upper triangle of $U$ of an LU factorization of $A \in \mathbb{R}^{n\times n}$ can be expressed explicitly in terms of determinants of submatrices $A$ (see [11, p. 35]) by

$$l_{ik} = \det\big(A([1\colon k-1, i], 1\colon k)\big) / \det\big(A(1\colon k, 1\colon k)\big), \quad i \geq k, \tag{2.3}$$

$$u_{kj} = \det\big(A(1\colon k, [1\colon k-1, j])\big) / \det\big(A(1\colon k-1, 1\colon k-1)\big), \quad j \geq k. \tag{2.4}$$

Here the notation $[1\colon k-1, i]$ means the indices $1, 2, \ldots, k-1, i$. If both the numerator and denominator in either (2.3) or (2.4) are zero, then the convention $0/0 = 0$ is used. Using this fact and heuristic (2.2), Hook and Tisseur [20] 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

$$\boldsymbol{\ell}_{ik} = \operatorname{perm}\big(\mathcal{A}([1\colon k-1, i], 1\colon k)\big) - \operatorname{perm}\big(\mathcal{A}(1\colon k, 1\colon k)\big), \quad i \geq k, \tag{2.5}$$

$$\boldsymbol{u}_{kj} = \operatorname{perm}\big(\mathcal{A}(1\colon k, [1\colon k-1, j])\big) - \operatorname{perm}\big(\mathcal{A}(1\colon k-1, 1\colon k-1)\big), \quad j \geq k, \tag{2.6}$$

and $\boldsymbol{\ell}_{ik} = \boldsymbol{u}_{kj} = -\infty$ if $i, j < k$. If the two terms on the right-hand side of either (2.5) or (2.6) 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

$$\mathcal{V}(L) \approx \mathcal{L}, \qquad \mathcal{V}(U) \approx \mathcal{U}, \tag{2.7}$$

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

*Example* 2.1. Consider

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

We compute the max-plus LU factors of $\mathcal{V}(A)$ using (2.5) and (2.6). For such a small example, we can evaluate the permanent of a matrix or submatrix by simply evaluating all possible permutations and recording the maximum. Note that the permanent can also be expanded along a row or a column in a similar way to the Leibniz formula for determinants but with no sign function, and the sum replaced by a maximum and the product by a sum (see [19, Prop. 1.2]). For example,

$$\begin{aligned} u_{33} &= \operatorname{perm}\big(\mathcal{A}([1\colon 3],[1\colon 3])\big) - \operatorname{perm}\big(\mathcal{A}(1\colon 2, 1\colon 2)\big) \\ &= \max\big(1+\max(-2,0), 3+\max(1,3)\big) - \max(1,-\infty) = 5. \end{aligned}$$

We obtain

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

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

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

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

**2.1. Hungarian matrices.** A matrix $H \in \mathbb{R}^{n\times n}$ is said to be *Hungarian* if its entries satisfy $|h_{ij}| \le 1$ and $|h_{ii}| = 1$ for all $i, j = 1, \dots, n$. 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 [29] in the mid 1990s. They proposed using the solution to an optimal assignment problem as 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 [13]). The idea was subsequently extended to symmetric systems [7, 9] and over the last 15 years or so, it has been adopted by the sparse linear algebra community for both nonsymmetric and symmetric problems (see, for example, [1, 14, 17, 18, 25, 32, 33]).

A max-plus matrix $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ is said to be *Hungarian* if its entries satisfy $\hbar_{ij} \le 0$ and $\hbar_{ii} = 0$ for all $i, j = 1, \dots, n$. Note that $H \in \mathbb{R}^{n\times n}$ is Hungarian if and only if $\mathcal{V}(H)$ is Hungarian.

PROPOSITION 2.2. *Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be Hungarian; then $\mathcal{H}$ admits max-plus LU factors.*

*Proof.* From (2.5) and (2.6) we have that $\operatorname{perm}\big(\mathcal{H}(1\colon k, 1\colon k)\big) \ne -\infty$ for all $k$ is a sufficient condition for $\mathcal{H}$ to admit max-plus LU factors. But since $\hbar_{ij} \le 0$ and $\hbar_{ii} = 0$ for all $i, j$, we have $\operatorname{perm}\big(\mathcal{H}(1\colon k, 1\colon k)\big) = 0$ for all $k$. ∎

**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 [29].

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.*

Thus given a symmetric positive-definite matrix $A \in \mathbb{R}^{n\times n}$, we can easily scale it to obtain a symmetric positive-definite Hungarian matrix $H \in \mathbb{R}^{n\times n}$. Since the Hungarian property is very beneficial to our max-plus IC algorithm, in the remainder of the paper we assume that the matrix $A$ has been scaled so that the scaled matrix $H = DAD$ is symmetric positive-definite Hungarian.

