SIAM J. MATRIX ANAL. APPL.
Vol. 34, No. 4, pp. 1431–1447

OPTIMAL WEIGHTED MATCHINGS FOR RANK-DEFICIENT SPARSE MATRICES*

J. D. HOGG† AND J. A. SCOTT†

Abstract. The maximum matching of a sparse matrix A that maximizes the product of the matched entries can be used to increase the speed and reliability of sparse linear algebra operations. One popular method of solution is to transform to an assignment problem and to use a sparse variant of the Hungarian algorithm. If A is structurally rank deficient, this approach chooses a set $\mathcal{I} \times \mathcal{J}$ of rows and columns such that the restriction of A to $\mathcal{I} \times \mathcal{I}$ is nonsingular but, in general, the chosen $\mathcal{I}$ is suboptimal. In this paper, we propose modifying the approach to obtain an optimal $\mathcal{I}$. We focus on the symmetric case and present results for rank-deficient sparse symmetric matrices arising from practical applications.

Key words. Hungarian algorithm, matrix scaling, maximum weight maximum cardinality matching, rank-deficient matrix, singular matrix, sparse matrix, weighted matching

AMS subject classifications. 05C70, 65F35, 65F50

DOI. 10.1137/120884262

1. Introduction.

In the mid 1990s, Olschowka and Neumaier [15] introduced the use of weighted matchings for the scaling of unsymmetric matrices; their ideas were further developed and implemented for sparse unsymmetric problems by Duff and Koster [6] (see also Gupta and Ying [9]). The symmetric adaption was originally proposed by Duff and Gilbert [5] in 2002 and was subsequently built upon by Duff and Pralet [7]. Over the last ten years or so, the use of both unsymmetric and symmetric matchings has been widely adopted by the sparse linear algebra community (see, for example, [2, 10, 13, 16, 17]). Of particular note is the widely used software package `MC64` from the HSL Mathematical Software Library [11] that implements the work of [6] and [7].

The core problem is as follows: given an $n \times n$ matrix $A = \{a_{ij}\}$, find a matching of the rows to the columns such that the product of the matched entries is maximized. That is, find a permutation $\sigma = \{\sigma(i)\}$ that solves the maximum product matching problem

$$\max_{\sigma} \prod_{i=1}^{n} |a_{i\sigma(i)}|. \tag{1.1}$$

If A is structurally rank deficient with structural rank $r < n$, this problem is ill-posed because the objective takes the value 0 for all possible σ. In this case, an ideal algorithm should return a matching on $\mathcal{I} \times \mathcal{J}$ (where $\mathcal{I}$ is a subset of the rows and $\mathcal{J}$ is a subset of the columns) of cardinality r that solves the problem

$$\max_{\mathcal{I}:|\mathcal{I}|=r} \max_{\sigma} \prod_{i \in \mathcal{I}} |a_{i\sigma(i)}|. \tag{1.2}$$

*Received by the editors July 11, 2012; accepted for publication (in revised form) by B. Hendrickson July 19, 2013; published electronically October 1, 2013.
http://www.siam.org/journals/simax/34-4/88426.html
†Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Oxford, Didcot, Oxfordshire OX11 0QX, UK (Jonathan.Hogg@stfc.ac.uk, Jennifer.Scott@stfc.ac.uk). This work supported by EPSRC grant EP/I013067/1.

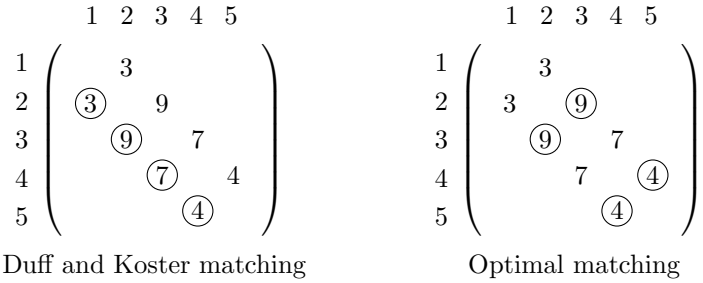

FIG. 1.1. *Example demonstrating the suboptimality of the Duff and Koster algorithm on a symmetric structurally rank-deficient matrix. Matched entries are circled.*

The algorithm implemented by Duff and Koster (which is summarized in section 2) fails to do this. Instead, it returns a matching of cardinality r that is optimal on a submatrix of A. This is demonstrated in Figure 1.1. Here and elsewhere, matrix entries are circled to indicate that they belong to a matching.

In this paper, we consider the structurally rank-deficient case and, in particular, the symmetric structurally rank-deficient case. This is of interest because sparse symmetric indefinite linear systems that are structurally rank deficient can arise in a range of practical applications (including, for example, optimization). Duff and Pralet [7] also considered this case. In particular, they proved that if A is symmetric and there is a maximum matching of A on $\mathcal{I} \times \mathcal{J}$, then $A_{\mathcal{I}\times\mathcal{I}}$ (the restriction of A to $\mathcal{I} \times \mathcal{I}$) is structurally nonsingular. Thus there is a maximum matching for $A_{\mathcal{I}\times\mathcal{I}}$ of cardinality r, and this can be used to obtain a matching for A. However, their approach may lead to a suboptimal matching (that is, the choice of $\mathcal{I}$ may be suboptimal). The contribution of this paper is to use a modified approach to obtain an optimal solution.

This paper is organized as follows. We end this section by introducing notation and terminology that we use in the remainder of the paper. The algorithm implemented by Duff and Koster in `MC64` is briefly described in section 2. We discuss the extensions by Duff and Pralet for symmetric matrices (including structurally singular matrices) in section 3. In section 4, a new preprocessing is proposed for symmetric matrices. Modifications to the Duff and Koster algorithm for structurally rank-deficient matrices are discussed in section 5. An analysis of our algorithmic modification when applied to symmetric problems from practical applications is presented in section 6. Finally, some concluding comments are made in section 7.

We remark that there is a wealth of literature on algorithms for computing maximum product matchings. For a survey and a list of references, we refer the reader to the report by Gupta and Ying [9]. This has an emphasis on the sparse case and presents a useful tutorial description of implementations of some algorithms. We also note that, as far as we are aware, the approach that we propose for structurally rank-deficient sparse matrices is new in the sparse linear algebra community. However, in the graph algorithms community the equivalent *maximum weight maximum cardinality matching* problem has been studied. The recent paper of Banerjee, Chowdhury, and Ghosh [1] surveys the state-of-the-art and develops an improved approach. Their algorithm converts the maximum weight maximum cardinality matching problem to a maximum weight assignment problem through the simple expedient of adding a sufficiently large number to each edge weight (following earlier works such as LEDA [14]). This assignment problem is then solved using a standard algorithm

with $O(n(\tau + n \log n))$ worst case complexity. However, the apparent average case complexity is typically much lower, and this is true of the algorithms throughout this paper. Our approach differs in that we instead modify a maximum weight assignment algorithm already in widespread use by the sparse linear algebra community, and it also includes the ability to generate matchings on symmetric subgraphs.

