NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.1978

On the use of suboptimal matchings for scaling and ordering sparse symmetric matrices

Jonathan Hogg*,† and Jennifer Scott

Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Oxford, Didcot OX11 0QX, Oxfordshire

SUMMARY

The use of matchings is a powerful technique for scaling and ordering sparse matrices prior to the solution of a linear system $Ax = b$. Traditional methods such as implemented by the HSL software package MC64 use the Hungarian algorithm to solve the maximum weight maximum cardinality matching problem. However, with advances in the algorithms and hardware used by direct methods for the parallelization of the factorization and solve phases, the serial Hungarian algorithm can represent an unacceptably large proportion of the total solution time for such solvers. Recently, auction algorithms and approximation algorithms have been suggested as alternatives for achieving near-optimal solutions for the maximum weight maximum cardinality matching problem. In this paper, the efficacy of auction and approximation algorithms as replacements for the Hungarian algorithm is assessed in the context of sparse symmetric direct solvers when used in problems arising from a range of practical applications. High-cardinality suboptimal matchings are shown to be as effective as optimal matchings for the purposes of scaling. However, matching-based ordering techniques require that matchings are much closer to optimality before they become effective. The auction algorithm is demonstrated to be capable of finding such matchings significantly faster than the Hungarian algorithm, but our $\frac{1}{2}$-approximation matching approach fails to consistently achieve a sufficient cardinality.

Received 29 January 2014; Revised 9 January 2015; Accepted 9 February 2015

KEY WORDS: approximation algorithm; auction algorithm; matching; sparse symmetric matrix; scaling; ordering

1. INTRODUCTION

Our aim is to efficiently solve the large sparse linear system

$$Ax = b.$$

Our main interest is the use of direct solvers when A is symmetric indefinite. In this case, it is necessary to incorporate numerical pivoting to maintain stability. This can mean that the pivot sequence chosen during the analyse phase on the basis of sparsity has to be modified as the numerical factorization proceeds. In particular, some pivots have to be delayed until they satisfy the stability criteria. Our recent studies comparing scaling algorithms [1, 2] demonstrate that the number of delayed pivots provides a good predictor for the effectiveness of a scaling at reducing the wall-clock time for the factorization. This is because the number of delayed pivots corresponds to the amount of additional work performed because of the requirements for numerical pivoting. Our work also demonstrates that, for tough indefinite problems, the widely used MC64 scaling algorithm [3] is particularly effective compared with other scalings and techniques tested.

*Correspondence to: Jonathan Hogg, Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Oxford, Didcot OX11 0QX, Oxfordshire.

†E-mail: jonathan.hogg@stfc.ac.uk

MC64 seeks to find an ordering such that the product of the entries on the diagonal of the reordered matrix is maximized. The problem is only well defined if a permutation exists such that the diagonal is zero-free; however, the degenerate case can also be treated; see for example our recent work [4]. Stated mathematically, given a sparse matrix $A = \{a_{ij}\}$, we associate a bipartite graph having vertex sets V_r, V_c corresponding to the rows and columns, and an edge $(i, j) \in E$ joining row i to column j if a_{ij} is nonzero. An edge subset $\mathcal{M} \subseteq E$ is called a matching if no two edges in $\mathcal{M}$ are incident to the same vertex. We seek a matching $\mathcal{M}$ of the row vertices to the column vertices such that the cardinality $|\mathcal{M}|$ is maximized and the product of the entries $|a_{ij}|$ for each edge in the matching is maximized. If σ is an indicator function such that $\sigma_{ij} = 1$ if edge $(i, j) \in \mathcal{M}$ and is zero otherwise, then we aim to solve the following problem:

$$\max \prod_{(i,j)\in E} |a_{ij}|\sigma_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in V_c} \sigma_{ij} = 1, \qquad \forall i \in V_r$$

$$\sum_{i\in V_r} \sigma_{ij} = 1, \qquad \forall j \in V_c$$

$$\sigma_{ij} \in \{0, 1\}.$$

The MC64 algorithm applies a transformation to a_{ij} to give an associated (positive) edge weight

$$w_{ij} = \log c_j - \log |a_{ij}|, \tag{2}$$

where $c_j = \max_i |a_{ij}|$. The maximization in (1) is then equivalent to

$$\min \sum_{(i,j)\in E} w_{ij}\sigma_{ij}. \tag{3}$$

By way of a sign change, this is classified as a maximum weight maximum cardinality matching problem (also known as an assignment problem). In MC64, this is solved using the Hungarian algorithm [5]. From standard theory, at optimality, the following conditions are satisfied for some row and column dual variables u and v:

$$w_{ij} - u_i - v_j = 0, \qquad \forall (i, j) \in \mathcal{M}, \tag{4}$$

$$w_{ij} - u_i - v_j \geqslant 0, \qquad \text{otherwise.}$$

The matching $\mathcal{M}$ provides a permutation that, in the unsymmetric case, can be used to achieve a zero-free diagonal with large entries. In the symmetric case, it can be used to permute large entries onto the subdiagonal [6–9]. The dual variables u and v can be used to calculate a scaling as follows. Define the diagonal scaling matrices D_r, D_c and S with diagonal entries

$$\begin{aligned} d_i^r &= \exp(u_i), \\ d_j^c &= \exp(v_j - c_j), \\ s_i &= \sqrt{d_i^r d_i^c}. \end{aligned}$$

Then, $D_r A D_c$ is such that the largest entry in each row and column is exactly one, and all other entries are less than or equal to one. If A is symmetric, SAS has the same property. The permutation and scaling can be used independently if required, and it is especially common in the symmetric case to use only the scaling because of the unsymmetric nature of the permutation. However, more recently, techniques have been developed to permute the large entries onto the subdiagonal [6, 7]. We consider both approaches in this paper.

DOI: 10.1002/nla

There are two potential problems with MC64: (i) the runtime is hard to predict and can vary significantly when the data are permuted; and (ii) an application of MC64 can represent a significant fraction of the total factorization time when using a direct solver, particularly when the solver is run in parallel (see Table VI in Section 4). The latter point is compounded by Amdahl's law, as MC64 is a serial code while the factorization obtains good parallel speedups on a modest number of cores. The main issue lies with the Hungarian algorithm that MC64 uses to solve the assignment problem. This seeks optimal augmenting paths through the matrix from an unmatched row to an unmatched column. In those cases where performance is poor, it is because of the need to scan a significant portion of the entire matrix while proving optimality for each augmenting path.

We note that it may be possible to parallelize the Hungarian algorithm using similar techniques to those used for the unweighted case [10, 11]. However, we expect the speedups to be significantly more limited because at each stage, optimal independent augmenting paths must be found, whereas in the unweighted case, any augmenting path will do.

In this paper, we relax the requirement for a maximum cardinality matching to allow us to use algorithms that deliver near-optimal results in weight and cardinality. The solution hence does not provide the zero-free diagonal often desired by unsymmetric solvers, but does allow the majority of large entries to be permuted to the subdiagonal for symmetric matrices and, as we shall demonstrate, still provides a high-quality scaling for most of our test matrices, which are taken from practical applications.

The main contribution of this paper is a comparison of the performance and effectiveness of two alternative algorithms for the relaxed maximum weight maximum cardinality matching problem with that of the MC64 implementation of the Hungarian algorithm when the resulting scaling is used prior to the factorization of sparse symmetric matrices. We do not consider unsymmetric factorizations in this paper as the application of the matching and subsequent factorization is substantially different to the symmetric case. These alternatives are an auction algorithm and a $\frac{1}{2}$-approximation algorithm. Both solve the problem approximately while claiming to offer significantly better parallel speedups than the Hungarian algorithm for large problems [12, 13]. We assess performance in terms of time to find the matching and their effectiveness when used as scaling and/or ordering heuristics for a sparse direct symmetric linear solver.

