Pivoting Strategies for Tough Sparse Indefinite Systems

JONATHAN D. HOGG and JENNIFER A. SCOTT, Rutherford Appleton Laboratory

The performance of a sparse direct solver is dependent upon the pivot sequence that is chosen before the factorization begins. In the case of symmetric indefinite systems, it may be necessary to modify this sequence during the factorization to ensure numerical stability. These modifications can have serious consequences in terms of time as well as the memory and flops required for the factorization and subsequent solves. This study focuses on hard-to-solve sparse symmetric indefinite problems for which standard threshold partial pivoting leads to significant modifications. We perform a detailed review of pivoting strategies that are aimed at reducing the modifications without compromising numerical stability. Extensive numerical experiments are performed on a set of tough problems arising from practical applications. Based on our findings, we make recommendations on which strategy to use and, in particular, a matching-based approach is recommended for numerically challenging problems.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*Numerical algorithms*; G.4 [**Mathematical Software**]

General Terms: Algorithms

Additional Key Words and Phrases: Sparse symmetric linear systems, indefinite systems, direct solver, pivoting.

ACM Reference Format:
Hogg, J. D. and Scott, J. A. 2013. Pivoting strategies for tough sparse indefinite systems. ACM Trans. Math. Softw. 40, 1, Article 4 (September 2013), 19 pages.
DOI: http://dx.doi.org/10.1145/2513109.2513113

1. INTRODUCTION

The accurate and efficient solution of sparse symmetric indefinite linear systems has long been an important area of interest since such systems arise in a wide range of practical applications, including incompressible flow problems, electromagnetic scattering, eigenvalue problems and augmented systems within linear and nonlinear optimization problems. A key difference between a sparse direct solver for the solution of symmetric positive-definite systems and one for symmetric indefinite systems is that the latter needs to incorporate pivoting to maintain numerical stability. Not only does pivoting contribute significantly to the complexity of the development of the solver, it also adds overheads when the solver is run. These overheads occur in the search for a suitable pivot at each stage of the factorization and then in the handling of candidate pivots that are found to be unsuitable. We have recently developed new task-based sparse direct solvers for multicore machines [Hogg et al. 2010; Hogg and Scott 2010b, 2012a]. In the indefinite case, the need for pivoting means there is less scope for achieving parallelism and also a reduction in performance when compared to the positive definite case. Our

This work was funded by the EPSRC Grant EP/1013067/1.
Authors' addresses: Scientific Computing Department, STFC Rutherford Appleton Laboratory, Oxon OX11 0QX, England; email: jennifer.scott@stfc.ac.uk.

DOI: http://dx.doi.org/10.1145/2513109.2513113

long-term goal is to develop an efficient communication-avoiding strategy that, while fast, is as stable as standard threshold partial pivoting. This will be dependent on being able to precompute pivot sequences that can be used with (almost) no modification during the factorization, without compromising numerical stability and the accuracy of the computed solution while retaining sparsity in the factors.

The most common direct solution method for the sparse symmetric indefinite linear system

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

involves factorizing A into the form

$$A = LDL^T, \tag{2}$$

where D is a diagonal matrix with 1×1 and 2×2 pivot blocks and L is a sparse unit lower triangular matrix. In practice, a more general factorization of the form

$$SAS = PLD(PL)^T \tag{3}$$

is computed, where S is a diagonal scaling matrix and P is a permutation matrix (or, more generally, a product of permutation matrices) that holds the pivot order. In some implementations, this is further generalized to

$$S(A+E)S = PLD(PL)^T, \tag{4}$$

where E is a diagonal matrix with small entries. It is the choice of S, P and E that determines the sparsity of L as well as the accuracy and stability of the numerical factorization. The aim of this study is to examine different choices for P and E and, using a range of hard-to-solve symmetric indefinite linear systems, illustrate how well they work in practice. Based on the findings reported in Hogg and Scott [2008, 2012b], for S we use the scaling generated by the well-known weighted-matching code HSL_MC64 [Duff and Koster 2001].

Before the factorization commences, a pivot sequence (elimination order) is computed using one of the many available algorithms (for example, a variant of nested dissection [George 1973] or minimum degree [Tinney and Walker 1967; Liu 1985; Amestoy et al. 1996]). An analyze phase uses this sequence to set up the (provisional) data structures for the subsequent factorization phase. If at a given stage of the factorization there are p candidate pivots but only $q < p$ pivots are selected as being suitable, the remaining $p - q$ candidates are called *delayed*. The effects of this are that at a later stage in the factorization, the number of candidate pivots will be greater than predicted for the supplied pivot sequence; the data structures set up during the analyze phase will have to be modified to accommodate this, more operations will be performed in the computation of the factors, and L will be less sparse than if the pivot sequence were used without modification. These issues are well-known and over the last two decades, a number of techniques have been proposed to limit delays. We review and summarize these approaches and, using a set of hard-to-solve symmetric indefinite problems arising from practical applications, we report on how well they work in practice.

The article is organized as follows. In Section 2, we consider threshold partial pivoting and discuss variants that are designed to weaken the standard test criteria to reduce delayed pivots without, it is hoped, substantially affecting stability. We then consider, in Section 3, strategies to choose 2×2 pivot blocks during the analyze phase. Numerical experiments are reported in Section 4. Finally, we summarize our findings and recommendations in Section 5.

We observe that it is not our intention to attempt to exhaustively describe and review the pivoting strategies that are used in each of the modern state-of-the-art sparse indefinite solvers, although where appropriate, we indicate in which solver(s) a strategy is used. Our key contribution is a systematic review of the techniques that have been proposed to improve the performance of indefinite solvers and, using a set of tough problems that these methods are aimed at, we examine and compare their effectiveness as well as looking at using them in combination. Experts in the development of sparse direct solvers undoubtedly understand the issues well but we have been unable to find any comprehensive results that bring together and compare and combine different strategies. Furthermore, our contact with users of sparse indefinite solvers has led us to believe that they are not always aware of the potential consequences of selecting a particular strategy and may not appreciate that some approaches may sacrifice robustness for speed.

2. PIVOTING STRATEGIES

2.1. Threshold Partial Pivoting

In the case of symmetric linear systems, 1×1 and 2×2 pivoting must be performed if symmetry is to be kept while stability is retained. Stability of the factorization of symmetric indefinite systems was considered in detail by Ashcraft et al. [1999]. They showed that bounding the size of the entries of L, together with a backward stable scheme for solving 2×2 linear systems, suffices to show backward stability for the entire solution process. They found that the widely used strategy of Bunch and Kaufmann [1977] does not have this property whereas the threshold pivoting technique first used by Duff and Reid [1983] in their original multifrontal solver does.