1.1. Notation and terminology. Let $A = \{a_{ij}\}$ be a general $n \times n$ sparse matrix with τ nonzero entries. With A we associate a bipartite graph $\mathcal{G}_A = (V_{row} \cup V_{col}, E)$. The node sets $V_{row} = \{row_1, row_2, \ldots, row_n\}$ and $V_{col} = \{col_1, col_2, \ldots, col_n\}$ correspond to the rows and columns of A; an edge (row_i, col_j) belongs to E if and only if $a_{ij} \neq 0$. An edge $(row_i, col_j) \in E$ is said to be *incident* on nodes row_i and col_j.

An edge subset $\mathcal{M} \subseteq E$ is called a *matching* (or *assignment*) if no two edges in $\mathcal{M}$ are incident to the same node. In matrix terms, a matching corresponds to a set of nonzero entries with no two in the same row or column. A node is matched if there is an edge in the matching incident on the node, and is unmatched (or *free*) otherwise. We denote by $\mathcal{M}'$ the *transpose* matching $\{(row_j, col_i) | (row_i, col_j) \in \mathcal{M}\}$.

The *cardinality* of a matching is the number of edges in it. A maximum cardinality matching (or *maximum matching*) is a matching of maximum cardinality (this is n if A is structurally nonsingular). If A is symmetric and $\mathcal{M}$ is a maximum matching in $\mathcal{G}_A$, then, because of symmetry, $\mathcal{M}'$ is also of maximum cardinality and has the set of row indices V_{col} and the set of column indices V_{row}.

A *path* $\mathcal{P}$ in $\mathcal{G}_A$ is an ordered set of edges in which successive edges are incident to the same node. $\mathcal{P}$ is called an *$\mathcal{M}$-alternating path* if the edges of $\mathcal{P}$ are alternately in $\mathcal{M}$ and not in $\mathcal{M}$. An $\mathcal{M}$-alternating path is an *$\mathcal{M}$-augmenting path* if it connects an unmatched column node with an unmatched row node.

Let $\mathcal{M}$ and $\mathcal{P}$ be subsets of E and define

$$\mathcal{M} \oplus \mathcal{P} := (\mathcal{M} \setminus \mathcal{P}) \cup (\mathcal{P} \setminus \mathcal{M}).$$

If $\mathcal{M}$ is a matching and $\mathcal{P}$ is an $\mathcal{M}$-augmenting path, then $\mathcal{M} \oplus \mathcal{P}$ is again a matching and $\mathcal{M} \oplus \mathcal{P} = |\mathcal{M}| + 1$.

Let $W = \{w_{ij}\}$ be a real-valued $n \times n$ sparse matrix with $w_{ij} \geq 0$. Let $\mathcal{G}_W = (V_{row}, V_{col}, E)$ be the corresponding bipartite graph each of whose edges $(row_i, col_j) \in E$ has weight w_{ij}. The weight $w(\mathcal{M})$ of a matching $\mathcal{M}$ in $\mathcal{G}_W$ is the sum of its edge weights, that is,

$$w(\mathcal{M}) = \sum_{(row_i, col_j) \in \mathcal{M}} w_{ij}.$$

An $\mathcal{M}$-augmenting path $\mathcal{P}$ starting at an unmatched column node j is called *shortest* if

$$w(\mathcal{M} \oplus \mathcal{P}) \leq w(\mathcal{M} \oplus \tilde{\mathcal{P}})$$

for all other possible M-augmenting paths $\tilde{\mathcal{P}}$ starting at node j. The *length* of an alternating path $\mathcal{P}$ is defined to be

$$l(\mathcal{P}) := w(\mathcal{M} \oplus \mathcal{P}) - w(\mathcal{M}) = w(\mathcal{P} \setminus \mathcal{M}) - w(\mathcal{M} \cap \mathcal{P}).$$

Finally, we define a *weight-augmenting path* $\mathcal{P}$ to be an $\mathcal{M}$-alternating path such that $l(\mathcal{P}) > 0$. Observe that $\mathcal{P}$ can be weight-augmenting without being $\mathcal{M}$-augmenting as it may contain a single unmatched row or column node.

In the following, if $\mathcal{M}$ is a matching, $\mathcal{I}$ and $\mathcal{J}$ denote the set of row and column indices matched by $\mathcal{M}$, i.e., $\mathcal{I} = \{i : (row_i, col_j) \in \mathcal{M} \text{ for some } j\}$ and $\mathcal{J} = \{j : (row_i, col_j) \in \mathcal{M} \text{ for some } i\}$. The matrix formed by selecting the rows and columns indexed by $\mathcal{I}$ and $\mathcal{J}$, respectively, is called the submatrix of A *restricted* to the rows in $\mathcal{I}$ and the columns in $\mathcal{J}$ and is denoted by $A_{\mathcal{I}\times\mathcal{J}}$.

$\mathcal{M}$ corresponds to a column permutation $\sigma = \{\sigma(i)\}$; it is convenient to also refer to σ as a matching. $\mathcal{M}$ is called a *symmetric matching* if and only if $j = \sigma(i) \Rightarrow i = \sigma(j)$.

2. The Duff and Koster algorithm. As we explain in this section, the algorithm of Duff and Koster transforms the maximum product matching problem to an assignment problem that can be solved with minimal modifications to standard methods.

Let $c_j = \max_i |a_{ij}|$ be the maximum absolute value in column j of A. Define a preprocessing of A to $\hat{A} = \{\hat{a}_{ij}\}$ as follows:

$$\hat{a}_{ij} = \begin{cases} \log c_j - \log |a_{ij}| & \text{for } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases} \tag{2.1}$$

The linear sum assignment problem

$$\min \sum_{i=1}^{n} \hat{a}_{i\sigma(i)} \tag{2.2}$$

has the same optimal permutation as the maximum product matching problem (1.1). Further, it is well known (see [8, 12]) that if there exists variables u_i and v_j with

$$\hat{a}_{ij} - u_i - v_j \begin{cases} = 0 & \text{for } j = \sigma(i), \\ \geq 0 & \text{otherwise,} \end{cases}$$

then u_i and v_j are optimal dual variables for (2.2). The dual solution may be used to scale the matrix [15] by defining diagonal matrices S^r and S^c with entries

$$\begin{aligned} s_i^r &= exp(u_i), \\ s_j^c &= exp(v_j - \log c_j). \end{aligned} \tag{2.3}$$

The matched entries in the scaled matrix $S^r A S^c$ are one in absolute value and all other entries are less than or equal to one in absolute value, that is,

$$|(S^r A S^c)_{ij}| \begin{cases} = 1 & \text{for } j = \sigma(i), \\ \leq 1 & \text{otherwise.} \end{cases}$$

The assignment problem can be solved efficiently using a variant of the Hungarian algorithm [12] outlined as Algorithm 1. The algorithm can be initialized by setting $\mathcal{I} = \phi, \mathcal{J} = \phi, u = 0, v = 0$, although in practice it is warm-started. For full rank matrices, each iteration of the algorithm adds an additional row i to $\mathcal{I}$ and an additional column j to $\mathcal{J}$ and updates the matching so that it is optimal on the new $\mathcal{I}\times\mathcal{J}$. If the matrix is structurally rank deficient, the cardinality of the final matching will be $r < n$, where r is the structural rank of A. The matching is optimal on $A_{\mathcal{I}\times\mathcal{J}}$ but may be suboptimal on A (that is, another choice of $\mathcal{I} \times \mathcal{J}$ could result in increasing (1.2)).