The remainder of this paper is laid out as follows. In Section 2, we describe the auction algorithm and associated work; both serial and parallel versions are discussed. Then, in Section 3, we briefly describe the approximation algorithm. Section 4 provides a comparison of the effectiveness of these algorithms, both in terms of performance and numerical improvement, to the scaled matrix; comparisons are made with the Hungarian algorithm. Finally, in Section 5, our conclusions are presented.

2. THE AUCTION ALGORITHM

The auction algorithm for the maximum weight matching problem was first proposed by Bertsekas [14] and since then has been studied in a number of papers, including [15–17]. Most recently, Sathe *et al.* [13] showed that the algorithm can quickly find high-quality matchings and is readily parallelizable.

The auction algorithm solves the following maximum weight matching problem:

$$\max \sum_{(i,j)\in E} w_{ij}\sigma_{ij} \tag{5}$$

$$\text{s.t.} \quad \sum_{j\in V_c} \sigma_{ij} \leqslant 1, \qquad \forall i \in V_r$$

$$\sum_{i\in V_r} \sigma_{ij} \leqslant 1, \qquad \forall j \in V_c$$

$$\sigma_{ij} \in \{0, 1\}.$$

DOI: 10.1002/nla

To transform (1) to (5), in place of (2), we use the related transformation

$$w_{ij} = \alpha + \log |a_{ij}| + (\alpha - c_j), \tag{6}$$

where $c_j = \max_i \log |a_{ij}|$ and $\alpha = \max_{i,j}\{c_j - \log |a_{ij}|\}$. The transformation $\log |a_{ij}| + (\alpha - c_j)$ is sufficient to transform the maximum product into a scaled maximum sum over positive weights. The extra α term transforms the objective from a maximum weight to maximum weight maximum cardinality because α is greater than each individual $\log |a_{ij}| + (\alpha - c_j)$ term; only a maximum cardinality solution can be optimal (this trick has been used by other authors previously, e.g. in LEDA [18]). A corresponding unsymmetric scaling of A is then given by

$$\begin{aligned} d_i^r &= \exp(\alpha - u_i), \\ d_j^c &= \exp(\alpha - v_j - c_j), \end{aligned}$$

which can again be symmetrized using

$$s_i = \sqrt{d_i^r d_i^c}.$$

For each nonzero entry a_{ij} of A, $w_{ij} - u_i$ is the increase in the objective (3) obtained by augmenting $\mathcal{M}$ with (i, j), displacing any edge currently in $\mathcal{M}$ that contains row i or column j. The auction algorithm starts with the row dual variables, $u = \{u_i\}$, initialised to zero and proceeds by scanning each unmatched column j to find the row index i such that

$$i = \arg\max_k \left\{w_{kj} - u_k\right\}.$$

If $w_{ij} - u_i > 0$, column j 'bids' for row i. The highest bid for row i wins; say in column j_1, the matching $\mathcal{M}$ is augmented by adding the edge (i, j_1), and any column previously matched with row i is returned to the pool of unmatched columns. The dual variable u_i is updated to be the cost of using the second best row, that is, u_i is the (first-order) reduction in the objective if j_1 was not matched to i. For this reason, the dual vector u is also referred to in some contexts as the vector of 'reduced costs'. By adding $\epsilon > 0$ to u_i, a minimum increase in the objective can optionally be required. This accelerates convergence of the algorithm by ignoring opportunities for trivial improvement. ϵ is chosen to be small but much larger than machine precision, and it is increased as the algorithm proceeds.

Once the matching $\mathcal{M}$ and row dual variables u have been computed, the column dual variables v may be obtained directly from (4).

We have implemented both the serial and OpenMP versions of the auction algorithm, which we outline as Algorithms 1 and 2, respectively. In Algorithm 1, $\mathcal{U}$ is the set of columns that have been found to be unmatchable. In Algorithm 2, we use pseudo-code modelled after OpenMP: **DEFAULT(private)** and **SHARED** specify data locality, and **BARRIER** indicates that no thread continues past that point until all threads have reached it.

For the serial algorithm, the (average) number of iterations and cost per iteration is reduced by treating every bid as immediately winning. This reduces the cost per iteration, as there is no longer a need to determine the highest bid (each bid wins) and the data needed for the resulting update are already in cache. Further, if a bid by column j for row i wins immediately, any existing k such that $(i, k) \in \mathcal{M}$ becomes available for rematching in the current iteration. However, in the parallel version, the work must be split into separate bid generation and reconciliation phases, as bids must now be communicated between threads. As the process is memory bound, this additional phase essentially doubles the time, so significant speedup is required for the parallel code to outperform the serial code. This is illustrated in Section 4.1.

The termination conditions for both algorithms are the same, and the basis for these is illustrated in Figures 1 and 2. These show the convergence of the serial auction algorithm for two symmetric problems taken from our test set (see Section 4 for details) that are chosen to demonstrate typical behaviour (one converges almost immediately in fewer than 10 iterations while the other takes over 200). We define the effectiveness of a matching $\mathcal{M}$ as the reduction in the number of delayed pivots

DOI: 10.1002/nla

Algorithm 1 Serial auction algorithm

Input: Size n, positive weights w_{ij}, iteration limit *maxitr*
Output: Matching $\mathcal{M}$, dual variables u
Initialise: $\mathcal{M} = \phi; \mathcal{U} = \phi; u = 0; \epsilon = 0.01$
for $itr = 1, maxitr$ **do**
  if (terminate()) **exit**
  $\epsilon = \min(1.0, \epsilon + 1/(n+1))$
  for each unmatched column $j \notin \mathcal{U}$ **do**
    Find $i = \arg\max_k \{w_{kj} - u_k\}$, $pval = w_{ij} - u_i$ and $qval = \max_{k \neq i} \{w_{kj} - u_k\}$
    if $(pval > 0)$ **then**
      ! Bid for row i and win immediately
      $u_i = u_i + pval - qval + \epsilon$
      Add (i, j) to $\mathcal{M}$
      if ($(i,k) \in \mathcal{M}$ for some k) add k to $\mathcal{U}$ and remove (i,k) from $\mathcal{M}$
    else
      ! No bid is worthwhile
      Add j to $\mathcal{U}$
    end if
  end for
end for

terminate(): *! Returns true if algorithm should terminate*
if $(|\mathcal{M}| = n)$ **return** true
if ($|\mathcal{M}|$ unchanged for 10 iterations **and** $|\mathcal{M}|/(n - |\mathcal{U}|) > 0.9$) **return** true
if ($|\mathcal{M}|$ unchanged for 100 iterations) **return** true
return false