PROPOSITION 3.2. *Let $H \in \mathbb{R}^{n\times n}$ be a symmetric positive-definite Hungarian matrix, and let $\mathcal{H} = \mathcal{V}(H) \in \mathbb{R}_{\max}^{n\times n}$. Then the max-plus LU factors $\mathcal{L}$ and $\mathcal{U}$ of $\mathcal{H}$ satisfy $\mathcal{U} = \mathcal{L}^T$ and*

$$\ell_{ik} = \begin{cases} \operatorname{perm}\big(\mathcal{H}([1\colon k-1, i], 1\colon k)\big), & i \geq k, \\ -\infty, & i < k. \end{cases} \tag{3.1}$$

*Proof.* It follows from (2.5)–(2.6) and the proof of Proposition 2.2 that $\ell_{ik} = \operatorname{perm}\big(\mathcal{H}([1\colon k-1, i], 1\colon k)\big)$ for $i \geq k$, and $u_{ki} = \operatorname{perm}\big(\mathcal{H}(1\colon k, [1\colon k-1, i])\big)$ for $i \geq k$. Since $\mathcal{H}$ is symmetric, $\ell_{ik} = u_{ki}$ for all $i$ and $k$, i.e., $\mathcal{U} = \mathcal{L}^T$. ∎

We say that $\mathcal{L}$ as defined in (3.1) is the *max-plus lower Cholesky factor* of $\mathcal{H}$. In the special case of a symmetric positive-definite Hungarian matrix, the heuristic order of magnitude approximation in (2.7) can be rewritten as follows.

HEURISTIC 3.3 (max-plus approximation of Cholesky factors). *Let $H \in \mathbb{R}^{n\times n}$ be a symmetric positive-definite Hungarian matrix, let $L \in \mathbb{R}^{n\times n}$ be the lower Cholesky factor of $H$, and let $\mathcal{H} = \mathcal{V}(H)$. Then*

$$\mathcal{V}(L) \approx \mathcal{L},$$

*where $\mathcal{L}$ is the max-plus lower Cholesky factor of $\mathcal{H}$.*

**3.1. Precedence graphs and fill paths.** Hook and Tisseur [20] present several algorithms for computing max-plus LU factors for nonsymmetric matrices. Their algorithms all work on a directed bipartite graph, with a pair of vertices for each row in the matrix. In the special case of a symmetric Hungarian matrix, we are able to simplify their algorithm so that it works on an undirected graph with a single vertex for each row.

Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be a symmetric Hungarian matrix, and let $G(\mathcal{H})$ be the *precedence* graph of $\mathcal{H}$, that is, the graph with vertices $\{1, \ldots, n\}$ and an undirected edge $e : i \leftrightarrow j$ of weight $h_{ij}$ whenever $h_{ij} \neq -\infty$. A *path $\sigma$ of length $\ell$* from $i$ to $j$ in $G(\mathcal{H})$ is a sequence of $\ell + 1$ distinct vertices $\sigma = \big(\sigma(1), \sigma(2), \ldots, \sigma(\ell), \sigma(\ell+1)\big)$ such that $\sigma(1) = i$, $\sigma = (\ell + 1) = j$ and $h_{\sigma(k),\sigma(k+1)} \neq -\infty$ 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 $\sigma$ of length $\ell$ from $i$ to $j$ is a *fill path* if $\sigma(1)=i$, $\sigma(k)<\min\{i,j\}$ for $k=2,\dots,\ell$ 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, $\Sigma^F(i,j,\mathcal{H})\subseteq\Sigma(i,j,\mathcal{H})$.