Duff and Reid choose the pivots one-by-one, with the aim of limiting the size of the entries $l_{i,j}$ in L so that

$$|l_{i,j}| < u^{-1}, \tag{5}$$

where the pivot threshold parameter u is in the range $0 \le u \le 1.0$. In the case where u is zero, this is interpreted as requiring that the entries be finite. Suppose q is the number of rows and columns of D found so far (that is, the number of 1×1 pivots plus twice the number of 2×2 pivots). Let $a_{i,j}$, with $i > q$ and $j > q$, denote an entry of the matrix after it has been updated by all the permutations and pivot operations so far. For a 1×1 pivot in column $j = q+1$, the requirement for inequality (5) corresponds to the threshold test

$$|a_{q+1,q+1}| > u \max_{q+1<i\le n} |a_{i,q+1}|. \tag{6}$$

The original test used by Duff and Reid for 2×2 pivots proved unnecessarily severe. Instead, following Duff et al. [1991], an appropriate test for a 2×2 pivot is

$$\left| \begin{pmatrix} a_{q+1,q+1} & a_{q+1,q+2} \\ a_{q+1,q+2} & a_{q+2,q+2} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{q+2<i\le n} |a_{i,q+1}| \\ \max_{q+2<i\le n} |a_{i,q+2}| \end{pmatrix} < \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}, \tag{7}$$

where the absolute value notation for a matrix refers to the matrix of corresponding absolute values and the threshold parameter u is between 0 and 0.5. In the case where u is zero, this is interpreted as requiring that the pivot be nonsingular. While this test is not used universally by modern sparse direct solvers, it has been incorporated into all recent sparse indefinite solvers within the HSL mathematical software library [HSL 2013] (including `MA57` [Duff 2004], `HSL_MA77` [Reid and Scott 2008, 2009], `HSL_MA86` [Hogg and Scott 2010b] and `HSL_MA97` [Hogg and Scott 2011]) and is thus the one used in this study.

2.2. Choice of Pivot Threshold Parameter

The choice of the pivot threshold parameter u not only influences the number of candidate pivots that are rejected and hence delayed, but also the stability of the factorization. A large u means a tight bound on the size of the entries of L at the possible cost of a large number of delays, whereas a smaller value potentially reduces the number of delays but, because the pivot test is less strict, the factorization may be less accurate and it may be necessary to perform refinement during the solve phase to try and recover the required accuracy. In extreme cases, entries of the factors may become unbounded. Frequently used choices are $u = 0.1$ or 0.01. On the basis of extensive numerical experience, these values generally provide a good compromise between stability and sparsity. However, in some application areas it is common to use a much smaller threshold, despite the attendant risk of numerical instability. For example, the well-known optimization package Ipopt [Wächter and Biegler 2006] optionally uses one of the HSL indefinite solvers `MA27` [Duff and Reid 1992] and `MA57`. For both solvers, the default setting for u within Ipopt is $u = 10^{-8}$. If, at some stage of the computation, this is found to give an unstable factorization (detected via an unexpected inertia or large backward error following solution), the factorization is recomputed with a larger of value of u, and this process repeated as necessary until either a stable factorization is achieved or the maximum allowable value for u is reached (which, by default, in Ipopt is 10^{-4}). The use of small u in an optimization context is common and has been discussed, for example, in Fourer and Mehrotra [1993] and Gill et al. [1996]. Numerical experiments using $u = 10^{-8}$ are reported on in Section 4.2.

2.3. Relaxed Threshold Pivoting

The idea behind relaxed threshold pivoting is to relax the threshold parameter during the factorization when no pivot satisfying the threshold tests for the input u is available. In this way, the pivot threshold used during the computation may be smaller than at the start of the factorization, reducing the number of delays but ideally without seriously compromising stability. Relaxed pivoting strategies have been explored by Duff and Pralet [2007] (see also Reid and Scott [2011]). The factorization commences with the user-supplied u. If at some stage no 1×1 or 2×2 candidate pivot satisfies the tests (6) or (7), pivots are accepted using a weaker threshold, provided this is at least a user-defined minimum $u_{\min}$. Our numerical experiments [Hogg and Scott 2012b] found that, compared to using a small u throughout the factorization, the gains were modest and do not justify the additional complexity they add to the implementation.

2.4. Restricted Pivoting

Threshold partial pivoting requires all the entries below the diagonal in the candidate pivot column(s) to be searched when checking the threshold tests (6) and (7). This is expensive, particularly in a parallel implementation. One way of avoiding this is, at each stage, to limit the search to a restricted set of rows (typically the rows corresponding to the candidate pivot columns). Thus, if there are r such rows, (6) is replaced by

$$|a_{q+1,q+1}| > u \max_{q+1<i\leq r} |a_{i,q+1}|, \tag{8}$$

and (7) is modified in the same way. The hope here is that, for a well-scaled matrix, restricting the search will save time without leading to the acceptance of pivots that would otherwise have been rejected. Because of the greater potential for numerical instability, refinement is more likely to be needed to recover accuracy. Note that time is saved not only in the search for pivots but also because entries in the candidate pivot columns that lie in rows $r + 1$ to n do not need to be updated after each pivot has been chosen. This allows block update operations (using high level BLAS kernels) to be employed, thereby improving efficiency. Furthermore, additional parallelism can be exploited when performing these updates.

2.5. Static Pivoting

A static pivoting strategy refers to one that allows the factorization to respect the pivot ordering passed to it from the analyze phase. The factorization does not necessarily follow the analysis exactly and some slight variations are allowed. For example, in a multifrontal algorithm it is sufficient that the factorization decisions be compatible with the assembly tree (numerical pivoting can be performed within a front). The key point is that pivots are not delayed so that the analysis predicts exactly the memory and operations needed to perform the factorization and thus the data structures can remain static. A static approach for LU factorization was proposed by Li and Demmel [1998] for the SuperLU solver. If necessary, during the factorization, small perturbations are added to the diagonal to prevent pivots from becoming too small and failing the threshold test. The computed factorization is thus not of A but of a perturbed matrix $A + E$, where E is a diagonal matrix.

For symmetric indefinite systems, Schenk and Gärtner [2006] combine static pivoting with Bunch-Kaufman and restricted pivoting strategies. Their strategy is implemented within the solver PARDISO [Schenk and Gärtner 2004]. A similar approach is available as an option within WSMP [Gupta 2000]. In this case, the matrix is perturbed whenever numerically acceptable 1×1 and 2×2 pivots cannot be found within a diagonal supernode block (checks on potential pivots are only made within the supernode block). This has been shown to perform well in many applications. An important downside is that the solve phase can require an increased number of refinement steps to achieve the requested accuracy (and, in the extreme case, as our numerical results in Section 4.3 show, it may not be possible to recover accuracy). Furthermore, since a perturbed system is solved, the computed inertia of the original matrix A may not be reliable and, in some applications, accurate knowledge of the inertia is required (for example, to ensure local convexity in a nonlinear interior point method).

Duff and Pralet [2007] propose combining threshold partial pivoting with static pivoting to try to minimize both the number of perturbations that are added and the amount of refinement required. If no 1×1 or 2×2 candidate pivot satisfies the test (6) or (7) the 1×1 pivot that is nearest to satisfying the test is accepted. If its absolute value is less than another user-supplied threshold *static*, it is given the value that has the same sign but absolute value *static* [Reid and Scott 2011]. This strategy (or variants of it) is available as an option within a number of solvers, including `MA57`, `HSL_MA77`, `HSL_MA86`, and MUMPS [Amestoy et al. 1999; MUMPS 2011].