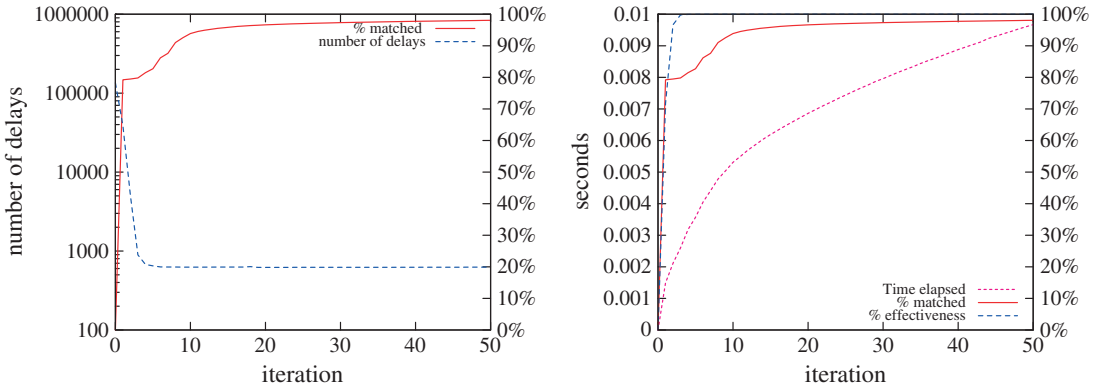

Figure 1. Convergence behaviour of the serial auction algorithm on the Schenk_IBMNA/c-62 matrix.

compared with the reduction for an optimal matching $\mathcal{M}^*$ calculated using `MC64`. That is, we use the following formula:

$$\%\,\text{effectiveness} = 100 \times \frac{\text{ndelay}_{\phi} - \text{ndelay}_{\mathcal{M}}}{\text{ndelay}_{\phi} - \text{ndelay}_{\mathcal{M}^*}},$$

where ndelay is the number of delayed pivots and ϕ is the empty set and denotes no scaling. The figures demonstrate that $|\mathcal{M}|$ and the effectiveness of the matching do not correlate well. However, in all our tests, we observed that a matching of high cardinality was sufficient to achieve high

DOI: 10.1002/nla

Algorithm 2 Parallel auction algorithm

Input: Size n, positive weights w_{ij}, iteration limit *maxitr*
Output: Matching $\mathcal{M}$, dual variables u
Running on P **threads, DEFAULT(private), SHARED**(n, w_{ij}, *maxitr*, $\mathcal{M}$)
Initialise: $\mathcal{M} = \phi$; $u = 0$; $\epsilon = 0.01$
Partition the columns equally between the threads
for $itr = 1$, *maxitr* **do**
  if (terminate()) **exit**
  $\epsilon = \min(1.0, \epsilon + 1/(n+1))$
  generate_bids()
  BARRIER (wait for all bids to be made)
  determine_winners()
  BARRIER (wait for winners to be found)
  Reassign columns so that each thread has approximately $(n - |\mathcal{M}|)/P$ columns
end for

generate_bids():
for each unmatched column j owned by this thread **do**
  Find $i = \arg\max_k\{w_{kj} - u_k\}$, $pval = w_{ij} - u_i$ and $qval = \max_{k \neq i}\{w_{kj} - u_k\}$
  if ($pval > 0$) **then**
    $u_i = u_i + pval - qval + \epsilon$
    Delete any existing bid by this thread for i.
    Record bid (i, j) and u_i.
  end if
end for

determine_winners():
for all rows i **do**
  Find highest bid (i, j) with value *pval* among all threads
  Add (i, j) to $\mathcal{M}$
  if($(i, k) \in \mathcal{M}$ for some k) mark k as unmatched and remove (i, k) from $\mathcal{M}$
  Update local $u_i = pval$
end for

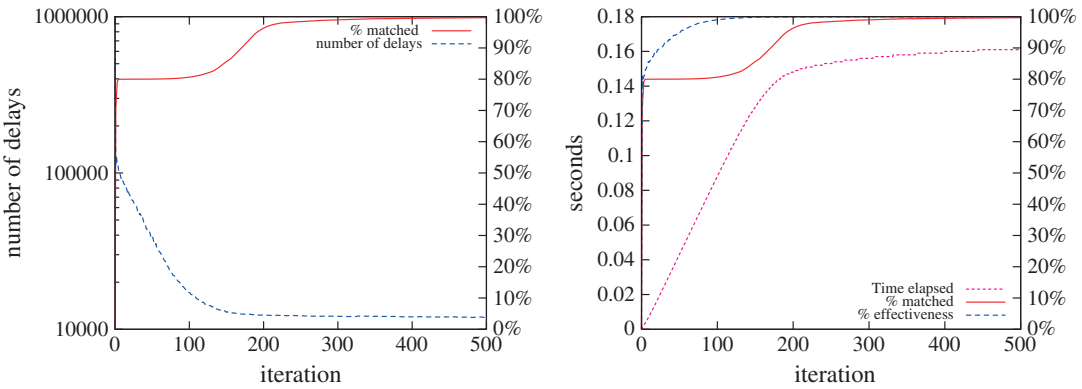

Figure 2. Convergence behaviour of the serial auction algorithm on the GHS_indef/ncvxqp5 matrix.

DOI: 10.1002/nla

effectiveness (but we could stop much earlier in some cases). With the further observation that later iterations are much cheaper than earlier iterations as they involve many fewer unmatched columns, convergence to a high-cardinality matching is a good stopping criterion.

Our preliminary experiments showed the quality of the matching to be sensitive to the choice of ϵ at each iteration. Sathe *et al.* [13] describe a strategy for choosing ϵ, and, as we also found this to be effective, we employ their approach (except that, in place of initializing ϵ to $16/(n+1)$, we use the constant initial value of 0.01).

The question arises as to how unmatched rows and columns should be scaled. The transform (6) can lead to large entries in D_c and small entries in D_r that (by construction) cancel to give a moderate scaling of A for entries in matched rows and columns. However, for unmatched rows and columns, we cannot simply choose $u_i = 0$ or $v_j = 0$ without taking into account the w_{ij} transformation: if the unmatched row contains an entry in a matched column (or vice versa), it ends up very poorly scaled. Instead, we consider values in w_{ij} space and transform back to a_{ij} space to obtain the relevant scaling. We found that most reasonable values for such dual variables work, but for simplicity, our code initialises $u_i = 0$ and $v_j = \max_i w_{ij}$ ($u_i = 0, v_j = 0$ performed considerably less well in our tests).

Finally, we observe that, for an unmatched columns j, the $pval = w_{ij} - u_i$ value calculated most recently makes a good guess for v_j, as it corresponds to the value of the dual variable for a recent trial matching. As each (non-empty) column has such a *pval* calculated at least once during the execution of the algorithm, these values are always available. However, storing this value creates additional memory traffic that may not be desirable, particularly in the parallel case where false sharing may be an issue. Further, no similar value is available for unmatched rows. For these reasons, we avoid this approach.

3. THE APPROXIMATION ALGORITHM

We start with some definitions that we require in our description of the approximation algorithm. We assume that all edge weights w_{ij} are positive and define $W(\mathcal{M}) = \sum_{(i,j)\in\mathcal{M}} w_{ij}$ as the total weight of a matching $\mathcal{M}$ for a graph $\mathcal{G}$. Let $\mathcal{M}^*$ be an optimal matching; then, an α-approximation matching algorithm is defined to be a matching algorithm that guarantees to find $\mathcal{M}$ such that $W(\mathcal{M}) \geqslant \alpha W(\mathcal{M}^*)$. An edge (i, j) of $\mathcal{G}$ with weight w_{ij} is defined to be locally dominant if $\arg\max_k \{w_{kj}\} = \arg\max_k \{w_{ki}\} = w_{ij}$.