THEOREM 3.4. *Let $H\in\mathbb{R}^{n\times n}$ be a symmetric positive-definite Hungarian matrix, and let $\mathcal{H}=\mathcal{V}(H)$. Then the max-plus Cholesky lower factor $\mathcal{L}$ of $\mathcal{H}$ is given by*

$$\mathcal{l}_{ik}=\begin{cases}\max_{\sigma\in\Sigma^F(i,k,\mathcal{H})} W(\sigma), & i\geq k,\\ -\infty, & i<k.\end{cases} \tag{3.2}$$

*Proof.* From Proposition 3.2 and (2.2) we have that for $i\geq k$,

$$\mathcal{l}_{ik}=\operatorname{perm}\big(\mathcal{H}([1\colon k-1,i],1\colon k)\big)=\max_{\phi\in\Phi}\sum_{j\in\{1,2,\dots,k-1,i\}}\mathcal{h}_{j\phi(j)}, \tag{3.3}$$

where $\Phi$ denotes the set of bijections from $\{1,2,\dots,k-1,i\}$ to $\{1,2,\dots,k\}$.

If $i=k$, then $\mathcal{l}_{kk}=\operatorname{perm}\big(\mathcal{H}(1\colon k,1\colon k)\big)=0$ and $\max_{\sigma\in\Sigma^F(k,k,\mathcal{H})}W(\sigma)=0$ since we allow paths of zero length that consist of a single vertex and have zero weight. So (3.3) holds when $i=k$.

Assume now that $i>k$. Define the map $\ell:\Phi\mapsto\mathbb{N}$ by $\phi^{\ell(\phi)}(i)=k$, where $\phi^t(i)$ means that the map $\phi$ is applied $t$ times to the vertex $i$. Also define the map $P:\Phi\mapsto\Sigma^F(i,k,\mathcal{H})$ by $P(\phi)=\sigma$, where $\sigma=\big(i,\phi(i),\phi^2(i),\dots,\phi^{\ell(\phi)}(i)\big)$. To see that the maps $\ell$ and $P$ are well defined, consider the sequence $\sigma=\big(i,\phi(i),\phi^2(i),\dots\big)$. By injectivity of $\phi$ and the fact that $i$ has no $\phi$ preimage when $i>k$, the sequence $\sigma=P(\phi)$ can never repeat itself and must therefore terminate at vertex $k$ after at most $k$ steps.

We show that the map $P$ is a surjection, i.e., every fill path $\sigma\in\Sigma^F(i,k,\mathcal{H})$ has a preimage $Q(\sigma)\in\Phi$. Indeed, let $\sigma=\big(i,\sigma(2),\dots,\sigma(\ell-1),k\big)$, and set $Q(\sigma)=\phi$ with, for $j\in\{1,2,\dots,k-1,i\}$,

$$\phi(j)=\begin{cases}\sigma(t+1) & \text{if } j=\sigma(t) \text{ for some } t\in\{1,\dots,\ell\},\\ j & \text{otherwise.}\end{cases}$$

Then $P\big(Q(\sigma)\big)=\sigma$ by construction.

Next, we show that for any $\phi\in\Phi$, the bijection $\psi=Q\big(P(\phi)\big)\in\Phi$ satisfies

$$\sum_{j\in\{1,2,\dots,k-1,i\}}\mathcal{h}_{j\phi(j)}\leq\sum_{j\in\{1,2,\dots,k-1,i\}}\mathcal{h}_{j\psi(j)}=W\big(P(\phi)\big).$$

To see this, let $V\subseteq\{1,\dots,k-1,i\}$ be the set of vertices in $G(\mathcal{H})$ visited by the sequence $\sigma=P(\phi)$, and let $U=\{1,\dots,k-1,i\}\setminus V$ be the complement of $V$. Then

$$\sum_{j\in\{1,2,\dots,k-1,i\}}\mathcal{h}_{j\phi(j)}=\sum_{j\in V}\mathcal{h}_{j\phi(j)}+\sum_{j\in U}\mathcal{h}_{j\phi(j)}\leq\sum_{j\in V}\mathcal{h}_{j\phi(j)}$$

since $\mathcal{h}_{ij}\leq 0$ for all $i,j$. Now for $\psi$ we have

$$\sum_{j\in\{1,2,\dots,k-1,i\}}\mathcal{h}_{j\psi(j)}=\sum_{j\in V}\mathcal{h}_{j\phi(j)}+\sum_{j\in U}\mathcal{h}_{j\psi(j)}=\sum_{j\in V}\mathcal{h}_{j\phi(j)}$$

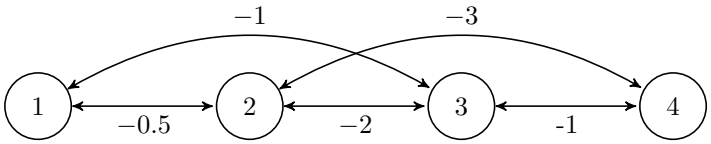


FIG. 3.1. *Precedence graph $G(\mathcal{H})$ for the matrix of Example* 3.5.