The set $\mathcal{I} \times \mathcal{J}$ is extended by selecting an unmatched column j and then finding the shortest σ-augmenting path $\mathcal{P}$ from j to an unmatched row i. In matrix terms,

ALGORITHM 1. HUNGARIAN ALGORITHM FOR ASSIGNMENT PROBLEM.

Input: Preprocessed matrix $\hat{A}$, an initial matching σ, and dual variables u, v that are optimal on $\mathcal{I}_{in} \times \mathcal{J}_{in}$.

Set $\mathcal{I} = \mathcal{I}_{in}$ and $\mathcal{J} = \mathcal{J}_{in}$.
for each unmatched column j **do**
  Find the shortest σ-augmenting path $\mathcal{P}$ (with length $lsap$) from j to an unmatched row i.
  if (no such path $\mathcal{P}$ exists) **cycle** *! that is, go to next unmatched column j*
  For each column k, let $l(k)$ be the length of the shortest alternating path from j to k.
  for each row t for which $l(\sigma(t)) < lsap$ **do**
    $u_t \leftarrow u_t + l(\sigma(t)) - lsap$
  end for
  $\sigma \leftarrow \sigma \oplus \mathcal{P}$ and $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}, \mathcal{J} \leftarrow \mathcal{J} \cup \{j\}$.
  for each row $t \in \mathcal{I}$ **do**
    $v_{\sigma(t)} = \hat{a}_{t\sigma(t)} - u_t$,
  end for
end for

Output: Matching σ and dual variables u, v that are optimal on $\mathcal{I} \times \mathcal{J}$.

an alternating path starts from the unmatched column j and proceeds through zero or more pairs of entries $(\hat{a}_{tk}, \hat{a}_{t\sigma(t)})$ to subsequent columns before terminating at an unmatched row. The edge weights are given by $\hat{a}_{t\sigma(t)} - u_t - v_{\sigma(t)}$. The Hungarian algorithm is essentially the repeated application of a sparse version of Dijkstra's algorithm [4] to accomplish the task of extending a matching on a column-by-column basis.

Once $\mathcal{P}$ is found, the matching σ is augmented along it, that is, all the unmatched entries on the path become matched entries, while the previously matched entries become unmatched. As the shortest σ-augmented path was chosen, the new matching is optimal on the extended set $\tilde{\mathcal{I}} \times \tilde{\mathcal{J}}$ with $\tilde{\mathcal{I}} = \mathcal{I} \cup \{i\}$ and $\tilde{\mathcal{J}} = \mathcal{J} \cup \{j\}$.

The dual variables u and v must be updated for the new matching. Let $lsap$ be the length of the shortest augmenting path and $l(k)$ be the length of the shortest alternating path from j to column k. For each row t for which $l(\sigma(t)) < lsap$, the update

$$u_t \leftarrow u_t + l(\sigma(t)) - lsap$$

is computed. Then for each $t \in \tilde{\mathcal{I}}$,

$$v_{\sigma(t)} = \hat{a}_{t\sigma(t)} - u_t.$$

Further details may be found in [6], where the worst case complexity is given as $O(n(\tau + n) \log n)$. For clarity, we summarize the complete algorithm implemented by Duff and Koster for unsymmetric matrices as Algorithm 2.

3. Duff and Pralet modification for symmetric matrices. So far, we have described the matching algorithm for a general (unsymmetric) matrix. We now focus on the symmetric case. First, we observe that a symmetric matrix need not have a

ALGORITHM 2. MAXIMUM WEIGHTED MATCHING ALGORITHM (DUFF AND KOSTER).

Input: Unsymmetric matrix A.

1. Apply the unsymmetric preprocessing (2.1) to A to obtain $\hat{A}$.
2. Determine an initial matching σ and dual variables u, v optimal on $\mathcal{I}_{in} \times \mathcal{J}_{in}$.
3. Apply Algorithm 1 to $\hat{A}$.
   Return the maximum matching and dual variables.
   Compute the scaling factors (2.3).

Output: A column permutation (with unmatched columns flagged) and scaling factors for A.

symmetric matching. To illustrate this, consider the following example:

$$\begin{pmatrix} & \textcircled{1} & 2 \\ 1 & & \textcircled{3} \\ \textcircled{2} & 3 & \end{pmatrix}.$$

However, it is generally important when scaling a symmetric matrix that symmetry is preserved (i.e., the scaling factors (2.3) should satisfy $s = s^r = s^c$).

Duff and Pralet [7] show that if A is nonsingular, the geometric average of the row and column scalings is sufficient to maintain desirable properties. That is, they build a diagonal scaling matrix S with entries

$$s_i = \sqrt{s_i^r s_i^c} \tag{3.1}$$

with s_i^r and s_i^c given by (2.3), and compute the symmetrically scaled matrix SAS. The entries in the scaled matrix that are in the matching have absolute value one, while the rest have absolute value less than or equal to one.

For rank-deficient matrices, Duff and Pralet prove that, if Algorithm 1 returns a matching on $\mathcal{I} \times \mathcal{J}$ (with cardinality equal to the structural rank r of A), $A_{\mathcal{I}\times\mathcal{I}}$ is structurally nonsingular. A maximum matching for $A_{\mathcal{I}\times\mathcal{I}}$ is obtained by applying Algorithm 1 to $A_{\mathcal{I}\times\mathcal{I}}$ and then mapped back to a (suboptimal) matching for A (with $n - r$ rows and columns unmatched). If the scaling factors for $A_{\mathcal{I}\times\mathcal{I}}$ are s_i^r and s_j^c, Duff and Pralet propose using scaling factors

$$s_i = \begin{cases} \sqrt{s_i^r s_i^c} & \text{if } i \in \mathcal{I}, \\ \dfrac{1}{\max_{k\in\mathcal{I}} |a_{ik} s_k|} & \text{otherwise} \end{cases} \tag{3.2}$$

with the convention $1/0 = 1$. With this choice, the absolute values of the entries of SAS are again less than or equal to one, with those in the matching having absolute value equal to one. Moreover, the entry of largest absolute value in each nonempty row/column is one.

The approach of Duff and Pralet is summarized in Algorithm 3.

4. Symmetric preprocessing. If A is symmetric, in general the preprocessing (2.1) results in an unsymmetric matrix $\hat{A}$. This is undesirable in the rank-deficient case. For example, consider the following rank-2 matrix, where the optimal solution

ALGORITHM 3. VARIANT FOR SYMMETRIC MATRICES (DUFF AND PRALET).

Input: An $n \times n$ symmetric matrix A.

1. Apply the unsymmetric preprocessing (2.1) to A to obtain $\hat{A}$.
2. Determine an initial matching σ and dual variables u, v optimal on $\mathcal{I}_{in} \times \mathcal{J}_{in}$.
3. Apply Algorithm 1 to $\hat{A}$ to obtain a maximum matching on $\mathcal{I} \times \mathcal{J}$.
4. **if** $(|\mathcal{I}| = n)$ **then**
   Compute the scaling factors (3.1).
   else
5. Apply Algorithm 1 to $A_{\mathcal{I}\times\mathcal{I}}$
   Use the maximum matching for $A_{\mathcal{I}\times\mathcal{I}}$ to compute a matching for A.
   Compute the scaling factors (3.2).
   end if

Output: A symmetric permutation (with unmatched rows/columns flagged) and scaling factors for A.

to (1.2) has been circled

$$\begin{pmatrix} 1.0 & 10^3 & \textcircled{10^9} \\ 10^3 & & \\ \textcircled{10^9} & & \end{pmatrix}.$$

Preprocessing (using $\log_{10}$ for convenience) gives

$$\begin{pmatrix} 9 & 0 & 0 \\ 6 & & \\ 0 & & \end{pmatrix}.$$

In this case it is impossible to distinguish columns 2 and 3, and Algorithm 1 matches columns 1 and 2 to rows 3 and 1, respectively.