The greedy approach of Avis [19] provides a simple $\frac{1}{2}$-approximation algorithm. This matches the heaviest edges in decreasing order if they are locally dominant (this is roughly equivalent to a single round of the auction algorithm). In this paper, we use the more advanced parallel implementation of Halappanavar *et al.* [12]. Rather than the sorting-based approach of Avis, a queue-based mechanism is used, as originally proposed by Preis [20]. The central concept is that, at each iteration, locally dominant edges are added to the matching; the matched edges and their vertices are removed, and the resulting reduced graph is considered at the next iteration.

The algorithm has two phases. In the first, a list of the locally dominant edges in $\mathcal{G}$ is made. This is performed in parallel by passing through the graph data twice. On the first pass, for each vertex j, the neighbour p_j that maximises w_{kj} is found (ties are broken by taking the lowest such p_j). The edge (j, p_j) is held as the candidate locally dominant edge for vertex j. A second pass confirms whether a candidate edge is a locally dominant edge by checking if $p_{p_j} = j$. Each locally dominant edge (j, p_j) is added to the matching, and the vertex j is added to the list $\mathcal{Q}$ of vertices to be removed from $\mathcal{G}$ for the next iteration. Observe that $k = p_j$ will be added to this list when vertex k is processed.

The second phase of the algorithm consists of a number of iterations, each of which removes vertices from $\mathcal{G}$ and, for each remaining vertex i that is a neighbour of a vertex in $\mathcal{Q}$, updates its candidate locally dominant edge. The list of vertices for removal can be iterated over in parallel as long as updates to the list of candidate edges and additions to the vertex removal list $\mathcal{Q}'$ for the next iteration are performed atomically. The unmatched neighbours of each vertex j that is to be removed must be checked. Specifically, any unmatched neighbour i for which (i, j) was the candidate edge must have its candidate edge updated to the edge (i, p_i), where p_i is the unmatched neighbour

DOI: 10.1002/nla

Table I. Test sets used for testing. $nz(A)$ denotes the number of entries in the lower triangular part of A; $nz(L)$ and $nflops$ denote the number of entries and floating-point operations, respectively, returned by the analyse phase of the `HSL_MA97` solver.

Identifier	n	$nz(A)$	$nz(L)$	$nflops$	Description/ Application
Test set 1					
Schenk_IBMNA/c-54	31 793	385 987	1.0374×10^6	7.6222×10^7	Nonlinear optimization
Boeing/pcrystk02	13 965	968 583	4.3969×10^6	1.9128×10^9	Crystal vibration
HB/bcsstk30	28 924	1 036 208	3.9946×10^6	9.3470×10^8	Offshore generator platform
GHS_indef/boyd1	93 279	1 211 231	6.5262×10^5	4.6722×10^6	Convex QP
Rothberg/gearbox	153 746	9 080 404	3.8829×10^7	2.1001×10^{10}	Aircraft flap actuator
Gupta/gupta3	16 783	9 323 427	6.3516×10^6	3.1067×10^9	Linear programming
Andrianov/mip1	66 463	10 352 819	4.5317×10^7	1.4552×10^{11}	Mixed integer programming
DNVS/fullb	199 187	11 708 077	7.6518×10^7	1.0081×10^{11}	Full-breadth barge
DNVS/troll	213 453	11 985 111	6.6466×10^7	5.6414×10^{10}	Structural problem
Chen/pkustk14	151 926	14 836 504	1.0945×10^8	1.4796×10^{11}	Tall building
Test set 2					
GHS_indef/copter2	55 476	759 952	1.0444×10^7	5.4949×10^9	CFD helicopter rotor blade
Cunningham/qa8fk	66 127	1 660 579	2.4259×10^7	2.1322×10^{10}	3D acoustic FE stiffness matrix
Boeing/crystk03	24 696	1 751 178	9.8413×10^6	5.7087×10^9	Crystal vibration
Lin/Lin	256 000	1 766 400	1.1359×10^8	2.7918×10^{11}	Structural eigenvalue problem
Boeing/bcsstk39	46 772	2 060 662	7.0169×10^6	1.6613×10^9	Rocket booster
Boeing/pct20stif	52 329	2 698 463	1.1952×10^7	9.1960×10^9	Engine block
Oberwolfach/filter3D	106 437	2 707 179	2.0099×10^7	7.6986×10^9	3D heat transfer PDE
Oberwolfach/t3dh	79 171	4 352 105	4.8137×10^7	6.9077×10^{10}	Micropyros thruster
Koutsovasilis/F2	71 505	5 294 285	2.1290×10^7	1.1450×10^{10}	Piston rod
PARSEC/Ge99H100	112 985	8 451 395	6.5419×10^8	7.0120×10^{12}	Density function theory
Test set 3					
GHS_indef/ncvxqp1	12 111	73 963	1.6839×10^6	7.2793×10^8	Non-convex QP
GHS_indef/cvxqp3	17 500	122 462	3.1398×10^6	1.7670×10^9	Convex QP
GHS_indef/ncvxqp5	62 500	424 966	1.2052×10^7	9.7223×10^9	Non-convex QP
GHS_indef/ncvxqp3	75 000	499964	1.9007×10^7	2.0692×10^{10}	Non-convex QP
GHS_indef/stokes128	49 666	558 594	2.9813×10^6	3.6881×10^8	FE model Stokes problem
Schenk_IBMNA/c-62	41 731	559 341	8.4562×10^6	8.0164×10^9	Optimization problem
Schenk_IBMNA/c-64	51 035	707 985	1.6971×10^6	1.3801×10^8	Optimization problem
GHS_indef/boyd2	466 316	1 500 397	2.5854×10^6	1.5582×10^7	Optimization problem
Test set 4					
GHS_indef/bratu3d	27 792	173 796	6.2769×10^6	4.4174×10^9	Optimization
GHS_indef/cont-201	80 595	438 795	4.7815×10^6	8.6513×10^8	Convex QP
GHS_indef/ncvxqp7	87 500	574 962	2.4731×10^7	3.0939×10^{10}	Non-convex QP
GHS_indef/cont-300	180 895	988 195	1.1744×10^7	2.9559×10^9	Convex QP
GHS_indef/darcy003	389 874	2 101 242	8.1587×10^6	5.5664×10^8	Mixed FE model Darcy's equation
TSOPF/TSOPF_FS_b300_c2	56 814	8 767 466	2.1433×10^7	8.9629×10^9	Optimal power flow

DOI: 10.1002/nla

of i that maximises w_{ki}. If edge (i, p_i) is locally optimal in the reduced graph, it is added to the matching, and vertices i and p_i are included in the removal list that is to be used in the next iteration.

An algorithm outlined together with a simple example to illustrate the algorithm are given in [21].

Recall that our interest is in the problem (1), but the approximation algorithm addresses the maximum weight maximum cardinality problem. Applying the transformation

$$w_{ij} = \log |a_{ij}| + (\alpha - c_j),$$

where $\alpha = \max_{i,j}\{c_j - \log |a_{ij}|\}$ and $c_j = \max_i \log |a_{ij}|$, we obtain $w_{ij} > 0$, and the final matching provides an approximate solution to (1). Note that the extra α term present in (6) is omitted as

Table II. Description of machine used for numerical experiments.