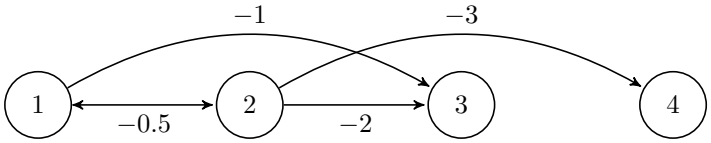


FIG. 3.2. *Precedence graph $G\big(\mathcal{H}(2)\big)$ for the matrix of Example* 3.5.

since $\mathcal{H}_{ii} = 0$ for all $i$. Hence

$$\sum_{j\in\{1,2,\ldots,k-1,i\}} \mathcal{H}_{j\psi(j)} = \sum_{j\in V} \mathcal{H}_{j\phi(j)} = \sum_{t=1}^{\ell(\phi)} w\big(\sigma(t),\sigma(t+1)\big) = W(\sigma). \tag{3.4}$$

Finally, the permanent in (3.3) is given by the maximum weight of a bijection $\phi \in \Phi$. From the construction of $P : \Phi \mapsto \Sigma^F(i,k,\mathcal{H})$, we know that each of these bijections can be mapped to a fill path $P(\phi)$ from $k$ to $i$. Moreover, we know that for every bijection $\phi \in \Phi$, there is a bijection $\psi = Q\big(P(\phi)\big) \in \Phi$ whose weight, greater than or equal to the weight of $\phi$, is equal to the weight of the fill path $P(\phi)$. Therefore, since $P$ is a surjection, the maximum weight of a bijection is equal to the maximum weight of a fill path.

$$\mathcal{L}_{ik} = \max_{\phi\in\Phi} \sum_{j\in\{1,2,\ldots,k-1,i\}} \mathcal{H}_{j\phi(j)} = \max_{\psi\in Q(P(\Phi))} \sum_{j\in\{1,2,\ldots,k-1,i\}} \mathcal{H}_{j\psi(j)} = \max_{\sigma\in\Sigma^F(i,k,\mathcal{H})} W(\sigma). \qquad \square$$

*Example* 3.5. 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 second column of $\mathcal{L}$. From (3.2) we have $\mathcal{L}_{12} = -\infty$, $\mathcal{L}_{22} = 0$, $\mathcal{L}_{32} = \max_{\sigma\in\Sigma^F(3,2,\mathcal{H})} W(\sigma)$, where $\Sigma^F(3,2,\mathcal{H}) = \{(3,2),(3,1,2)\}$. Since $W(3,2) = -2$ and $W(3,1,2) = -1.5$, we have $\mathcal{L}_{32} = -1.5$. Similarly, $\mathcal{L}_{42} = \max_{\sigma\in\Sigma^F(4,2,\mathcal{H})} W(\sigma)$, where $\Sigma^F(4,2,\mathcal{H}) = \{(4,2)\}$, and since $W(4,2) = -3$, we have $\mathcal{L}_{42} = -3$. Note that $\Sigma(4,2,\mathcal{H})$ also contains the paths $(4,3,2)$ and $(4,3,1,2)$. Note also that $W(4,3,1,2) = -2.5$, which is greater than $W(4,2)$, but we do not count the weight of the path $(4,3,1,2)$ 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 3.5 and $k = 2$ in Figure 3.2.

LEMMA 3.6. *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 $\sigma$ 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, $\sigma$ traverses $\ell - 1$ edges between the vertices $\{1,\dots,k\}$ and 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 $\sigma$ 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 $\sigma$ 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 $\sigma$ 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 3.7. *Let $\mathcal{H} \in \mathbb{R}_{\max}^{n\times n}$ be symmetric and Hungarian, and let $\mathcal{L}$ be the Cholesky factor of $\mathcal{H}$. Then for $i \ge k$,*

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

To compute the $k$th 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 nonpositive, 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, while the operation *(index, priority) = pop_max(heap)* returns and removes the index-priority pair with maximum priority.