In the nonsingular case, this loss of symmetry is not a problem since all the entries in each column are scaled and, by nonsingularity, there is sufficient information retained to distinguish between the columns.

For symmetric matrices, to retain symmetry, we propose the following preprocessing:

$$\hat{a}_{ij} = (\log c_j + \log r_i)/2 - \log |a_{ij}|, \tag{4.1}$$

where $r_i = \max_j |a_{ij}|$ is the maximum absolute value in row i (observe $c_i = r_i$ for symmetric matrices).

Applying (4.1) to our example, Algorithm 1 returns the desired matching

$$\begin{pmatrix} 9 & 3 & \textcircled{0} \\ 3 & & \\ \textcircled{0} & & \end{pmatrix}.$$

Although for this simple case, restricting the matching to $\mathcal{I} \times \mathcal{I}$ would have been sufficient to obtain the better matching, this is not necessarily always the case. Thus we

need to also consider modifying the matching algorithm for rank-deficient symmetric matrices, and this is what we consider in the next section.

5. Modified algorithm for rank-deficient matrices. Recall the example from Figure 1.1: the reason that Algorithm 1 returns a suboptimal matching is as follows. Column 5 is the last unmatched column to be considered, with the matching shown in Figure 1.1 already existing for columns 1 to 4. There is no path between column 5 and row 1, so no augmenting path is found. Hence column 5 remains unmatched regardless of the values of the nonzero entries. A suboptimal matching is only possible in the structurally rank-deficient case.

To motivate our proposed approach, consider the application of Algorithm 1 to the following modified version of the example matrix. All the zeros have been replaced by a nonzero value ϵ, a number so small that it will never be chosen as part of the matching if there is an alternative:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} \textcircled{\epsilon} & 3 & \epsilon & \epsilon & \epsilon \\ 3 & \epsilon & \textcircled{9} & \epsilon & \epsilon \\ \epsilon & \textcircled{9} & \epsilon & 7 & \epsilon \\ \epsilon & \epsilon & 7 & \epsilon & \textcircled{4} \\ \epsilon & \epsilon & \epsilon & \textcircled{4} & \epsilon \end{pmatrix} \end{array}.$$

By the optimality of Algorithm 1 for nonsingular matrices, the computed matching shown is optimal. The rank-deficiency is apparent in that the matching contains an epsilon in the (1,1) position. An optimal matching for the original rank-deficient matrix is obtained by restricting the matching to those rows and columns that are not matched on an epsilon.

This approach is not practical for real problems as the conversion from a sparse to a dense matrix increases the storage (and hence memory bandwidth) requirements to $O(n^2)$. However, by implementing the addition of the ϵ entries implicitly, the approach can be made workable, and this is what we now explain.

Denote the modified matrix by $A(\epsilon)$. The preprocessing (4.1) transforms each ϵ entry to

$$\hat{a}_{ij}(\epsilon) = (\log c_j + \log r_i)/2 + \alpha,$$

where α is very large. Such entries can be calculated on the fly, and an implicit representation of $\hat{A}(\epsilon)$ can be stored in the same space as $\hat{A}$ plus a vector of column maximums and a vector of row maximums. (In the symmetric case, $c_i = r_i$ for all i so a single vector is needed.) An entry of $\hat{A}(\epsilon)$ that corresponds to an ϵ in $A(\epsilon)$ will be referred to as an α entry.

If a path exists from the current unmatched column to an unmatched row in the original matrix A and the dual variables u contain no term in α, an α entry will never be included on the path $\mathcal{P}$ during the execution of Algorithm 1 (α is too large for it to be included in a shortest path). The condition on u is trivially satisfied if no path containing α has been augmented along (as $lsap \ll \alpha$); thus it follows that α entries are never considered if A is structurally of full rank and Algorithm 1 is applied to $\hat{A}(\epsilon)$.

When an unmatched column is found and there exists no path to an unmatched row in the original matrix, we start considering α entries. Observe that since A is square, the existence of an unmatched column j in $\hat{A}(\epsilon)$ implies the existence of an

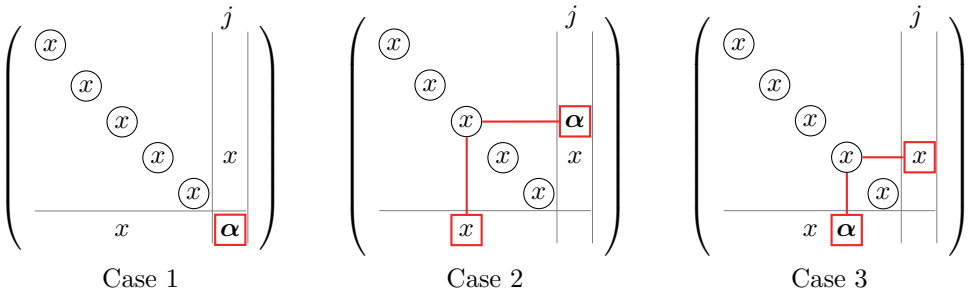

FIG. 5.1. *Different paths involving α entries.*

unmatched row i and, as no path to i exists in A, each of the entries in column j that corresponds to an unmatched row must be α entry. Hence a trivial path consisting of a single entry exists with length $\alpha + \gamma$, where $|\gamma| \ll \alpha$. If the dual variables u are much less than α, we do not need to consider paths that contain more than one α entry since the lengths of such paths will be at least 2α (and hence greater than the length of the trivial path). There are three possibilities for paths with one α entry:

1. A trivial path is chosen using an α entry in column j and an unmatched row.
2. An α entry in column j and a currently matched row i is chosen. There exists a path from column $\sigma(i)$ that is matched with row i to an unmatched row.
3. An α entry in a column belonging to Reach(j) and an unmatched row is chosen, where Reach(j) is the set of columns k such that a path exists from j to k in the original matrix A.

These are illustrated in Figure 5.1. Note that case 1 is a special case of case 3. We further observe that we can choose to augment along paths involving both cases 2 and 3, as they must be disjoint.

So far, we have assumed that the dual variables u are much less than α. This requires that there are no α entries in the current matching. If we were to relax this requirement, the storage needed for u would double because a coefficient for α would have to be stored. (Merely using a large value results in loss of the precision needed to distinguish between candidate paths.) Moreover, for some columns, all the entries (including the α entries) would need to be considered, and this means the full matrix would have to be explored each time such a column is encountered. This additional computation is not desirable and is prohibitive for large problems. To avoid this, we propose that rather than including the α entry in the matching, we augment along the path excluding the α value. The size of the matching will not increase, but the optimal value will be improved. That is, once the algorithm starts considering the α entries, the cardinalities $|\mathcal{I}|$ and $|\mathcal{J}|$ remain constant, with one column (or row) swapped for another at each iteration. This may result in a suboptimal result. However, since an improvement in the objective value is made with each such augmentation, repeatedly iterating over unmatched columns until no further improvement is made will terminate with the optimal solution in a finite number of steps.