Processor	2 × Intel Xeon E5-2687W
Physical cores	16
Memory	64 GB
Compiler	ifort 12.1.0
BLAS	MKL 10.3 update 6
L1/L2 cache (per core)	32 KB / 256 KB
L3 cache (shared)	20 MB
Compiler flags	`ifort -O3 -xAVX -no-prec-div -ip -openmp`

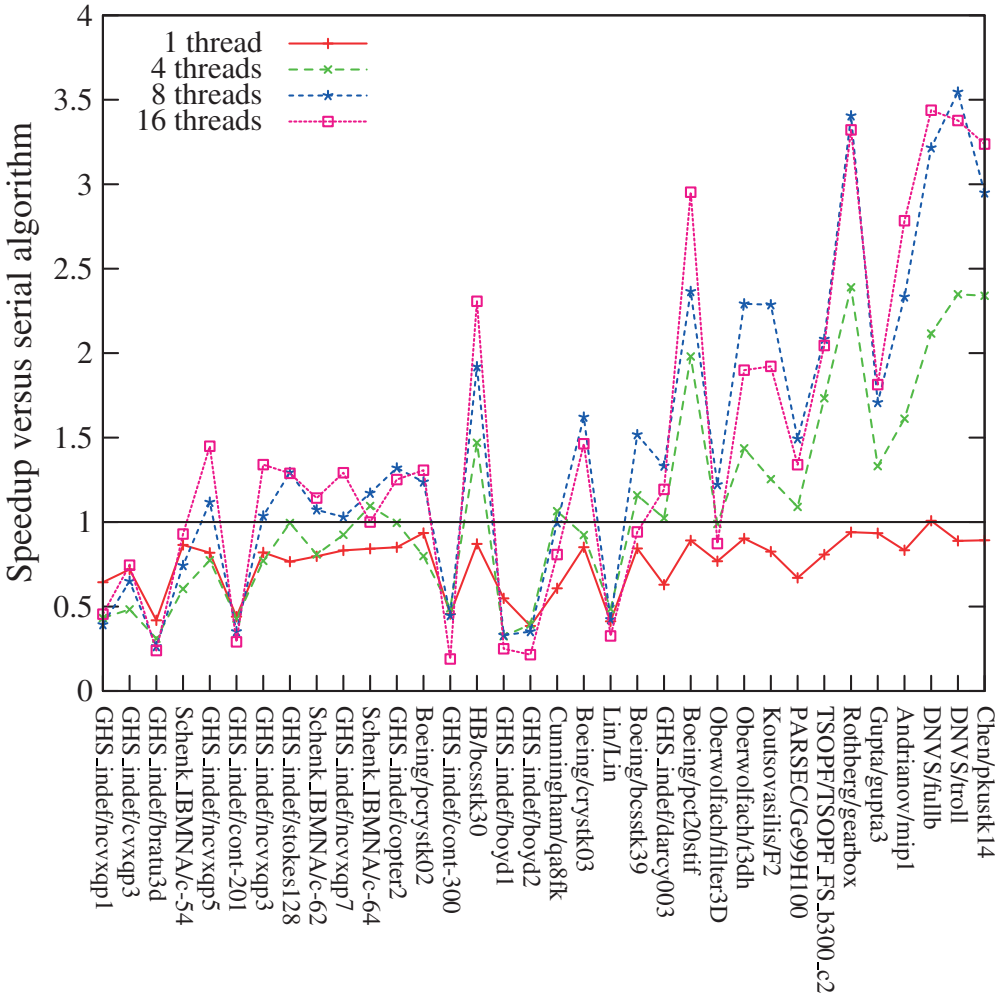

Figure 3. Speedup of the parallel auction algorithm against the serial auction algorithm. Matrices from all four test sets are ordered by increasing number of entries in A.

DOI: 10.1002/nla

the algorithm works with relative rather than absolute edge weights: its inclusion would be pointless. As the approximation algorithm does not use dual variables, we must define u and v. In our implementation, we define

$$u_i = 0 \qquad \forall\, i, \tag{7}$$

$$v_j = \begin{cases} w_{ij}, & (i,j) \in \mathcal{M}, \\ c_j, & \text{otherwise.} \end{cases} \tag{8}$$

This guarantees the equality condition $w_{ij} - u_i - v_j = 0$ for edges in $\mathcal{M}$. The choice $v_j = c_j$ for unmatched columns ensures that the largest entry in the column is scaled towards 1, and that the scaling is appropriate after the w_{ij} transformation is reversed. Note that the choice (7)–(8) is not unique.

4. COMPUTATIONAL EXPERIMENTS

For the purposes of our experiments, we use four sets of symmetric indefinite test problems drawn from the University of Florida Sparse Matrix Collection [22] and detailed in Table I. Test Sets 1 and 2 are matrices that do not significantly benefit from an `MC64` scaling compared with no scaling or the application of a cheap norm equilibration algorithm. The purpose of these sets is to assess

Table III. Cardinality of the matching obtained using each algorithm as a percentage of the entries matched. Cardinalities less than 99% are in bold. Numbers in brackets give deficiencies.

Problem	Hungarian	sAuction	Approx
Schenk_IBMNA/c-54	100	**98.89 (353)**	99.33 (213)
Boeing/pcrystk02	100	99.68 (44)	**98.22 (249)**
HB/bcsstk30	100	99.57 (124)	**97.22 (803)**
GHS_indef/boyd1	100	99.99 (8)	99.99 (11)
Rothberg/gearbox	100	99.88 (180)	**97.85 (3303)**
Gupta/gupta3	100	99.79 (36)	**96.57 (576)**
Andrianov/mip1	100	99.60 (263)	**97.01 (1985)**
DNVS/fullb	100	99.89 (218)	**97.63 (4726)**
DNVS/troll	100	99.89 (242)	**97.85 (4593)**
Chen/pkustk14	100	99.84 (249)	**98.33 (2533)**
GHS_indef/copter2	100	99.78 (123)	**94.79 (2890)**
Cunningham/qa8fk	100	100	100
Boeing/crystk03	100	100	100
Lin/Lin	100	100	100
Boeing/bcsstk39	100	100	99.54 (214)
Boeing/pct20stif	100	99.61 (202)	**97.33 (1399)**
Oberwolfach/filter3D	100	100	100
Oberwolfach/t3dh	100	100	100
Koutsovasilis/F2	100	100	100
PARSEC/Ge99H100	100	100	100
GHS_indef/ncvxqp1	100	**96.28 (450)**	**61.08 (4714)**
GHS_indef/cvxqp3	100	**98.06 (340)**	**57.14 (7500)**
GHS_indef/ncvxqp5	100	99.84 (100)	**83.50 (10315)**
GHS_indef/ncvxqp3	100	**96.00 (3000)**	**68.84 (23367)**
GHS_indef/stokes128	100	99.87 (67)	**67.01 (16384)**
Schenk_IBMNA/c-62	100	99.68 (133)	**84.42 (6502)**
Schenk_IBMNA/c-64	100	99.04 (488)	**93.78 (3173)**
GHS_indef/boyd2	100	100	**93.38 (30857)**
GHS_indef/bratu3d	100	100	**94.53 (1519)**
GHS_indef/cont-201	100	100	99.75 (199)
GHS_indef/ncvxqp7	100	**98.06 (1700)**	**61.02 (34109)**
GHS_indef/cont-300	100	100	99.83 (299)
GHS_indef/darcy003	100	99.98 (63)	99.90 (383)
TSOPF/TSOPF_FS_b300_c2	100	99.93 (38)	**75.03 (14187)**

DOI: 10.1002/nla

the cost of applying a scaling algorithm when scaling is not actually needed. For the problems in test set 1, the time to run MC64 is high, while for those in test set 2, MC64 represents a much smaller overhead in the solver time. Test sets 3 and 4 are drawn from our recent paper on pivoting techniques for difficult problems [2]. Test set 3 is a set of problems for which using the MC64 scaling is sufficient to reduce the number of delayed pivots to reasonable levels, while test set 4 comprises those problems that require a matching-based ordering and scaling to achieve this (further details on matching-based orderings are given in Section 4.3).