3. BLOCK PIVOT STRATEGIES

Standard algorithms for computing a pivot sequence, including minimum degree and nested dissection, compute a sequence of 1×1 pivots. For many indefinite problems, it is necessary to use 2×2 pivots during the factorization and so the supplied pivot sequence will be modified. To try to minimize modifications, it is therefore natural to try and construct a tentative pivot sequence that contains 2×2 pivots. In this section, we discuss a number of approaches that have been proposed for this.

3.1. Reusing a Pivot Sequence

If a sequence of linear systems

$$A_k x_k = b_k \tag{9}$$

is to be solved in which the matrix A_k varies from the previous matrix A_{k-1} by a relatively small amount (in terms of both the sparsity pattern and the values of the entries), an obvious possibility is to pass the pivot sequence actually used by the factorization phase for A_{k-1} to the analyze phase for A_k. This is a strategy that could be used, for example, within a nonlinear optimization package, such as Ipopt.

In Hogg and Scott [2012b], we performed experiments with $k = 2$ and $A_2 = A_1 = A$ (that is, we call the analyze and factorization phases twice and on the second call to the analyze phase we input the pivot sequence returned by the first call to the factorization phase). The results were disappointing, with only 8 out of 23 test problems achieving a modest speedup while 12 ran more slowly (in some cases significantly slower). The cause of this under performance is that the assembly trees used in the first and subsequent factorizations are very different because of the effects of supernode amalgamation. It may be that additional work will yield better techniques for preserving assembly trees between the factorizations but it is beyond the scope of this study.

3.2. `MA47` Orderings

`MA47` [Duff and Reid 1996] is a sparse symmetric solver specifically designed for solving indefinite systems and, in particular, for matrices that have some zeros on the diagonal. A key feature is that the analyze phase may choose tentative block pivots. A variant of the Markowitz criterion recommended in Duff et al. [1991] is used to extend the strategy of minimum degree to block pivots. As with other minimum degree algorithms, the pivots are chosen during the analyze using the sparsity structure of A alone (without the assumption that the diagonal is implicitly present that is made by standard fill-reducing ordering algorithms).

`MA47` distinguishes between block pivots with different sparsity patterns. Specifically, pivots may be classified as

—*oxo* pivots of the form

$$\begin{pmatrix} 0 & a_1 \\ a_1 & 0 \end{pmatrix}, \tag{10}$$

—*tile* pivots of the form

$$\begin{pmatrix} a_2 & a_1 \\ a_1 & 0 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & a_1 \\ a_1 & a_2 \end{pmatrix}, \tag{11}$$

—or of any other form, called *full*.

Tile and oxo pivots are termed *structured* pivots and their structure is taken into account within the analyze and factorize phases. As far as we are aware, `MA47` is the only sparse symmetric solver that exploits the structure of block pivots. However, the analyze phase of `MA47` can be used to compute a tentative pivot sequence containing 2×2 structured pivots that may be passed to other solvers.

3.3. Constraint Pivot Orderings

Many tough indefinite problems are saddle-point problems of the form

$$A = \begin{pmatrix} H & B^T \\ B & -C \end{pmatrix}, \tag{12}$$

with H symmetric positive semidefinite, B rectangular, and C symmetric positive semidefinite. The problem of finding a permutation P such that PAP^T can be factorized stably without the need for numerical pivoting and without modifying the entries of A, while still limiting the number of entries in L, has been examined for special classes of such matrices [de Niet and Wubs 2009; Tůma 2002]. For general saddle-point problems, Bridson [2007] proposed splitting the nodes of the adjacency graph of A into two disjoint sets: those that correspond to the diagonal entries of H are known as H-nodes and the remaining nodes as C-nodes. The ordering constraint proposed by Bridson is extremely simple: a C-node can only be ordered after all its H-node neighbours have been ordered. Bridson shows that, provided H is semi-definite and B is of full row rank, with this constrained ordering the LDL^T factorization exists. Moreover, the pivots associated with the H-nodes are guaranteed to be positive and those associated with C-nodes are guaranteed to be negative. By rescaling, $L \leftarrow L|D|^{1/2}$ and $D \leftarrow \text{sign}(D) = \text{diag}(\pm 1)$, the diagonal matrix is fully determined in advance by the structure of the problem, independent of the numerical values. This constrained ordering allows a Cholesky factorization code to be modified to perform the factorization of the indefinite A with **no** threshold pivoting (that is, this ordering can be used without the need to perform numerical pivoting or to alter the pivot sequence). While a stability analysis is lacking, Bridson reports the constrained ordering is generally sufficient to avoid the need for numerical pivoting (see also Scott [2009]). The hope is that, if an initial ordering is chosen to reduce fill in L, the modifications required to obtain a constrained ordering will be such that the additional fill will be modest.

Bridson proposed two approaches to computing a constrained ordering. The first modifies the minimum degree algorithm (or one of its variants) to incorporate the constraint within it. An alternative approach is to post process a given fill-reducing ordering to satisfy the constraint. If a C-node is the next node in the supplied ordering it is only included in the modified ordering once all its H-node neighbours have been ordered (that is, a C-node is postponed until after all its H-node neighbours). This approach can be applied to any fill-reducing ordering and is very cheap and straightforward to implement. As Bridson reports that neither approach consistently outperforms the other, our experiments use post processing.

We remark that WSMP offers a limited form of constrained ordering for indefinite systems. This allows the user to specify that the final $m < n$ columns are to be pivoted on last. This is recommended in the WSMP documentation for problems that have a few zero (or near-zero) entries on the diagonal. For such matrices, an LDL^T factorization is performed without pivoting. The documentation states. "By ordering the $n - m$ rows and columns with zero diagonal entries in the end, the user ensures (unless there is numerical cancellation) that these diagonal entries are nonzero by the time they are pivoted on."

3.4. Matching-Based Orderings

In the unsymmetric case, maximum weighted matching algorithms are used to move large entries on to the diagonal of the matrix. The idea is that these will provide potentially good candidate pivots and the number of delayed pivots during the subsequent factorization will be reduced. In the symmetric case, symmetry needs to be preserved but a symmetric permutation leaves the diagonal unchanged. Thus the aim is to permute a large off-diagonal entry $a_{i,j}$ close to the diagonal so that the 2×2 block

$$\begin{pmatrix} a_{i,i} & a_{i,j} \\ a_{i,j} & a_{j,j} \end{pmatrix} \tag{13}$$

is potentially a good 2×2 candidate pivot. Duff and Gilbert [2002] noticed that the cycle structure of the permutation $P_{\mathcal{M}}$ associated with the unsymmetric maximum weighted matching $\mathcal{M}$ can be exploited to obtain such a permutation P_s. This has been explored further by Duff and Pralet [2005] and, amongst others, Schenk et al. [Hagemann and Schenk 2006; Schenk and Gärtner 2006; Schenk et al. 2007], and symmetric maximum weighted matchings are (optionally) used within the sparse solvers HSL_MA97, MUMPS, PARDISO and WSMP.