Let us consider the practical implementation further. For case 3, the set Reach(j) and path lengths $l(k)$ are built in the process of determining that there is no path for the original matrix $\hat{A}$ from the unmatched column j to an unmatched row. Since it does not matter which particular unmatched row i the α entry belongs to (it will not be part of the augmenting path), contributions to the path length from $\log r_i$ and u_i for the unmatched row can be ignored. We thus choose the column $k \in$ Reach(j) for

which $l(k) + log(c_k)/2 - v_k$ is a minimum. The column set $\mathcal{J}$ is altered by removing k and including j.

Case 2 requires considerably more work. For each matched row with matching column not in Reach(j), we must explore possible paths that lead to an unmatched row and pick the shortest. This is very costly. However, observe that Algorithm 1 can be implemented rowwise rather than columnwise, and case 2 in a rowwise implementation is covered by case 3 in a columnwise implementation. A rowwise implementation is equivalent to running the columnwise algorithm on A^T. The optimal solution can thus be found by applying Algorithm 1 alternately to A and A^T until the objective value ceases to decrease. In the symmetric case, $A = A^T$, and this can be exploited to save storage (provided symmetric preprocessing is used).

Based on our practical experience, we found it necessary to specify a minimum improvement (`min_improv`) to avoid cycling between two identical columns (or identical rows when running on A^T) because of floating-point inaccuracies. That is, we want to avoid at one step removing k from the column set $\mathcal{J}$ and including j and then, at the next step, removing j and including k. We found it to be sufficient to require a minimum decrease of the square root of machine precision before altering $\mathcal{J}$. Using this, for each of the test problems used in section 6, a single application to A followed by one to A^T yielded the optimal solution. This was unexpected and is not guaranteed. (We are unable to prove that this should yield the optimal solution.)

These modifications are summarized as a generalized Hungarian algorithm (Algorithm 4), which is for both symmetric and unsymmetric A, and in our proposed variant of the symmetric matrix matching and scaling algorithm (shown as Algorithm 5). If A is structurally nonsingular, Algorithm 4 reduces to Algorithm 1 but it differs in the action taken if no σ-augmenting path $\mathcal{P}$ is found and in the addition of a flag (`change_flag`) to indicate whether changes to σ have occurred. Observe that as Reach(j) was constructed as part of Algorithm 1, the new algorithm has the same worst case complexity of $O(n(\tau+n)\log n)$. In Algorithm 5, the generalized Hungarian algorithm is applied repeatedly to A and A^T until either the matching has cardinality n or no changes to σ are made. Then, in the rank-deficient case, we proceed as in the variant of Duff and Pralet to compute $A_{\mathcal{I}\times\mathcal{I}}$ and apply Algorithm 4 to it. The maximum matching obtained for $A_{\mathcal{I}\times\mathcal{I}}$ is finally mapped to a matching for A. We note that if A is nonsingular, a single application of the generalized Hungarian algorithm to A gives a matching of cardinality n and, in the rank-deficient case, a single application of the generalized Hungarian algorithm to $A_{\mathcal{I}\times\mathcal{I}}$ gives a matching of cardinality r. We further observe that when $A_{\mathcal{I}\times\mathcal{I}}$ is computed, $\mathcal{I}$ and $\mathcal{J}$ are optimal and so we can take the restriction of A to $\mathcal{I}\times\mathcal{I}$ or to $\mathcal{J}\times\mathcal{J}$.

In graph terms, Algorithm 4 extends the Hungarian algorithm with the ability to find weight augmenting paths. Algorithm 5 first finds a maximum cardinality matching with maximum weight on $\mathcal{I}\times\mathcal{J}$; it then seeks weight augmenting paths starting alternately from the set of unmatched columns and the set of unmatched rows. As a weight augmenting path does not reduce the cardinality and always leads to a nontrivial improvement in the weight, Algorithm 5 must return with a maximum weight maximum cardinality matching in finite time.

6. Numerical experiments. For our numerical experiments, we use the two sets of symmetric structurally rank-deficient test problems given in Tables 6.1 and 6.2. Both are taken from the University of Florida Sparse Matrix Collection [3]. Test Set 1 comprises 11 problems from the GHS_indef, Newman, and Pajek groups; for this set we use the supplied values for the matrix entries. Test Set 2 consists of 8 problems that are structurally rank deficient but, because only the matrix pattern is available,

ALGORITHM 4. GENERALIZED HUNGARIAN ALGORITHM FOR THE DEGENERATE ASSIGNMENT PROBLEM.

Input: Preprocessed matrix $\hat{A}$, column maximums c_j for original matrix A, an initial matching σ, and dual variables u, v that are optimal on $\mathcal{I}_{in} \times \mathcal{J}_{in}$, and a minimum improvement threshold `min_improv`.

Set $\mathcal{I} = \mathcal{I}_{in}$ and $\mathcal{J} = \mathcal{J}_{in}$.
`change_flag` = `false`.
for each unmatched column j **do**
  For each column k, let $l(k)$ be the length of the shortest σ-alternating path from j to k.
  Find the shortest σ-augmenting path $\mathcal{P}$ (with length $lsap$) from j to an unmatched row i.
  if (no such path $\mathcal{P}$ exists) **then**
    Find column $k \in \text{Reach}(j)$ that minimizes $\delta_k = l(k) + log(c_k)/2 - v_k$.
    if $(\delta_k + \texttt{min\_improv} \geq \delta_j)$ **cycle** *! that is, go to next unmatched column j.*
    Let $\mathcal{P}$ be the shortest σ-alternating path from j to k.
  end if
  for each row t for which $l(\sigma(t)) < lsap$ **do**
    $u_t \leftarrow u_t + l(\sigma(t)) - lsap$
  end do
  Modify σ along $\mathcal{P}$ and, if $\mathcal{P}$ is σ-augmenting, $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}, \mathcal{J} \leftarrow \mathcal{J} \cup \{j\}$, otherwise, $\mathcal{J} \leftarrow (\mathcal{J} \setminus \{k\}) \cup \{j\}$.
  for each row $t \in \mathcal{I}$ **do**
    $v_{\sigma(t)} = \hat{a}_{t\sigma(t)} - u_t$,
  end do
  `change_flag` = `true`.
end do

Output: Matching σ and dual variables u, v that are optimal on $\mathcal{I} \times \mathcal{J}$. If `change_flag` = `true`, either the product of matched entries in A is larger than for the initial matching or the size of the matching is larger than on input.

we generate values for the entries. The generated values are distributed uniformly over the interval $[-1, 1]$. Note that this means that the problems in Test Set 2, some of which are highly rank deficient, are well-scaled. We comment Duff and Pralet's focus in [7] was on more general indefinite matrices; their test set included only two rank-deficient problems for which $r = n - 1$, and so they did not report on any highly rank-deficient matrices.

The tests are performed on an Intel Xeon E5420 using the gfortran complier (version 4.4) with flags -O3. Our Fortran 95 implementations are built on the original implementation of Duff and Koster that is available within `MC64`. In all the tests, an initial matching σ and dual variables u, v that are optimal on a set $\mathcal{I} \times \mathcal{J}$ are computed in the same way (again following `MC64`).