All our tests are performed on the 16-core machine detailed in Table II. All times and results for the Hungarian algorithm are obtained using HSL_MC64 version 2.4.0; the sparse direct solver used is HSL_MA97 version 2.2.0 [23, 24]. Both HSL codes are run with default settings, except where otherwise stated. In particular, this means that a heuristic choice is made between approximate minimum degree and nested dissection orderings, and the pivot threshold parameter is 0.01. Results for the $\frac{1}{2}$-approximation algorithm were obtained with MATCHBOX software [25]. We use the letters OOM to indicate a problem ran out of memory during the factorization phase of HSL_MA97 because of the generation of too many delayed pivots.

4.1. Scalability

Figure 3 shows the speedup of our implementation of the parallel auction algorithm against our implementation of the serial auction algorithm. The problems in the four test sets have been amal-

Table IV. Number of delayed pivots reported by HSL_MA97 with different scalings.

Problem	None	Hungarian	sAuction	Approx
Schenk_IBMNA/c-54	6355	1281	2566	11 203
Boeing/pcrystk02	11	11	11	11
HB/bcsstk30	16	16	16	16
GHS_indef/boyd1	OOM	0	0	43 671
Rothberg/gearbox	102	101	103	110
Gupta/gupta3	41	33	34	36
Andrianov/mip1	122	50	51	100
DNVS/fullb	145	145	145	150
DNVS/troll	150	147	146	167
Chen/pkustk14	102	102	101	113
GHS_indef/copter2	87	86	72	80
Cunningham/qa8fk	0	0	0	0
Boeing/crystk03	0	0	0	0
Lin/Lin	0	0	0	0
Boeing/bcsstk39	0	0	0	32
Boeing/pct20stif	43	40	41	40
Oberwolfach/filter3D	0	0	0	0
Oberwolfach/t3dh	0	0	0	0
Koutsovasilis/F2	0	0	0	0
PARSEC/Ge99H100	3	3	3	1
GHS_indef/ncvxqp1	124 018	10 303	31 462	76 197
GHS_indef/cvxqp3	312 033	26 039	26 058	120 608
GHS_indef/ncvxqp5	544 291	11 858	11 944	522 545
GHS_indef/ncvxqp3	1 446 194	65 161	66 027	OOM
GHS_indef/stokes128	30 509	5502	5502	5502
Schenk_IBMNA/c-62	135 154	594	669	180 979
Schenk_IBMNA/c-64	32 356	574	570	120 103
GHS_indef/boyd2	27 077	0	0	39 339
GHS_indef/bratu3d	59 569	59 657	59 590	59 650
GHS_indef/cont-201	88 299	88 276	88 276	88 284
GHS_indef/ncvxqp7	1 697 334	272 146	273 327	1 673 909
GHS_indef/cont-300	148 526	148 509	148 509	148 512
GHS_indef/darcy003	44 900	44 900	44 900	44 900
TSOPF/TSOPF_FS_b300_c2	100 652	45 306	46 175	97 031

DOI: 10.1002/nla

gamated into a single set and then rearranged in the order of increasing number of entries in A. It is clear that the matrix must have a large number of entries before parallelization is worthwhile (a significant speedup is required to overcome the overhead of the separate bid generation and reconciliation phases). Further analysis of the results shows that the quality of the scalings computed using the serial and parallel versions are comparable (see [21] for detailed results). Thus, with our current implementation, we only recommend the parallel algorithm if there are more than 2×10^6 entries in A.

We found that no appreciable parallel speedup was achieved by running the approximation algorithm in parallel (however, slowdown was observed in the smallest problems).

On the basis of these findings, in the rest of this section, we present results for the serial implementations only.

4.2. Effectiveness of algorithms: scaling only

Table III provides results on the quality of the matching achieved by each of the matching algorithms in terms of cardinality, while Table IV measures the effectiveness of the associated scaling by counting the number of delayed pivots when the orderings are run with the `HSL_MA97` solver. Table V compares the runtime of each algorithm to achieve this.

Table V. Time (in seconds) to compute different scalings and the `HSL_MA97` time to compute the factorization on 16 cores.

	Scaling			Factor			
	Hungarian	sAuction	Approx	None	Hungarian	sAuction	Approx
Schenk_IBMNA/c-54	0.20	0.01	0.00	0.09	0.06	0.05	0.09
Boeing/pcrystk02	0.16	0.01	0.00	0.07	0.07	0.05	0.07
HB/bcsstk30	0.70	0.03	0.01	0.12	0.12	0.11	0.12
GHS_indef/boyd1	1.85	0.00	0.00	OOM	0.06	0.06	145
Rothberg/gearbox	1.96	0.19	0.04	0.42	0.42	0.40	0.42
Gupta/gupta3	1.61	0.06	0.03	0.39	0.37	0.35	0.38
Andrianov/mip1	4.71	0.25	0.03	2.34	2.28	2.26	2.33
DNVS/fullb	2.27	0.27	0.05	1.82	1.87	1.78	1.82
DNVS/troll	2.05	0.26	0.05	0.68	0.68	0.65	0.67
Chen/pkustk14	7.61	0.25	0.06	1.61	1.59	1.57	1.60
GHS_indef/copter2	0.06	0.02	0.01	0.15	0.14	0.14	0.14
Cunningham/qa8fk	0.00	0.00	0.00	0.36	0.36	0.34	0.35
Boeing/crystk03	0.00	0.00	0.00	0.14	0.14	0.12	0.13
Lin/Lin	0.01	0.00	0.01	4.57	4.64	4.44	4.63
Boeing/bcsstk39	0.01	0.00	0.01	0.17	0.16	0.15	0.16
Boeing/pct20stif	0.69	0.05	0.01	0.29	0.29	0.26	0.28
Oberwolfach/filter3D	0.01	0.01	0.01	0.14	0.14	0.13	0.14
Oberwolfach/t3dh	0.01	0.01	0.01	0.83	0.80	0.80	0.82
Koutsovasilis/F2	0.01	0.01	0.01	0.62	0.63	0.59	0.63
PARSEC/Ge99H100	0.02	0.01	0.02	259	262	259	261
GHS_indef/ncvxqp1	0.09	0.00	0.00	2.84	0.17	0.37	0.97
GHS_indef/cvxqp3	0.17	0.02	0.00	18.2	0.43	0.39	2.15
GHS_indef/ncvxqp5	0.41	0.17	0.00	69.1	0.77	0.71	90.5
GHS_indef/ncvxqp3	3.76	0.20	0.00	329	2.48	2.53	OOM
GHS_indef/stokes128	0.05	0.01	0.00	0.08	0.05	0.03	0.05
Schenk_IBMNA/c-62	0.05	0.02	0.00	6.85	0.44	0.39	17.0
Schenk_IBMNA/c-64	0.16	0.02	0.01	1.02	0.05	0.04	16.1
GHS_indef/boyd2	0.03	0.01	0.01	15.5	0.09	0.09	57.6
GHS_indef/bratu3d	0.00	0.00	0.00	0.86	0.86	0.83	0.86
GHS_indef/cont-201	0.00	0.00	0.00	0.19	0.19	0.16	0.16
GHS_indef/ncvxqp7	2.23	0.21	0.00	146	14.4	14.4	174
GHS_indef/cont-300	0.01	0.00	0.01	0.31	0.31	0.30	0.31
GHS_indef/darcy003	0.18	0.26	0.02	0.10	0.10	0.08	0.11
TSOPF/TSOPF_FS_b300_c2	0.23	0.23	0.02	1.62	0.48	0.39	1.54

DOI: 10.1002/nla

Table VI. Percentage of the total factorization time spent in the scaling algorithm on 16 cores.