A maximum weighted matching $\mathcal{M}$ is first computed. Assume initially that A is not found to be structurally singular. Any diagonal entries that are in the matching are immediately considered as potential 1×1 pivots and are held in a set $\mathcal{M}_1$. A set $\mathcal{M}_2$ of potential 2×2 pivots is then built by expressing the computed permutation $P_{\mathcal{M}}$ in terms of its component cycles. Because of the scaling, all the entries in the cycles of $P_{\mathcal{M}}$ are 1 in absolute value so a structural criterion is used to select the potential 2×2 pivots. A cycle of length 1 corresponds to an entry a_{ii} in the matching. A cycle of length 2 corresponds to two nodes i and j, where a_{ij} and a_{ji} are both in the matching. k potential 2×2 pivots can be extracted from even cycles of length $2k$ or from odd cycles of length $2k+1$. The idea is to select k entries a_{ij} and their symmetric counterpart and to discard the other matched entries. In practice, most of the cycles in the matching permutation are of length 1 or 2 [Duff and Pralet 2005]. Where there are longer cycles, Duff and Pralet discuss possible ways of extracting 2×2 pivots, based on the sparsity patterns of the rows of A (in particular, they seek to pair up rows that have as similar a structure as possible). Since long cycles do not occur often, we adopt the more straightforward approach of taking the first two entries as the first 2×2 pivot, the next two as the next 2×2 pivot, and so on, until if the cycle is of odd length, a single entry remains, which is added to the set $\mathcal{M}_1$.

To combine the resulting permutation with a fill-reducing ordering (such as nested dissection or minimum degree), the graph of $P_s A P_s^T$ is compressed and the ordering applied to the compressed graph. In the compression step, the union of the sparsity structure of the two rows and columns corresponding to a potential 2×2 pivot is built and used as the structure of a single row and column in the compressed matrix. A fill-reducing ordering is applied to the (weighted) compressed graph, and the resulting permutation is expanded to a permutation P_f for the original matrix. The final permutation is the product $P = P_f P_s$. The rows/columns corresponding to a potential 2×2 pivot are reordered consecutively.

A by-product of computing a matching-based ordering is a scaling for A and this is the scaling computed by HSL_MC64. To employ a matching algorithm within the analyze phase it is necessary for the analyze phase to have available the numerical values of the entries of A (and thus the analyze phase will not depend solely on the structure of A). If the user wants to factorize more than one matrix with the same sparsity pattern but different numerical values, it may be necessary to recompute the ordering and the scaling. This can add a significant overhead when compared with an analyze phase that uses the sparsity structure only. However, if scaling factors need to be computed prior to the factorization of each matrix, then the additional cost associated with the matching-based ordering will generally be modest compared with the total solution time.

4. NUMERICAL EXPERIMENTS

4.1. Test Environment

As already indicated, the HSL mathematical software library contains a number of sparse solvers that are designed for symmetric indefinite systems. In this study, most of our experiments are performed using HSL_MA77 (Version 5.8.0). HSL_MA77 implements a multifrontal algorithm and includes the possibility of holding the matrix data, the computed factors, and some of the intermediate work arrays in files on disk, thus allowing the solution of much larger problems than would otherwise be possible. HSL_MA77 offers the user a number of options. Importantly for this study, these include the use of threshold pivoting and static pivoting. Furthermore, the user is required to supply the pivot sequence (which may include 2×2 pivots) to the analyze phase, allowing us to experiment with constraint and matching-based orderings. An option also exists to supply a scaling. In our experiments, we use the scaling returned by HSL_MC64.

All our experiments are performed using double precision on a Dell Precision T5400 with two Intel E5420 quad core processors running at 2.5 GHz. The ifort compiler

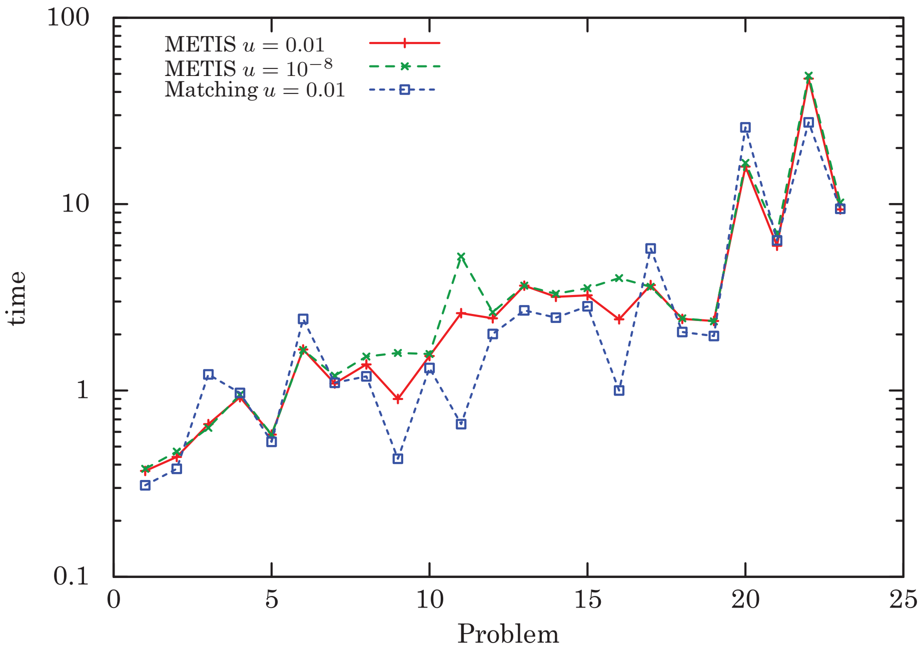

Fig. 1. The complete solution times (in seconds) for HSL_MA97 run with METIS and the matching ordering ($u = 0.01$ and $u = 10^{-8}$).

(Version 12.0) with option -O3 and MKL BLAS (Version 10.2) are used. The right-hand side b is chosen so that the solution is $x_i = 1$ for all i. We measure the accuracy of the computed solution using the scaled residual:

$$\frac{\|Ax - b\|}{\|A\|\,\|x\| + \|b\|} \tag{14}$$

with the infinity norm $\|x\| = \max_i |x_i|$ and its induced matrix norm $\|A\| = \max_i \sum_j |a_{i,j}|$. The computed solution is only accepted if the scaled residual is less than 10^{-14}; where necessary, iterative refinement or refinement using Flexible GMRES (FGMRES) [Saad 1993; Arioli and Duff 2009] is performed. For some problems, FGMRES is able to compute backward stable solutions when iterative refinement fails to converge but, as it is more expensive, we restrict its use to such cases. We use the restarted FGMRES algorithm described in [Hogg and Scott 2010a]. This is essentially as given in Arioli and Duff [2009] but additionally uses an adaptive restart parameter that was found in numerical experiments to be more efficient than using a fixed restart parameter (that is, in general, it reduced the number of iterations required). Our choice of initial restart parameter of 4 is based on the results given in Hogg and Scott [2010a]. Note that the results are sensitive to this choice: using a larger value can lead to a larger total number of solves (that is, to a larger number of calls to the solve phase of the direct solver) because we only test the termination conditions when FGMRES is restarted. When iterative refinement or FGMRES is used, we limit the number of solves to 100.