**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 $k$th column of the max-plus Cholesky factor $\mathcal{L}$ of $\mathcal{H}$.

set $d_i = -\infty$ and $checked_i = false$, $1 \le i \le n$,
initialize heap; push(heap, $k$, 0)
**while** heap is nonempty **do**
    $(i, d_i)$ = pop_max(heap)
    set $checked_i$ =*true*
    **if** $i \le k$ **then**
        **for all** $j$ such that $\mathcal{H}_{ij} \ne -\infty$ and $checked_j = false$ **do**
            $d_{cand} = d_i + \mathcal{H}_{ij}$
            **if** $d_{cand} > d_j$ **then**
                set $d_j = d_{cand}$
                push(heap, $j$, $d_{cand}$)
            **end if**
        **end for**
    **end if**
    **if** $i \ge k$ **then**
        $\mathcal{L}_{ik} = d_i$
    **end if**
**end while**

**3.2. 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 Heuristic 3.3 suggests using the pattern $S \in \{0,1\}^{n\times n}$ given by

$$s_{ij} = \begin{cases} 1 & \text{if } \mathcal{L}_{ij} \ge \log\epsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

The total number of nonzero entries per column can also be restricted to an integer $m$, by setting $s_{ij} = 1$ for the $m$ largest positions in the $j$th column only, as predicted by Heuristic 3.3. Once we have a suitable pattern matrix, we can compute an IC factor with that pattern using standard algorithms from the level of fill IC($k$) approach. Specifically, we compute one column of the preconditioner at a time using a left-looking algorithm; details are given in [34].

If we are computing the max-plus Cholesky factor of $\mathcal{H}$ 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 $k$th 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 around 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 (3.5) 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 (3.2), the max-plus IC pattern $S \in \{0,1\}^{n\times n}$ in (3.5) 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 levels of fill and including some larger entries with higher levels 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 $\epsilon = \gamma^{k+1}$. Note also that Scott and Tůma [34] explored IC factorizations with variable levels of fill, allowing large entries to contribute to more levels of fill than small entries. Their numerical results illustrated the potential effectiveness of such an approach.

**4. Numerical results.** We present numerical results for problems taken from the University of Florida 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 $\alpha$ is a nonnegative shift. The permutation matrix $Q$ is computed using the Sloan profile reduction ordering algorithm [30, 37, 38]. This ordering is used since, in our experience, it frequently leads to a reduction in the number of conjugate gradient iterations [35, 36]. Preconditioned conjugate gradient (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 $\alpha$ is as described in [35] (see also [26]). The max-plus IC factorization is started with $\alpha = 0$, but if a zero (or

**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 $k$th column of a pattern matrix with 0 and 1 entries such that there are 1's in the $r \le m$ entries corresponding to the largest entries in the $k$th column of the max-plus Cholesky factor $\mathcal{L}$ of $\mathcal{H}$ that are greater than $\log \epsilon$.

for $i = 1, \ldots, n$ set $d_i = -\infty$ and $checked_i = false$
initialize heap; push(heap, $k$, 0)
set $p_{ik} = 0$ for $i = 1, 2, \ldots, n$; set $r = 0$
**while** $r < m$ **do**
    ($i$, $d_i$) = pop_max(heap)
    **if** $d_i < \log \epsilon$ **then exit**
    set $checked_i = true$
    **if** $i \le k$ **then**
        **for all** $j$ such that $\mathcal{H}_{ij} \ne -\infty$ and $checked_j = false$ **do**
            $d_{cand} = d_i + \mathcal{H}_{ij}$
            **if** $d_{cand} > d_j$ **then**
                set $d_j = d_{cand}$
                push(heap, $j$, $d_{cand}$)
            **end if**
        **end for**
    **end if**
    **if** $i \ge k$ **then**
        $p_{ik} = 1$
        $r = r + 1$
    **end if**
**end while**

negative) pivot is encountered, a nonzero $\alpha$ is employed (the initial value used in our experiments is 0.001) and the factorization is restarted. This process may need to be repeated more than once, with $\alpha$ 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 [21]. 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 $i$th step if

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

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 vector 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 Intel MKL 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* is the number of iterations required for PCG to converge.