Problem	Hungarian	sAuction	Approx
Schenk_IBMNA/c-54	76.1	10.4	1.7
Boeing/pcrystk02	69.0	12.4	5.9
HB/bcsstk30	85.4	19.3	6.6
GHS_indef/boyd1	96.9	5.4	<0.1
Rothberg/gearbox	82.2	32.0	8.8
Gupta/gupta3	81.2	13.7	6.8
Andrianov/mip1	67.4	9.8	1.4
DNVS/fullb	54.9	13.3	3.0
DNVS/troll	75.2	28.2	7.6
Chen/pkustk14	82.7	13.6	3.5
GHS_indef/copter2	30.4	14.9	3.9
Cunningham/qa8fk	1.0	0.7	1.3
Boeing/crystk03	2.2	2.0	3.1
Lin/Lin	0.2	0.1	0.2
Boeing/bcsstk39	3.5	2.8	3.9
Boeing/pct20stif	70.6	16.0	4.3
Oberwolfach/filter3D	7.2	6.7	6.1
Oberwolfach/t3dh	1.5	1.3	1.3
Koutsovasilis/F2	2.1	2.0	2.0
PARSEC/Ge99H100	<0.1	<0.1	<0.1
GHS_indef/ncvxqp1	33.7	1.0	<0.1
GHS_indef/cvxqp3	28.9	5.7	<0.1
GHS_indef/ncvxqp5	34.9	19.4	<0.1
GHS_indef/ncvxqp3	60.3	7.4	<0.1
GHS_indef/stokes128	53.0	22.6	4.0
Schenk_IBMNA/c-62	9.5	4.3	<0.1
Schenk_IBMNA/c-64	75.1	33.3	<0.1
GHS_indef/boyd2	24.7	10.0	<0.1
GHS_indef/bratu3d	0.1	0.1	0.1
GHS_indef/cont-201	1.1	0.8	1.4
GHS_indef/ncvxqp7	13.4	1.4	<0.1
GHS_indef/cont-300	1.7	1.0	1.7
GHS_indef/darcy003	64.2	76.8	13.2
TSOPF/TSOPF_FS_b300_c2	32.9	36.9	1.2

These tables show that while the approximation algorithm is the fastest, it fails to provide an alternative to the Hungarian algorithm, both in terms of finding a high-cardinality matching and reducing the number of delayed pivots. On the other hand, both our serial and parallel auction codes lead to a similar number of delayed pivots as for the Hungarian algorithm on all but one problem (GHS_indef/ncvxqp1), where they perform slightly worse.

The ncvxqp1 discrepancy is an example where our stopping conditions for the auction algorithm cause it to terminate with a 96.3% match after 101 iterations, taking approximately 0.004 s. If we instead run for 383 iterations (which takes 0.007 s), we achieve a 96.6% match resulting in only 10 986 delayed pivots, which is comparable to the Hungarian algorithm. However, this run includes 268 iterations where the matching is stuck at 96.3%. Note that, for this problem, a complete matching requires 12 368 iterations and takes 0.013 s.

Table VI summarises the numbers in Table V by showing the fraction of the total factorization time spent in the scaling for each algorithm. The total factorization is taken to be the time to compute the scaling and then to factorize the scaled matrix (the time for pre-processing and post-processing the matrix data is not included, but is relatively small and easily parallelized). It shows that the use of the auction algorithm generally reduces the proportion of the time spent in scaling the matrix, especially for problems in test sets 1 and 3. The approximation algorithm spends a very small proportion of its time in scaling because the factorization time is so much larger.

DOI: 10.1002/nla

Table VII. Matching-based ordering and scaling: time (in seconds) to compute the ordering and to compute the factorization on 16 cores.

	Ordering and scaling			Factor		
Problem	Hungarian	sAuction	Approx	Hungarian	sAuction	Approx
GHS_indef/ncvxqp1	0.12	0.04	0.07	0.42	0.62	1.45
GHS_indef/cvxqp3	0.23	0.08	0.09	0.99	0.86	2.15
GHS_indef/ncvxqp5	0.66	0.41	0.29	1.66	1.97	177
GHS_indef/ncvxqp3	4.00	0.49	0.39	6.67	12.5	OOM
GHS_indef/stokes128	0.23	0.19	0.26	0.04	0.03	0.02
Schenk_IBMNA/c-62	0.25	0.22	0.26	2.42	1.87	130
Schenk_IBMNA/c-64	0.37	0.22	0.28	0.14	0.11	74.4
GHS_indef/boyd2	17.4	17.6	17.2	0.10	0.10	81.2
GHS_indef/bratu3d	0.06	0.05	0.06	0.18	0.16	0.16
GHS_indef/cont-201	0.12	0.12	0.16	0.05	0.04	0.03
GHS_indef/ncvxqp7	2.51	0.53	0.50	8.68	10.5	OOM
GHS_indef/cont-300	0.28	0.27	0.38	0.10	0.09	0.08
GHS_indef/darcy003	1.07	1.15	1.19	0.11	0.10	0.11
TSOPF/TSOPF_FS_b300_c2	1.68	1.67	2.91	0.41	0.38	1.38

Table VIII. Matching-based ordering and scaling: number of delayed pivots returned by `HSL_MA97`.

Problem	Hungarian	sAuction	Approx
GHS_indef/ncvxqp1	40	28 034	82 297
GHS_indef/cvxqp3	69	1675	120 608
GHS_indef/ncvxqp5	100	130	796 466
GHS_indef/ncvxqp3	260	15 722	OOM
GHS_indef/stokes128	7	4	9322
Schenk_IBMNA/c-62	1	0	617 535
Schenk_IBMNA/c-64	1	22	313 480
GHS_indef/boyd2	0	0	41 748
GHS_indef/bratu3d	340	340	1450
GHS_indef/cont-201	0	0	0
GHS_indef/ncvxqp7	226	9866	OOM
GHS_indef/cont-300	0	0	1
GHS_indef/darcy003	121	105	339
TSOPF/TSOPF_FS_b300_c2	2429	1403	148 164

4.3. Effectiveness of algorithms: ordering and scaling

We now consider the effectiveness of using a matching-based ordering combined with the matching-based scaling. As these techniques are known to be expensive [2], we only consider their application to problems in test sets 3 and 4.

Rather than the default choice between AMD and nested dissection, a matching-based ordering instead involves using a matching to identify 2×2 pivots, compressing the adjacency graph of the matrix such that the sparsity patterns of both members of the 2×2 pivot are merged into a single column before running a fill-reducing ordering on the compressed graph [6, 7]. There are thus three times to consider: (i) the time to run the matching algorithm (given in Table V of the previous section); (ii) the time to run the whole matching-based ordering routine, including the pre-processing, matching algorithm, graph compression and ordering; and (iii) the factorization time using the calculated scaling and ordering. Table VII reports the latter two times, while Table VIII demonstrates their ability to reduce the number of delayed pivots required during factorization.