For each test, we impose for each problem a time limit of 1 hour (note that runtimes for most problems are less than 1 minute; see Figure 1). In the tables of results, *nitr* is the number of solves performed during refinement (0 indicates refinement was not needed); OOT denotes the time limit was exceeded and * indicates the required accuracy was not achieved (after refinement).

Table I.
Test problems. n denotes the order of A, $nz(A)$ is the number of entries in A, $nz(L)$ is the predicted number of entries in L and *nflop* is the predicted number of flops required to compute L (METIS ordering). QP = quadratic programming problem, FE = finite-element.

Identifier	n	$nz(A)$	$nz(L)$	*nflop*	Description/Application
TSOPF/TSOPF_FS_b162_c1	10798	608540	1.3930×10^6	2.0509×10^8	Optimal power flow
TSOPF/TSOPF_FS_b39_c7	28216	730080	1.5755×10^6	1.0796×10^8	Optimal power flow
GHS_indef/ncvxqp1	12111	73963	1.6839×10^6	7.2793×10^8	Non-convex QP
QY/case39	40216	1042160	2.2885×10^6	1.5130×10^8	Optimal power flow
GHS_indef/stokes128	49666	558594	2.9813×10^6	3.6881×10^8	FE model Stokes problem
GHS_indef/cvxqp3	17500	122462	3.1398×10^6	1.7670×10^9	Convex QP
TSOPF/TSOPF_FS_b162_c3	30798	1801300	4.2194×10^6	6.3970×10^8	Optimal power flow
TSOPF/TSOPF_FS_b39_c19	76216	1977600	4.4010×10^6	2.8666×10^8	Optimal power flow
GHS_indef/cont-201	80595	438795	4.7815×10^6	8.6513×10^8	Convex QP
TSOPF/TSOPF_FS_b162_c4	40798	2398220	5.5772×10^6	8.3825×10^8	Optimal power flow
GHS_indef/bratu3d	27792	173796	6.2769×10^6	4.4174×10^9	Optimization
TSOPF/TSOPF_FS_b39_c30	120216	3121160	7.0473×10^6	4.7541×10^8	Optimal power flow
GHS_indef/darcy003	389874	2101242	8.1587×10^6	5.5664×10^8	Mixed FE model
TSOPF/TSOPF_FS_b300	29214	4400122	1.0603×10^7	4.3977×10^9	Optimal power flow
TSOPF/TSOPF_FS_b300_c1	29214	4400122	1.0603×10^7	4.3977×10^9	Optimal power flow
GHS_indef/cont-300	180895	988195	1.1744×10^7	2.9559×10^9	Convex QP
GHS_indef/ncvxqp5	62500	424966	1.2052×10^7	9.7223×10^9	Non-convex QP
GHS_indef/turon_m	189924	1690876	1.3723×10^7	4.2270×10^9	Mixed FE model
GHS_indef/d_pretok	182730	1641672	1.4581×10^7	5.0572×10^9	Mixed FE model
GHS_indef/ncvxqp3	75000	499964	1.9007×10^7	2.0692×10^{10}	Non-convex QP
TSOPF/TSOPF_FS_b300_c2	56814	8767466	2.1433×10^7	8.9629×10^9	Optimal power flow
GHS_indef/ncvxqp7	87500	574962	2.4731×10^7	3.0939×10^{10}	Non-convex QP
TSOPF/TSOPF_FS_b300_c3	84414	13135930	3.3105×10^7	1.4253×10^{10}	Optimal power flow

The problems used in this study are from the University of Florida Sparse Matrix Collection [Davis and Hu 2011]. The Collection includes a large number of nonsingular symmetric indefinite matrices. We ran our solver on all such problems of order at least 10,000 for which numerical values are supplied.[1] We used default settings, nested dissection ordering computed using routine METIS_NodeND from the METIS package [Karypis and Kumar 1998] (note we use METIS Version 4), and `HSL_MC64` scaling. We include in our test set those for which the number of reported delayed pivots is at least $\max(5000, 0.2 * n)$; this is the definition of a tough indefinite problem used in this study. Our test problems are listed in Table I. For each problem, we give its order n and number of entries $nz(A)$. In addition, we report the number of entries $nz(L)$ in L and the number *nflop* of floating-point operations (flops) required to compute L that are returned by the analyze phase of `HSL_MA77` and are the values that would be returned by the factorization phase if the supplied pivot sequence could be used without modification. Further details of the problems can be found by referring to [Davis and Hu 2011].

4.2. A Comparison of Pivoting Strategies (without Static Pivoting)

In Tables II to V, we report results for the following orderings: METIS ($u = 0.01$ and 10^{-8}), `MA47` ($u = 0.01$), matching ($u = 0.01$ and 10^{-8}) and matching combined with restricted pivoting ($u = 0.01$). For the matching orderings, routine METIS_NodeWND

[1]All problems available as of March 2012 were included.

Table II.
Number of delayed pivots for various methods. OOT denotes the 1 hour time limit was exceeded. The numbers in parentheses are for Matching using METIS_NodeND. * indicates required accuracy not achieved.

						Matching
Ordering:	METIS	METIS	MA47	Matching	Matching	Restrict
u:	0.01	10^{-8}	0.01	0.01	10^{-8}	0.01
TSOPF/TSOPF_FS_b162_c1	5087	3961	225	354 (348)	0	0
TSOPF/TSOPF_FS_b39_c7	11238	8527	883	1315 (1356)	0	0
GHS_indef/ncvxqp1	10359	9267	10731	337 (25)	292	224
QY/case39	14806	11011	4800	5392 (5925)	0	0
GHS_indef/stokes128	9274	9274	2548	5 (5)	5	50
GHS_indef/cvxqp3	26152	26145	18383	186 (64)	0	0
TSOPF/TSOPF_FS_b162_c3	17497	12126	441	1381 (1264)	0	0
TSOPF/TSOPF_FS_b39_c19	29668	22610	4640	8074 (9010)	0	0
GHS_indef/cont-201	70021	66002	185801	0 (0)	0	0
TSOPF/TSOPF_FS_b162_c4	21778	15926	532	1482 (1410)	0	0
GHS_indef/bratu3d	58888	41829	250517	0 (0)	0	0
TSOPF/TSOPF_FS_b39_c30	47578	37293	10135	5655 (5699)	0	0
GHS_indef/darcy003	43702	43702	18909	345 (119)	345	213*
TSOPF/TSOPF_FS_b300	20599	19033	1898	1105 (1017)	0	0
TSOPF/TSOPF_FS_b300_c1	24511	19310	2430	1220 (1269)	0	0
GHS_indef/cont-300	148976	141089	OOT	0 (0)	0	0
GHS_indef/ncvxqp5	11869	10636	OOT	85 (112)	0	0
GHS_indef/turon_m	18931	18928	15101	262 (129)	234	190*
GHS_indef/d_pretok	21399	18012	17715	758 (313)	698	441*
GHS_indef/ncvxqp3	65603	64876	OOT	641 (220)	564	547*
TSOPF/TSOPF_FS_b300_c2	52593	41836	OOT	2579 (2533)	0	0
GHS_indef/ncvxqp7	272409	270790	OOT	104 (195)	89	89
TSOPF/TSOPF_FS_b300_c3	116630	99361	OOT	5205 (6764)	0	0