$ma_{pcg}$ is 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 forward and a backward 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 prescaling 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 postfactorization filtering step (so that all entries in the computed factor that are of absolute value less than $\delta$ 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 postfactorization filtering step.

**`HSL_MI28`:** A limited memory IC preconditioner developed by Scott and Tůma [35, 36]. 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 (that is, the `HSL_MA28` 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 $\alpha$ may be needed. A nonzero value is only used if, during the construction of the preconditioner, a zero or negative pivot is encountered. We apply postfactorization filtering to the IC(0) and IC(1) patterns so as to be consistent with `HSL_MI28`. In practice, we find that dropping small entries can improve the sparsity of the factors without significantly affecting the quality of the preconditioner. See, e.g., [34].

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 nontrivial and outside our current study. We observe that the patterns of the columns of IC(1) (and, more generally, IC($k$)) can also be computed in parallel [22], 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, $\epsilon$, the max-plus drop tolerance, and $\delta$, a tolerance that is applied to filter the final $L$ factor. To be consistent with the IC(0) and IC(1) preconditioners, the latter

TABLE 4.1
*Breakdown of timings for problem Janna/Bump_2911.*

| Time taken | Max-plus | IC(0) | IC(1) | MI28 |
|---|---|---|---|---|
| computing pattern | a | a | a | - |
| computing numeric values | a | a | a | - |
| total preconditioner set up | 260 | 14 | 30 | 55 |
| PCG solve | 58 | 90 | 95 | 35 |
| total | 318 | 104 | 125 | 90 |

is set to $10^{-3}$.

To establish suitable settings for $m$ and $\epsilon$, 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 4.2 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 4.3. As $m$ increases and $\epsilon$ decreases, more entries are included in the factors. The results show that typically the relaxation of $\epsilon$ 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 than 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 best 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—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 fraction 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

TABLE 4.2

*Results for various values of the max-plus parameters $m$ and $\epsilon$ 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 4.3

*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 |

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 three 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.