6.1. An analysis of different variants. We are interested in assessing the quality of the computed ordering (and we omit computing the scaling factors). The statistic we use to measure the quality is

$$\log \prod_{i \in \mathcal{I}} |a_{i\sigma(i)}|, \tag{6.1}$$

ALGORITHM 5. VARIANT FOR SYMMETRIC MATRICES (HOGG AND SCOTT).

Input: An $n \times n$ symmetric matrix A and a minimum improvement threshold `min_improv`.

1. Compute row and column maximums r_i and c_j;
   apply symmetric preprocessing (4.1) to A to obtain $\hat{A}$.
2. Determine an initial matching σ and dual variables u, v optimal on a set $\mathcal{I} \times \mathcal{J}$.
3. `flag1` = `true` and `flag2` = `true`.
4. **while** (`flag1` or `flag2`) **do**
   `flag1` = `false`, `flag2` = `false`.
   Apply Algorithm 4 to $\hat{A}$ to improve the matching σ and $\mathcal{I} \times \mathcal{J}$,
   `flag1` = `change_flag`.
   if ($|\mathcal{I}| = n$) **exit** loop.
   Take transpose σ' and set $u' \leftarrow v, v' \leftarrow u, \mathcal{I}' \leftarrow \mathcal{J}, \mathcal{J}' \leftarrow \mathcal{I}$.
   Apply Algorithm 4 to $\hat{A}^T$ to improve σ' and $\mathcal{I}' \times \mathcal{J}'$. `flag2` = `change_flag`.
   Take transpose $\sigma = (\sigma')'$ and set $u \leftarrow v', v \leftarrow u', \mathcal{I} \leftarrow \mathcal{J}', \mathcal{J} \leftarrow \mathcal{I}'$.
   end do
5. **if** ($|\mathcal{I}| = n$) **then**
   Compute the scaling factors (3.1).
   else
6. Apply Algorithm 4 to $A_{\mathcal{I}\times\mathcal{I}}$.
   Use the maximum matching for $A_{\mathcal{I}\times\mathcal{I}}$ to compute a matching for A.
   Compute the scaling factors (3.2).
   end if

Output: A symmetric permutation (with unmatched rows/columns flagged) and scaling factors for A.

TABLE 6.1
Test Set 1: *Symmetric rank-deficient matrices from practical problems. n is the order of A, $n - r$ is the rank deficiency, and nz is the number of entries in A.*

Problem	n	$n - r$	nz
GHS_indef/aug2d	29 008	9 800 (33.8%)	76 832
GHS_indef/aug2dc	30 200	10 200 (33.8%)	80 000
GHS_indef/aug3d	24 300	12 636 (52.0%)	69 984
GHS_indef/dtoc	24 993	4 999 (20.0%)	69 972
Newman/astro-ph	16 706	990 (5.92%)	242 502
Newman/cond-mat	16 726	1 130 (6.76%)	95 188
Newman/cond-mat-2003	31 163	1 989 (6.38%)	240 058
Newman/cond-mat-2005	40 421	2 453 (6.07%)	351 382
Newman/hep-th	8 361	1 225 (14.6%)	31 502
Newman/netscience	1 589	165 (10.4%)	5 484
Pajek/Reuters911	13 332	2 647 (19.9%)	296 076

which we refer to as the objective function value. The better the matching, the greater this value is. We report results for the algorithms discussed in this paper and for a number of variants of them. We first consider matchings that do not preserve symmetry and so do not employ the restriction $A_{\mathcal{I}\times\mathcal{I}}$ that imposes symmetry. We implement the following variants:

TABLE 6.2

Test Set 2: *Symmetric rank-deficient matrices with patterns from practical problems and entries that are random values in* $[-1, 1]$. n *is the order of* A, $n - r$ *is the rank deficiency, and* nz *is the number of entries in* A.

Problem	n	$n-r$	nz
AG-Monien/diag	2 559	169 (6.60%)	8 184
AG-Monien/netz4504	1 961	617 (31.5%)	2 342
AG-Monien/stufe-10	24 010	152 (6.33%)	92 828
AG-Monien/ukerbe1	5 981	1 871 (31.3%)	15 704
DIMACS10/kron_g500-logn16	65 536	24 076 (36.7%)	4 912 469
Newman/as-22july06	22 963	16 362 (71.2%)	96 872
Pajek/Erdos981	485	58 (12.0%)	2 762
SNAP/Oregon-1	11 492	8 170 (71.1%)	46 818

I. Algorithm 2 (Duff and Koster).

II. Algorithm 2 with symmetric preprocessing.

III. Algorithm 5 with steps 5 and 6 omitted and, in step 4, only a single pass on $\hat{A}$. That is, step 4 is replaced by the following single statement:

4. Apply Algorithm 4 to $\hat{A}$ to improve the matching σ and $\mathcal{I} \times \mathcal{J}$.

IV. Algorithm 5 with steps 5 and 6 omitted.

Results are given in columns 2 to 5 of Table 6.3, and timings are reported in Table 6.4. Variant IV computes the optimal value of the objective function and, while it is independent of the initial ordering of A, the other variants depend on the initial ordering. For each problem, we present three results. In the first line, we report the value of the objective function (6.1) for the supplied ordering. We then apply each variant to 10 random permutations of A and, in the next two lines, we report the difference between the objective function value for the supplied ordering and the best and worst objective function value over the 10 permutations. Note that the reported timings are for the supplied ordering; if A is permuted, the subsequent time for computing the matching can be more or less than that given in Table 6.4. (In our experiments, timings could reduce by about 50 percent or, conversely, increase by as much as a factor of 2.) The problems GHS_indef/aug2d, GHS_indef/aug2dc, and GHS_indef/aug3d are omitted from Table 6.3 since the values of the nonzero entries in these matrices are ± 1 and so the objective function has value 0 in all cases.

Comparing variants I and II, we see that, for our test examples, symmetric preprocessing consistently computes a higher objective function value than unsymmetric preprocessing, but in both cases the objective function value can be far from optimal (for example, AG-Monien/stufe-10 and DIMACS10/kron_g500-logn16).

A comparison of the results for variants III and IV demonstrates how the matching is improved by applying Algorithm 4 to both $A_{\mathcal{I}\times\mathcal{I}}$ and $A^T_{\mathcal{I}\times\mathcal{I}}$. For problems with only a small rank deficiency, the improvement is generally small, but where there is a larger rank deficiency (such as Newman/as-22july06 and DIMACS10/kron_g500-logn16), using both $A_{\mathcal{I}\times\mathcal{I}}$ and $A^T_{\mathcal{I}\times\mathcal{I}}$ can substantially increase the objective function value. Unfortunately, the cost of achieving this improvement can be significant. Moreover, we see that Algorithm 4 (used within III and IV) can be considerably more expensive than the simpler Algorithm 1 (used within I and II) (notable examples being Pajek/Reuters911 and DIMACS10/kron_g500-logn16).

It is worth noting that the additional steps added to create Algorithm 4 from Algorithm 1 account for only a small proportion of the additional runtime. Most of it is due to the modified algorithm exploring a larger number of columns. It is perhaps also appropriate to comment that the application of Algorithm 4 with a near-optimal

TABLE 6.3

Comparison of the different variants. In each case, we give the objective function value (6.1) *for the initial ordering of* A *and the maximum increase and decrease in this value for* 10 *random permutations of* A.