is applied to the compressed graph, with a vertex weight of two for each vertex that corresponds to two rows/columns of the original matrix and a weight of one for all other vertices. For $u = 0.01$ we also include results for the matching ordering with the unweighted variant (METIS_NodeND) applied to the compressed graph. These show that the solver performance is generally not very sensitive to whether METIS_NodeWND or METIS_NodeND is used but, in terms of the flop counts (Table IV), for some problems (including GHS_indef/cvxqp3, GHS_indef/darcy003, and GHS_indef/d_pretok), the former gives worthwhile savings. Note that we also experimented with combining the METIS ordering with restricted pivoting but found that, for our tough indefinite problems, iterative refinement and FGMRES were unable, in general, to recover the required accuracy. If restricted pivoting is combined with a matching-based ordering, the approach is generally successful, but since it failed to achieve the required accuracy for three of our problems, it does not offer the robustness that we require. Table II reports the number of delayed pivots. If a candidate pivot is delayed more than once, we count the number of times it is delayed. We see that the matching-based orderings generally lead to the smallest number of delayed pivots, although for some of the TSOPF problems, the MA47 ordering also results in a relatively small number of delays. Indeed, in most cases where the time limit was not exceeded, the MA47 ordering led to fewer delays than the METIS ordering. However, for large problems, minimum degree-based orderings (such as the MA47 ordering) are generally not as effective as those based on nested dissection. This is seen in Tables III and IV, where the ratios

Table III.
Number of entries in the factors for various methods. Ratios of the actual number of entries against the predicted number of entries for the METIS ordering are given. The numbers in parentheses are for Matching using METIS_NodeND. OOT denotes the 1 hour time limit was exceeded. * indicates required accuracy not achieved.

						Matching
Ordering:	METIS	METIS	MA47	Matching	Matching	Restrict
u:	0.01	10^{-8}	0.01	0.01	10^{-8}	0.01
TSOPF/TSOPF_FS_b162_c1	1.31	1.20	9.67	1.25 (1.26)	1.23	1.23
TSOPF/TSOPF_FS_b39_c7	1.40	1.24	17.2	1.49 (1.50)	1.46	1.46
GHS_indef/ncvxqp1	1.35	1.32	12.4	1.79 (1.70)	1.83	1.83
QY/case39	1.36	1.21	15.1	1.63 (1.63)	1.56	1.56
GHS_indef/stokes128	1.10	1.10	1.12	1.52 (1.59)	1.52	1.52
GHS_indef/cvxqp3	1.56	1.56	3.63	1.69 (1.82)	1.70	1.70
TSOPF/TSOPF_FS_b162_c3	1.44	1.23	27.1	1.43 (1.43)	1.40	1.40
TSOPF/TSOPF_FS_b39_c19	1.49	1.22	29.0	1.56 (1.56)	1.49	1.49
GHS_indef/cont-201	1.79	1.72	55.9	0.92 (0.95)	0.92	0.92
TSOPF/TSOPF_FS_b162_c4	1.49	1.25	36.1	1.23 (1.23)	1.21	1.21
GHS_indef/bratu3d	1.88	1.53	16.0	0.90 (0.97)	0.90	0.90
TSOPF/TSOPF_FS_b39_c30	1.61	1.22	41.7	1.46 (1.46)	1.40	1.40
GHS_indef/darcy003	1.09	1.09	1.32	1.63 (1.75)	1.63	1.63*
TSOPF/TSOPF_FS_b300	1.20	1.19	9.37	1.26 (1.24)	1.23	1.23
TSOPF/TSOPF_FS_b300_c1	1.29	1.19	9.39	1.25 (1.27)	1.22	1.22
GHS_indef/cont-300	1.83	1.77	OOT	0.94 (0.95)	0.94	0.94
GHS_indef/ncvxqp5	1.11	1.11	OOT	1.53 (1.64)	1.53	1.53
GHS_indef/turon_m	1.05	1.05	3.28	1.22 (1.37)	1.22	1.21*
GHS_indef/d_pretok	1.05	1.04	2.92	1.20 (1.38)	1.20	1.20*
GHS_indef/ncvxqp3	1.32	1.32	OOT	1.79 (1.84)	1.78	1.78*
TSOPF/TSOPF_FS_b300_c2	1.35	1.21	OOT	1.23 (1.24)	1.20	1.20
GHS_indef/ncvxqp7	1.59	1.58	OOT	1.72 (1.75)	1.65	1.65
TSOPF/TSOPF_FS_b300_c3	1.38	1.19	OOT	1.21 (1.19)	1.16	1.16

for MA47 are generally much larger than for METIS (and, in 6 instances, the time limit using the MA47 ordering is exceeded).

From column 2 of Tables III and IV, we see that, for many of our test examples run with default settings, the actual number of entries in L is more than 50% greater than predicted and the difference between the predicted and actual number of flops can be significantly larger (close to a factor of 10 for TSOPF/TSOPF_FS_b39_c30). For each of our prescaled test examples, at most one step of iterative refinement is required, confirming that stability is achieved with threshold $u = 0.01$. Using a smaller u (column 3) may have little or no effect on reducing the ratios (they are unchanged for problems GHS_indef/darcy003 and GHS_indef/turon_m). However, for some problems, $u = 10^{-8}$ leads to worthwhile reductions in $nz(L)$ and *nflop* (for example, GHS_indef/bratu3d and TSOPF/TSOPF_FS_b39_c30), but possibly at the cost of a large number of solves (and possibly FGMRES) being needed during refinement.

The implementation of the matching-based ordering is that provided by the HSL package HSL_MC80, with METIS applied to the compressed graph. The predictions for the matching-based ordering are usually greater than for the METIS ordering (typically, between 50 and 100% greater). However, with the default threshold ($u = 0.01$), the matching-based ordering results in substantially fewer delayed pivots. If we run the matching-based ordering with $u = 10^{-8}$, for all our test examples, the number of delays reduces to 0 (or close to 0) and, in contrast to the METIS ordering with the same u, we found a single step of iterative refinement sufficient to achieve the requested accuracy

Table IV.
Number of flops to perform the factorization using various methods. Ratios of the actual number of flops against the predicted number of flops for the METIS ordering are given. The numbers in parentheses are for Matching using METIS_NodeND. OOT denotes the 1 hour time limit was exceeded. * indicates required accuracy not achieved.