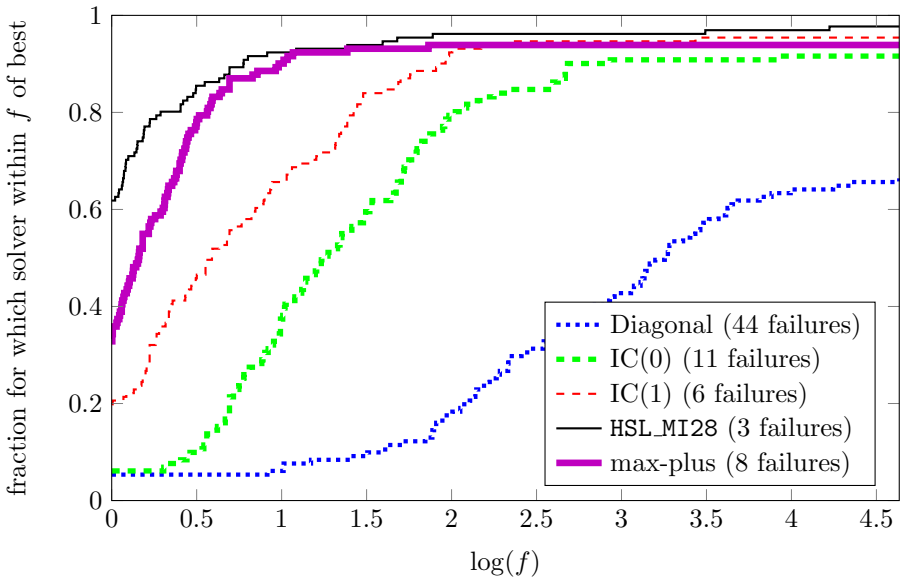


FIG. 4.1. *Performance profile comparing nitr across various preconditioners.*

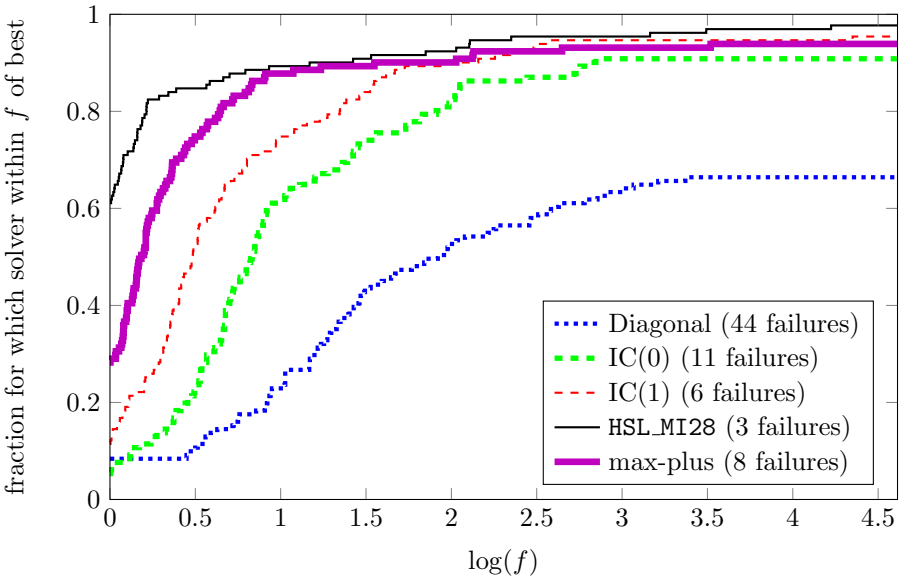


FIG. 4.2. *Performance profile comparing* $ma_{pcg}$ *in* (4.1) *across various preconditioners.*

**5. Concluding remarks.** We have described a novel approach to 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 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.

The process of computing the max-plus preconditioner sparsity pattern is remarkably similar to the process of computing a level of fill IC($k$) preconditioner sparsity pattern. The level of fill sparsity pattern is computed using an unweighted graph $G$, such that the level of fill of the $(i, j)$ entry in the Cholesky factor is equal to the length (counted in number of steps) of the shortest fill path through $G$ from $i$ to $j$. Just like in the max-plus case, the sparsity pattern for each column can be computed independently using a shortest path algorithm. The difference between the max-plus method and the level of fill method is that in the max-plus case the graph has weighted edges, and we seek the maximally weighted path rather than simply the path with the fewest steps. This extra flexibility allows the max-plus method to take into account the different-sized entries in the problem matrix and as a result produce better preconditioners. However, using a weighted graph means that we have to use Dijkstra's algorithm instead of a breadth first search (the algorithm typically used for the level of fill method). A breadth first search is generally a little faster than Dijkstra's algorithm as the data structure required to store the integer depths it requires is simpler. The exact costs of applying both methods depend strongly on the structure of the problem matrix and the parameters chosen for the preconditioner. For example, using smaller thresholds in the max-plus method allows Dijkstra's algorithm to be terminated early, which speeds up the computation of the preconditioner. If we can develop an implementation of Algorithm 1 that is not significantly slower than the symbolic phase of IC(1), then this should result in an overall faster method for solving sparse positive-definite linear systems.

Finally, we note that the Factorized Sparse Approximate Inverse (FSAI) preconditioner that was introduced more than 20 years by Kolotilina and Yeremin [24] requires a pattern for the nonzero entries of the factors. Originally, this had to be set statically by the user (typically using small powers of $A$), although, more recently, dynamic schemes have been proposed (see, for example, [10, 23] 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.

**Acknowledgment.** We would like to thank the reviewers for their constructive feedback.

REFERENCES

[1] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353, https://doi.org/10.1137/S1064827599361308.

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

[3] E. Chow and A. Patel, *Fine-grained parallel incomplete LU factorization*, SIAM J. Sci. Comput., 37 (2015), pp. C169–C193, https://doi.org/10.1137/140968896.

[4] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), 1.