Problem	I	II	III	IV	V	VI	VII	VIII
GHS_indef/dtoc	−42577	−42577	−42577	0.00	−85155	−85155	−42577	0.00
	+ 0.00	+ 6746	+ 16838	+ 0.00	+ 33677	+ 33677	+ 16838	+ 0.00
	− 5783	− 0.00	− 0.00	− 0.00	− 0.00	− 0.00	− 0.00	− 0.00
Newman/astro-ph	−15253	−15202	−15163	−15090	−15231	−15236	−15163	−15090
	+ 5.59	+ 0.00	+ 6.88	+ 0.00	+ 4.63	+ 13.76	+ 6.88	+ 0.00
	− 12.76	− 18.77	− 3.84	− 0.00	− 15.48	− 7.67	− 3.84	− 0.00
Newman/cond-mat	−10019	−9870	−9758	−9635	−9900	−9880	−9758	−9635
	+ 21.24	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00
	− 12.28	− 56.09	− 42.52	− 0.00	− 75.14	− 85.04	− 42.52	− 0.00
Newman/cond-mat-2003	−21758	−21571	−21235	−20961	−21496	−21508	−21235	−20961
	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00
	− 112.1	− 120.4	− 84.85	− 0.00	− 212.1	− 169.7	− 84.85	− 0.00
Newman/cond-mat-2005	−29695	−29449	−29022	−28673	−29392	−29370	−29022	−28673
	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00
	− 127.3	− 150.2	− 109.8	− 0.00	− 222.6	− 219.6	− 109.8	− 0.00
Newman/hep-th	−1589	−1490	−1421	−1320	−1529	−1522	−1421	−1320
	+ 16.52	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00
	− 6.87	− 33.00	− 17.83	− 0.00	− 35.98	− 35.67	− 17.83	− 0.00
Newman/netscience	−1072	−1066	−1063	−1059	−1066	−1067	−1063	−1059
	+ 3.87	+ 0.00	+ 0.88	+ 0.00	+ 0.00	+ 1.75	+ 0.88	+ 0.00
	− 3.18	− 3.79	− 1.67	− 0.00	− 4.73	− 3.35	− 1.67	− 0.00
Pajek/Reuters911	2304	2543	3338	4010	2497	2666	3338	4010
	+ 27.48	+ 12.81	+ 30.64	+ 0.00	+ 14.46	+ 61.28	+ 31.33	+ 0.00
	− 12.03	− 54.54	− 17.70	− 0.00	− 38.27	− 35.40	− 17.70	− 0.00
AG-Monien/diag	−1268	−1243	−1001	−951.9	−1037	−1050	−1001	−951.9
	+ 86.25	+ 96.91	+ 2.28	+ 0.00	+ 0.00	+ 4.56	+ 2.28	+ 0.00
	− 0.00	− 0.00	− 11.31	− 0.00	− 27.06	− 22.62	− 11.31	− 0.00
AG-Monien/netz4504	−704.4	−588.6	−554.5	−403.8	−670.0	−705.2	−554.5	−403.8
	+ 10.32	+ 31.02	+ 31.93	+ 0.00	+ 56.44	+ 63.87	+ 30.95	+ 0.00
	− 67.43	− 14.70	− 10.55	− 0.00	− 30.87	− 21.11	− 10.55	− 0.00
AG-Monien/stufe-10	−15110	−15099	−10582	−10531	−10626	−10632	−10582	−10531
	+ 3838	+ 3877	+ 5.82	+ 0.00	+ 9.42	+ 11.64	+ 5.82	+ 0.00
	− 0.00	− 0.00	− 1.61	− 0.00	− 4.11	− 3.22	− 1.61	− 0.00
AG-Monien/ukerbe1	−2247	−1872	−1696	−1242	−2063	−2150	−1696	−1242
	+ 72.41	+ 56.07	+ 49.51	+ 0.00	+ 45.99	+ 99.02	+ 49.51	+ 0.00
	− 24.27	− 29.96	− 11.86	− 0.00	− 58.92	− 23.72	− 13.18	− 0.00
DIMACS10/kron_g500-logn16	9.01	18.71	317.6	606.1	13.86	29.11	317.6	606.1
	+ 1.39	+ 1.39	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00	+ 0.00
	− 6.24	− 9.01	− 6.93	− 0.00	− 8.32	− 13.86	− 6.93	− 0.00
Newman/as-22july06	−5523	−4943	−4550	−3741	−5354	−5358	−4550	−3741
	+ 14.71	+ 0.00	+ 8.70	+ 0.00	+ 16.21	+ 17.39	+ 7.68	+ 0.00
	− 81.90	− 155.6	− 108.8	− 0.00	− 203.4	− 217.7	− 109.7	− 0.00
Pajek/Erdos981	−383.9	−374.	−371.0	−361.2	−380.9	−380.9	−371.0	−361.2
	+ 13.76	+ 6.67	+ 5.12	+ 0.00	+ 11.35	+ 10.25	+ 5.06	+ 0.00
	− 1.42	− 4.65	− 1.17	− 0.00	− 7.15	− 2.33	− 1.17	− 0.00
SNAP/Oregon-1	−2828	−2590	−2317	−1884	−2725	−2750	−2318	−1884
	+ 41.40	+ 30.97	+ 16.48	+ 0.00	+ 23.81	+ 32.96	+ 17.26	+ 0.00
	− 36.37	− 56.59	− 52.97	− 0.00	− 74.21	− 105.93	− 55.19	− 0.00

input matching is likely to be inefficient. If an unmatched column is explored and no augmenting path is found, then an efficient algorithm could cache the result gained from exploring those columns and use this information in subsequent searches. It is thus possible that with further work the time for variant IV could be brought much closer to that for variant III. However, this is outside the scope of the current study.

We next consider symmetric matchings using the following variants that are the symmetric counterparts of I to IV:

V. Algorithm 3 (Duff and Pralet).

VI. Algorithm 3 with symmetric preprocessing.

VII. Algorithm 5 with a single pass on $\hat{A}$. That is, step 4 is replaced by the following single statement:

TABLE 6.4
Times (in seconds) for each variant.