Ordering:	METIS	METIS	MA47	Matching	Matching	Matching Restrict
u:	0.01	10^{-8}	0.01	0.01	10^{-8}	0.01
TSOPF/TSOPF_FS_b162_c1	1.82	1.45	221	1.56 (1.58)	1.50	1.50
TSOPF/TSOPF_FS_b39_c7	2.56	1.64	673	1.96 (1.98)	1.87	1.87
GHS_indef/ncvxqp1	1.68	1.61	102	3.37 (2.84)	3.56	3.55
QY/case39	2.63	1.56	520	2.50 (2.53)	2.27	2.27
GHS_indef/stokes128	1.09	1.09	1.37	1.98 (2.23)	1.98	1.98
GHS_indef/cvxqp3	2.16	2.16	13.3	2.65 (3.08)	2.60	2.60
TSOPF/TSOPF_FS_b162_c3	2.78	1.52	1782	2.04 (2.03)	1.96	1.96
TSOPF/TSOPF_FS_b39_c19	5.38	1.56	3551	2.44 (2.44)	2.16	2.16
GHS_indef/cont-201	3.00	2.75	4134	0.82 (0.87)	0.82	0.82
TSOPF/TSOPF_FS_b162_c4	3.37	1.60	3179	1.52 (1.52)	1.46	1.46
GHS_indef/bratu3d	2.75	1.81	163	0.84 (0.96)	0.84	0.84
TSOPF/TSOPF_FS_b39_c30	9.81	1.56	8235	2.22 (2.22)	1.82	1.82
GHS_indef/darcy003	1.08	1.08	2.57	3.46 (3.95)	3.46	3.46*
TSOPF/TSOPF_FS_b300	1.45	1.41	209	1.57 (1.54)	1.49	1.49
TSOPF/TSOPF_FS_b300_c1	1.75	1.41	210	1.56 (1.61)	1.48	1.48
GHS_indef/cont-300	3.33	3.11	OOT	0.88 (0.87)	0.88	0.88
GHS_indef/ncvxqp5	1.17	1.16	OOT	2.14 (2.36)	2.14	2.14
GHS_indef/turon_m	1.02	1.02	14.6	1.11 (1.52)	1.12	1.12*
GHS_indef/d_pretok	1.03	1.02	11.4	1.18 (1.48)	1.18	1.18*
GHS_indef/ncvxqp3	1.61	1.60	OOT	2.79 (2.90)	2.72	2.72*
TSOPF/TSOPF_FS_b300_c2	2.03	1.50	OOT	1.52 (1.55)	1.44	1.44
GHS_indef/ncvxqp7	2.26	2.25	OOT	2.78 (2.77)	2.49	2.49
TSOPF/TSOPF_FS_b300_c3	2.26	1.43	OOT	1.47 (1.44)	1.35	1.35

(see columns 3 and 6 of Table V). Thus, the matching-based approach produces a very stable ordering, although for many problems there is a penalty of a (generally modest) increase in *nz*(L) and *nflop*.

4.3. Static Pivoting Results

The static pivoting strategy used in this study follows that of Duff and Pralet described in Section 2.5. We also consider combining static pivoting with restricted pivoting, which more closely follows the strategy used within PARDISO. In our tests, we follow Duff and Pralet [2007] and set the parameter *static* (see Section 2.5) to $\|\hat{A}\|\sqrt{\epsilon}$, where ϵ is the machine precision and $\hat{A}$ is the scaled matrix. Note that there is no deviation from the supplied pivoting sequence and the number of delays is zero. For the METIS ordering, the ratios of the actual to the predicted *nz*(L) and flop count are all 1's, while for the matching-based orderings the ratios are essentially the same as those in column 7 of Tables III and IV. In Table VI, we report the number of pivots that are perturbed together with the number of solves performed during refinement; the threshold $u = 0.01$ is used. The matching-based orderings require far fewer pivots to be perturbed and this in turn leads to only a few steps of refinement being required to recover accuracy (although, again, if restrictive pivoting is combined with the matching-based ordering, FGMRES fails to converge for a number of problems).

Table V.
Number of solves performed during refinement. The numbers in parentheses are for Matching using METIS_NodeND. OOT denotes the 1 hour time limit was exceeded. * indicates required accuracy not achieved. (F) indicates FGMRES was required to achieve requested accuracy.

Ordering:	METIS	METIS	MA47	Matching		Matching	Matching Restrict
u:	0.01	10^{-8}	0.01	0.01		10^{-8}	0.01
TSOPF/TSOPF_FS_b162_c1	1	2	1	0	(0)	1	1
TSOPF/TSOPF_FS_b39_c7	1	2	1	1	(1)	1	1
GHS_indef/ncvxqp1	1	1	1	1	(0)	0	1
QY/case39	1	2	1	0	(1)	1	1
GHS_indef/stokes128	0	0	0	0	(0)	0	0
GHS_indef/cvxqp3	1	1	1	1	(1)	1	1
TSOPF/TSOPF_FS_b162_c3	1	2	1	0	(0)	1	1
TSOPF/TSOPF_FS_b39_c19	1	2	1	1	(1)	1	1
GHS_indef/cont-201	1	(F) 24	1	1	(1)	1	1
TSOPF/TSOPF_FS_b162_c4	1	3	1	0	(1)	1	1
GHS_indef/bratu3d	1	(F) 44	1	0	(0)	0	0
TSOPF/TSOPF_FS_b39_c30	1	3	1	1	(1)	1	1
GHS_indef/darcy003	1	1	1	0	(0)	0	*
TSOPF/TSOPF_FS_b300	1	5	0	0	(0)	1	1
TSOPF/TSOPF_FS_b300_c1	1	4	1	0	(0)	1	1
GHS_indef/cont-300	1	(F) 12	OOT	1	(1)	1	1
GHS_indef/ncvxqp5	1	1	OOT	1	(1)	1	1
GHS_indef/turon_m	0	0	0	0	(0)	0	*
GHS_indef/d_pretok	0	0	0	0	(0)	0	*
GHS_indef/ncvxqp3	1	2	OOT	1	(1)	1	*
TSOPF/TSOPF_FS_b300_c2	1	5	OOT	0	(0)	1	1
GHS_indef/ncvxqp7	1	2	OOT	1	(1)	1	1
TSOPF/TSOPF_FS_b300_c3	1	5	OOT	0	(0)	1	1

Traditionally, the factorize phase of a sparse direct solver has generally required the greatest portion of the total execution time. It typically involves the majority of the floating-point operations (often as many as 98% of the flops are performed by the factorize phase) and is computation bound; the other phases are memory bound. Over the past decade or so, the increase in computational capacity (flops per second) has vastly outstripped the increase in memory bandwidth (bytes per second). The result of this for sparse direct solvers has been an increase in the proportion of the computation time taken by the solve phase (see Hogg and Scott [2010c] for further discussion and computational results). Thus, although static pivoting can eliminate delays, there is a potentially costly penalty to pay of subsequently requiring many solves and, since the solve does not parallelize well, this can become a bottleneck.