[5] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[6] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.
[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, Peebles, Scotland, 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. Matrix Anal. Appl., 22 (2001), pp. 973–996, https://doi.org/10.1137/S0895479899358443.
[9] I. S. Duff and S. Pralet, *Strategies for scaling and pivoting for sparse symmetric indefinite problems*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 313–340, https://doi.org/10.1137/04061043X.
[10] FSAIPACK, *User's Guide*, 2013; available online at http://www.dmsa.unipd.it/~janna/FSAIPACK/FSAIPACK_UG.pdf.
[11] F. R. Gantmacher, *The Theory of Matrices, Vol.* 1, Chelsea, New York, 1959.
[12] S. Gaubert and M. Sharify, *Tropical scaling of polynomial matrices*, in Positive Systems, Lecture Notes in Control and Inform. Sci. 389, Springer-Verlag, Berlin, 2009, pp. 291–303.
[13] A. Gupta and L. Ying, *On Algorithms for Finding Maximum Matchings in Bipartite Graphs*, Tech. Report RC 21576, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1999.
[14] M. Hagemann and O. Schenk, *Weighted matchings for preconditioning symmetric indefinite linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 403–420, https://doi.org/10.1137/040615614.
[15] S. Hammarling, C. J. Munro, and F. Tisseur, *An algorithm for the complete solution of quadratic eigenvalue problems*, ACM Trans. Math. Software, 39 (2013), 18, https://doi.org/10.1145/2450153.2450156.
[16] L. Hogben, ed., *Handbook of Linear Algebra*, 2nd ed., CRC Press, Boca Raton, FL, 2014.
[17] J. D. Hogg and J. A. Scott, *Optimal weighted matchings for rank-deficient sparse matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1431–1447, https://doi.org/10.1137/120884262.
[18] J. D. Hogg and J. A. Scott, *Pivoting strategies for tough sparse indefinite systems*, ACM Trans. Math. Software, 40 (2013), 4.
[19] J. Hook, *Max-plus algebraic statistical leverage scores*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1410–1433, https://doi.org/10.1137/16M1097596.
[20] J. Hook and F. Tisseur, *Incomplete LU preconditioner based on max-plus approximation of LU factorization*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1160–1189, https://doi.org/10.1137/16M1094579.
[21] HSL Mathematical Software Library, *A Collection of Fortran Codes for Large-Scale Scientific Computation*, 2016, http://www.hsl.rl.ac.uk/.
[22] D. Hysom and A. Pothen, *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM J. Sci. Comput., 22 (2001), pp. 2194–2215, https://doi.org/10.1137/S1064827500376193.
[23] C. Janna, M. Ferronato, F. Sartoretto, and G. Gambolati, *FSAIPACK: A software package for high performance factored sparse approximate inverse preconditioning*, ACM Trans. Math. Software, 41 (2015), 10.
[24] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58, https://doi.org/10.1137/0614004.
[25] 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, IEEE Computer Society, Washington, DC, 1998, pp. 1–17.
[26] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45, https://doi.org/10.1137/S1064827597327334.
[27] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comput., 34 (1980), pp. 473–497.
[28] V. Noferini, M. Sharify, and F. Tisseur, *Tropical roots as approximations to eigenvalues of matrix polynomials*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 138–157, https://doi.org/10.1137/14096637X.
[29] M. Olschowka and A. Neumaier, *A new pivoting strategy for Gaussian elimination*, Linear Algebra Appl., 240 (1996), pp. 131–151.
[30] J. K. Reid and J. A. Scott, *Ordering symmetric sparse matrices for small profile and wavefront*, Internat. J. Numer. Methods Engrg., 45 (1999), pp. 1737–1755.
[31] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.

[32] O. Schenk, S. Röllin, and A. Gupta, *The effects of unsymmetric matrix permutations and scalings in semiconductor device and circuit simulation*, IEEE Trans. Computer-Aided Design Integrated Circuits Syst., 23 (2004), pp. 400–411.
[33] 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 nonconvex interior-point optimization*, Comput. Optim. Appl., 36 (2007), pp. 321–341.
[34] J. A. Scott and M. Tůma, *The importance of structure in incomplete factorization preconditioners*, BIT, 51 (2011), pp. 385–404.
[35] J. A. Scott and M. Tůma, HSL_MI28*: An efficient and robust limited-memory incomplete Cholesky factorization code*, ACM Trans. Math. Software, 40 (2014), 24.
[36] J. A. Scott and M. Tůma, *On positive semidefinite modification schemes for incomplete Cholesky factorization*, SIAM J. Sci. Comput., 36 (2014), pp. A609–A633, https://doi.org/10.1137/130917582.
[37] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, Internat. J. Numer. Methods Engrg., 23 (1986), pp. 239–251.
[38] S. W. Sloan, *A Fortran program for profile and wavefront reduction*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 2651–2679.