We again see the approximation algorithm does not provide a sufficiently good matching for this approach to be effective. For most of our test problems, the Hungarian algorithm and the serial and parallel auction algorithms give comparable results and are extremely effective in substantially

DOI: 10.1002/nla

reducing the delayed pivots. However, for the ncvxqp/cvxqp problems, the Hungarian algorithm gives the best results, even for those problems for which the auction algorithms gave quality scalings of comparable quality (Table IV). These ncvxqp/cvxqp problems correspond exactly to those for which the cardinality of the auction algorithm matching was less than 99% (Table III). Additional experiments show that by running the serial auction algorithm until a 100% cardinality matching is reached, results comparable to the Hungarian algorithm can be obtained, while still offering a substantial time saving.

5. CONCLUSIONS

We have demonstrated that the auction algorithm fulfils its promise and provides comparable quality to the Hungarian algorithm in the context of scaling and ordering sparse symmetric matrices for use with direct solvers while being significantly faster. By contrast, the very fast $\frac{1}{2}$-approximation algorithm does not at present represent a reasonable alternative. We believe there is a substantial room for improvement in how the scaling is derived from the matching, and this is an obvious direction for future work.

Our results further show that high-quality scalings can be obtained using a suboptimal matching. However, the matching-based orderings generally require the matching to be of high cardinality to be fully effective in limiting the number of delayed pivots.

As the parallel auction algorithm requires additional work compared with the serial version, we recommend that the user is asked to choose which to use. In our tests, we were able to achieve consistent speedups with the parallel version on matrices that have in excess of two million entries; for smaller problems, it is more efficient to use the serial code.

In this paper, the emphasis has been on sparse symmetric systems. However, matchings are commonly used in the unsymmetric case to permute the matrix in order to obtain a zero-free diagonal of large elements to reduce the need for pivoting (see, e.g. the parallel solver SuperLU_DIST [26]). We suspect that this will have similar behaviour to that found in the symmetric case when permuting large entries to the sub-diagonal: specifically that the sub-optimal termination required to obtain a small runtime of the matching algorithm may not provide a matching of sufficient quality to avoid pivoting. Further, the failure to obtain a matching of maximum cardinality could necessitate some additional manipulation to ensure pivot candidates exist at the end of the factorization. A future objective is to investigate how effective the auction algorithm is for unsymmetric solvers and, in particular, whether a parallel implementation can reduce the time for the scaling and ordering without having a detrimental effect on the subsequent factorization (see also [17]).

Finally, we remark that an efficient implementation of the Hungarian algorithm is complicated, whereas that of both the serial and parallel versions of the auction algorithm is much more straightforward. We plan to include such implementations within our mathematical software libraries.

ACKNOWLEDGEMENTS

We would like to thank Mahantesh Halappanavar and Alex Pothen for providing access to their MATCHBOX Software used for the $\frac{1}{2}$-approximation matching. Thanks also to Sherry Li from Lawrence Berkeley National Laboratory and our colleague Iain Duff for commenting on a draft of this report. We are grateful to two anonymous reviewers for the constructive criticisms. Contract/grant sponsor: EPSRC; contract/grant number: EP/I013067/1.

REFERENCES

1. Hogg JD, Scott JA. The effects of scalings on the performance of a sparse symmetric indefinite solver. *Technical Report RAL-TR-2008-007*, Rutherford Appleton Laboratory, 2008.
2. Hogg JD, Scott JA. Pivoting strategies for tough sparse indefinite systems. *ACM Transactions on Mathematical Software* 2013; **40**:4:1–4:19.
3. Duff IS, Koster J. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal Matrix Analysis and Applications* 2001; **22**(4):973–996.

DOI: 10.1002/nla

4. Hogg JD, Scott JA. Optimal weighted matchings for rank-deficient sparse matrices. *SIAM Journal Matrix Analysis and Applications* 2013; **34**:1431–1447.
5. Kuhn HW. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 1955; **2**(1–2):83–97.
6. Duff IS, Pralet S. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM Journal Matrix Analysis and Applications* 2005; **27**:313–340.
7. Hagemann M, Schenk O. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM Journal Scientific Computing* 2006; **28**:403–420.
8. Schenk O, Gärtner K. On fast factorization pivoting methods for symmetric indefinite systems. *Electronic Transactions on Numerical Analysis* 2006; **23**:158–179.
9. Schenk O, Wächter A, Hagemann M. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Computer Optimization and Applications* 2007; **36**: 321–341.
10. Azad A, Halappanavar M, Rajamanickam S, Boman EG, Khan A, Pothen A. Multithreaded algorithms for maximum matching in bipartite graphs. *International Parallel and Distributed Processing Symposium*, IEEE Computer Society, Shanghai, China, 2012; 860–872.
11. Deveci M, Kaya K, Uçar B, Çatalyürek UV. GPU accelerated maximum cardinality matching algorithms for bipartite graphs. In *Euro-Par 2013 Parallel Processing, Lecture Notes in Computer Science*, Vol. 8097, Wolf F, Mohr B, Mey D (eds). Springer: Aachen, Germany, 2013; 850–861.
12. Halappanavar M, Feo J, Villa O, Tumeo A, Pothen A. Approximate weighted matching on emerging manycore and multithreaded architectures. *International Journal of High Performance Computing Applications* 2012; **26**(4): 413–430.
13. Sathe M, Schenk O, Burkhart H. An auction-based weighted matching implementation on massively parallel architectures. *Parallel Computing* 2012; **38**(12):595–614.
14. Bertsekas DP. A distributed asynchronous relaxation algorithm for the assignment problem. *24th IEEE Conference on Decision and Control*, Vol. 24, IEEE, Fort Lauderdale, Florida, USA, 1985; 1703–1704.
15. Bertsekas DP, Castañon DA. Parallel synchronous and asynchronous implementations of the auction algorithm. *Parallel Computing* 1991; **17**(6–7):707–732.
16. Buš L, Tvrdík P. Towards auction algorithms for large dense assignment problems. *Computer Optimization and Applications* 2009; **43**(3):411–436.
17. Riedy J. Making static pivoting scalable and dependable. *Ph.D. Thesis*, EECS Department, University of California, Berkeley, 2010. UCB/EECS-2010-172.
18. Mehlhorn K, Näher St. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press: Cambridge, UK, 1999.
19. Avis D. A survey of heuristics for the weighted matching problem. *Networks* 1983; **13**(4):475–493.
20. Preis R. Linear time $\frac{1}{2}$ - approximation algorithm for maximum weighted matching in general graphs. *16th Symposium on Theoretical Aspects of Computer Science (STACS)*, Trier, Germany, 1999; 259–269.
21. Hogg JD, Scott JA. On the efficient scaling of sparse symmetric matrices using an auction algorithm. *Technical Report RAL-P-2014-002*, Rutherford Appleton Laboratory, 2014.
22. Davis TA, Hu Y. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software* 2011; **38**(1):1:1–1:25.
23. Hogg JD, Scott JA. HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. *Technical Report RAL-TR-2011-024*, Rutherford Appleton Laboratory, 2011.
24. Hogg JD, Scott JA. New parallel sparse direct solvers for multicore architectures. *Algorithms* 2013; **6**:702–725. Special issue: Algorithms for Multi Core Parallel Computation.
25. Pothen A, Dobrian F, Halappanavar M. *Matchbox, A Library of Graph Matching Algorithms*. (Available from: http://www.cs.odu.edu/mhalappa/matching/), Development version, June 2013.
26. Li XS, Demmel JW. SuperLU_DIST: a scalable distributed-memory sparse direct solver or unsymmetric linear systems. *ACM Transactions on Mathematical Software* 2003; **29**(2):110–140.

DOI: 10.1002/nla