Problem	I	II	III	IV	V	VI	VII	VIII
GHS_indef/aug2d	0.043	0.043	0.054	0.099	0.045	0.045	0.056	0.104
GHS_indef/aug2dc	0.046	0.045	0.056	0.122	0.048	0.047	0.058	0.126
GHS_indef/aug3d	0.016	0.015	0.017	0.027	0.017	0.016	0.018	0.029
GHS_indef/dtoc	0.958	0.957	1.135	2.041	0.960	0.959	1.136	2.043
Newman/astro-ph	0.081	0.123	0.123	0.124	0.140	0.224	0.223	0.224
Newman/cond-mat	0.022	0.026	0.026	0.026	0.034	0.041	0.041	0.043
Newman/cond-mat-2003	0.098	0.140	0.139	0.141	0.173	0.248	0.247	0.247
Newman/cond-mat-2005	0.185	0.249	0.244	0.251	0.318	0.458	0.459	0.454
Newman/hep-th	0.009	0.008	0.008	0.009	0.013	0.013	0.013	0.014
Newman/netscience	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Pajek/Reuters911	0.183	0.113	0.846	1.910	0.196	0.127	0.860	1.948
AG-Monien/diag	0.011	0.011	0.006	0.021	0.011	0.011	0.006	0.021
AG-Monien/netz4504	0.002	0.001	0.001	0.002	0.002	0.001	0.002	0.002
AG-Monien/stufe-10	0.287	0.283	0.172	0.540	0.333	0.329	0.219	0.589
AG-Monien/ukerbe1	0.005	0.004	0.004	0.006	0.005	0.005	0.005	0.007
DIMACS10/kron_g500-logn16	8.013	5.024	52.31	133.7	8.382	5.327	52.55	138.1
Newman/as-22july06	0.026	0.034	0.098	0.179	0.027	0.036	0.100	0.186
Pajek/Erdos981	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
SNAP/Oregon-1	0.009	0.011	0.027	0.048	0.010	0.012	0.028	0.050

4. Apply Algorithm 4 to $\hat{A}$ to improve the matching σ and $\mathcal{I} \times \mathcal{J}$, and, at step 6, use $A_{\mathcal{J}\times\mathcal{J}}$.

VIII. Algorithm 5 (Hogg and Scott).

Results are given in columns 6 to 9 of Table 6.3, with timings in Table 6.4. Each variant V–VIII is again dependent on the initial ordering. Note that only variants I and V use unsymmetric preprocessing (2.1), while the remainder use symmetric preprocessing (4.1).

Looking at I and V, we can see how the Duff and Pralet strategy that imposes symmetry affects the performance. In some cases (including GHS_indef/dtoc) it leads to a poorer matching, but it can also give a better result. These mixed findings are because neither is optimal. Comparing V and VI allows us to assess how symmetric preprocessing effects the Duff and Pralet algorithm. We see that it can be beneficial but, disappointingly, the benefit is relatively small and there is no consistent advantage.

There are two key differences between VI (Duff and Pralet with symmetric preprocessing) and VII (Hogg and Scott applied to A only). The first is the action taken when no σ-augmenting path is found. Variant VI terminates, whereas VII continues to look to alter the column set $\mathcal{J}$ in such a way as to increase the product of the matched entries. Second, having chosen $\mathcal{I}$, variant VI uses $A_{\mathcal{I}\times\mathcal{I}}$. However, in VII, we interchange the roles of $\mathcal{I}$ and $\mathcal{J}$ and use $A_{\mathcal{J}\times\mathcal{J}}$. This is because it is $\mathcal{J}$ that has been constructed by Algorithm 4 to be optimal. (If we were to use $A_{\mathcal{I}\times\mathcal{I}}$, variant VII would reduce to variant VI since both give the same set $\mathcal{I}$.) Note that we experimented with modifying VI to also work with $\mathcal{J} \times \mathcal{J}$, but, as neither the $\mathcal{I}$ nor the $\mathcal{J}$ constructed by the Duff and Pralet approach is optimal, this was not beneficial.

A comparison of variants VII and VIII demonstrates how the matching is improved by applying Algorithm 4 to both $A_{\mathcal{I}\times\mathcal{I}}$ and $A^T_{\mathcal{I}\times\mathcal{I}}$. As before, using $A^T_{\mathcal{I}\times\mathcal{I}}$ offers a consistent advantage (with the largest benefit generally being for problems with a large rank deficiency), but for some problems the time overhead to achieve this can be substantial.

Since IV produces the optimal value of the objective function, comparing it with VIII illustrates the effect of imposing symmetry. For each of our test problems, the

symmetry constraint leads to no decrease in the objective function value (and the VIII results are seen not to be dependent on the initial ordering). Similarly, comparing III and VII illustrates the effect of imposing symmetry when only A is used. For some examples, the objective function value decreases when symmetry is imposed, but the improvements are very small. From Table 6.4, we see that the symmetry time overhead is generally less than 10 percent, although it can be close to 100 percent (for example, problem Newman/cond-mat).

7. Concluding remarks. In this paper, we have shown how to modify earlier work on weighted matchings to obtain optimal weighted matchings for rank-deficient matrices. Our approach is applicable to both unsymmetric and symmetric matrices, but our main emphasis is on computing optimal symmetric matchings. We have modified the approach of Duff and Pralet for symmetric matrices in three key ways. First, we have introduced a symmetric preprocessing to retain symmetry. Second, we implicitly work with a modified matrix $A(\epsilon)$ that allows us to circumvent the problem of some rows/columns being unmatched in the structurally singular case. Finally, by working with both A and A^T we have developed an implementation that is practical for large problems. We have shown, using examples from real applications, that while our approach can involve a considerable time overhead, the benefit for some problems is a significant improvement in the quality of the matching. We remark that it is not possible to know a priori how far the matching returned by the Duff and Pralet algorithm is from optimal and hence how much the matching will be improved by our modified approach.

Finally, we remark that the our modified approach may be useful when solving sparse symmetric linear systems in parallel. It is frequently the case that the parallel computation starts by computing a partitioning of the matrix prior to the numerical solution procedure and before any matching algorithm is applied. (Note that the development of high-quality and efficient parallel matching algorithms remains a challenge.) For symmetric indefinite matrices, this can lead to highly singular submatrices, which is precisely the situation addressed in this paper. Thus in the future, we would like to study how effective our approach is in improving performance of the solver that is applied to the subproblems.

Acknowledgments. We would like to thank Iain Duff for commenting on a draft of this paper. We would also like to thank three anonymous reviewers and the associate editor for their suggestions and constructive feedback.

REFERENCES

[1] S. Banerjee, A. D. Chowdhury, and S. K. Ghosh, *Efficient algorithms for variants of weighted matching and assignment problems*, Math. Comput. Sci., 1 (2008), pp. 673–688.

[2] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353.

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

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

[5] I. S. Duff and J. R. Gilbert, *Maximum-weighted matchingand block pivoting for symmetric indefinite systems*, in Abstract Book of Householder Symposium XV, 2002, pp. 73–75.

[6] 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.

[7] 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.

[8] J. Edmonds and R. M. Karpl, *Theoretical improvements in algorithmic efficiency for network problems*, J. ACM, 19 (1972), pp. 248–264.
[9] 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.
[10] M. Hagemann and O. Schenk, *Weighted matchings for preconditioning symmetric indefinite linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 403–420.
[11] HSL, *A Collection of Fortran Codes for Large-Scale Scientific Computation*, http://www.hsl.rl.ac.uk/ (2013).
[12] H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quarterly, 2 (1955), pp. 83–97.
[13] 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, 1998, pp. 1–17.
[14] K. Mehlhorn and St. Näher, *The LEDA Platform of Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge, 1999.
[15] M. Olschowka and A. Neumaier, *A new pivoting strategy for Gaussian elimination*, Linear Algebra Appl., 240 (1996), pp. 131–151.
[16] O. Schenk, S. Röllin, and A. Gupta, *The effects of nonsymmetric matrix permutations and scalings in semiconductor device and circuit simulation*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, 23 (2004), pp. 400–411.
[17] O. Schenk, A. Wächter, and M. Hagemann, *Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale non-convex interior-point optimization*, Comput. Optim. Appl., 36 (2007), pp. 321–341.