4.4. Constrained Ordering

Table VII presents results for the constrained ordering (obtained by post processing the METIS ordering) with $u = 0.0$ (no pivoting). Only a subset of our test problems are included because this approach is limited to saddle-point systems (12). With $u = 0.0$, there are no delays and the actual $nz(L)$ and *nflop* are equal to the predicted values. Comparing the ratios in Table VII with those in columns 2 and 5 of Tables III and IV, we see that the constrained ordering results in significantly denser factors and higher flop counts than for the corresponding unconstrained ordering and is thus unlikely to be competitive in practice.

Table VI.
Number of perturbed pivots (*nptrb*) and number of solves performed during refinement (*nitr*) for static pivoting approaches (with $u = 0.01$). * indicates required accuracy not achieved. (F) indicates FGMRES was required to achieve requested accuracy.

	nptrb			*nitr*		
Ordering:	METIS	Matching	Matching Restrict	METIS	Matching	Matching Restrict
TSOPF/TSOPF_FS_b162_c1	2026	0	0	4	1	1
TSOPF/TSOPF_FS_b39_c7	5846	0	0	6	1	1
GHS_indef/ncvxqp1	1309	9	7	2	1	1
QY/case39	7154	0	0	11	1	1
GHS_indef/stokes128	4988	5	5	*	(F) 16	(F) 16
GHS_indef/cvxqp3	2833	2	2	8	1	1
TSOPF/TSOPF_FS_b162_c3	6226	0	0	5	1	1
TSOPF/TSOPF_FS_b39_c19	14489	0	0	6	1	1
GHS_indef/cont-201	18384	0	0	(F) 24	1	1
TSOPF/TSOPF_FS_b162_c4	8549	0	0	4	1	1
GHS_indef/bratu3d	6341	0	0	4	0	0
TSOPF/TSOPF_FS_b39_c30	23895	0	0	11	1	1
GHS_indef/darcy003	19865	293	189	18	2	*
TSOPF/TSOPF_FS_b300	5622	0	0	8	1	1
TSOPF/TSOPF_FS_b300_c1	5630	0	0	10	1	1
GHS_indef/cont-300	38908	0	0	(F) 52	1	1
GHS_indef/ncvxqp5	2718	0	0	(F) 40	1	1
GHS_indef/turon_m	9018	196	171	(F) 76	1	*
GHS_indef/d_pretok	8632	551	403	(F) 60	1	*
GHS_indef/ncvxqp3	7958	24	19	*	2	*
TSOPF/TSOPF_FS_b300_c2	11634	0	0	9	1	1
GHS_indef/ncvxqp7	13741	7	7	64	1	1
TSOPF/TSOPF_FS_b300_c3	16056	0	0	(F) 40	1	1

Table VII.
Results for the constrained ordering with pivot threshold $u = 0.0$. The ratios of the number of entries in L and flop counts to those predicted for the METIS ordering is given together with the number of refinement solves.

Identifier	*nz*(L)	*nflop*	*nitr*
GHS_indef/cvxqp3	3.51	9.91	22
GHS_indef/cont-201	1.74	2.75	1
GHS_indef/darcy003	3.08	11.78	0
GHS_indef/cont-300	1.79	2.89	1
GHS_indef/turon_m	3.89	11.26	0
GHS_indef/d_pretok	3.93	11.23	0

4.5. Sparse Direct Solver Timings

So far, we have concentrated on looking at how different strategies impact the number of delayed pivots and the factor size and flop count. In this section, we illustrate how they affect computation time. Here we use Version 2.1.0 of the recent multifrontal solver `HSL_MA97`. `HSL_MA97` is used rather than `HSL_MA77` because the latter is an out-of-core solver and as such its solve phase is comparatively expensive; there are also other overheads associated with the out-of-core design that result in it being slower than `HSL_MA97`. Furthermore, `HSL_MA97` is a parallel code. Figure 1 reports complete solution times (which include the time for scaling, ordering and refinement) for `HSL_MA97` run

Table VIII.
HSL_MA97 statistics for some problems that are not in our set of tough problems. Results for METIS ordering used with no scaling, METIS ordering used with HSL_MC64 scaling, and the matching-based MC80 ordering with HSL_MC64 scaling. The number of delayed pivots and ratios of the actual number of factor entries and flops against the predicted number of entries and flops for the METIS ordering are given.

	Delayed pivots			*nz*(*L*)			*nflop*			Time (seconds)		
	None	MC64	MC80	None	MC64	MC80	None	MC64	MC80	None	MC64	MC80
GHS_indef/copter2	87	86	56	1.00	1.00	1.68	1.00	1.00	2.40	1.22	1.38	2.01
Chen/pkustk11	58	57	53	1.00	1.00	1.92	1.00	1.00	3.02	3.17	5.71	8.76
Pothen/onera_dual	160	137	85	1.00	1.00	1.61	1.00	1.00	2.39	0.91	0.98	1.22
Schenk_IBMNA/c-62	135154	594	0	4.11	1.01	1.37	12.1	1.02	1.49	28.3	2.25	8.38
Schenk_IBMNA/c-64	23361	809	0	2.72	1.01	1.34	27.8	1.05	1.85	1.97	1.47	1.08
GHS_indef/boyd2	27077	0	0	30.5	1.00	1.00	19106	1.00	1.00	57.9	47.3	37.6

on 8 processors with its default settings with the METIS ordering and the matching ordering; the problems are in the same order as in Table I. Times are wall-clock times in seconds. For the matching ordering, we omit results for $u = 10^{-8}$ because we found that they are very close to those for the default $u = 0.01$. We see that, although the matching-based ordering produces denser factors and requires more flops, for many problems it gives the best time. As it also produces few delayed pivots, this would appear to be an attractive approach. Observe that the large number of solves required for some problems for the METIS ordering with $u = 10^{-8}$ can add a significant overhead (for example, GHS_indef/bratu3d and GHS_indef/cont-300).

Finally, we briefly consider problems that are less numerically challenging and that are outside our tough test set. The problems are again taken from the University of Florida Sparse Matrix Collection; they are all symmetric indefinite and, for those that are supplied as pattern only, we generate random values in the range $[0, 1]$ for the nonzero entries. In Table VIII, we present HSL_MA97 results for the METIS ordering with no scaling (denoted by "None"), the METIS ordering with HSL_MC64 scaling (denoted by "MC64"), and the matching-based ordering with HSL_MC64 scaling (denoted by "MC80"). The times are complete solution times. The first three problems are included because they do not require scaling; they illustrate the overhead involved in scaling and in employing a matching-based ordering. The final three problems are chosen because, without scaling, they experience a large number of delayed pivots but scaling using MC64 is sufficient to eliminate most of these. For two of these problems, Schenk_IBMNA/c-62 and Schenk_IBMNA/c-64, the matching-based ordering results in significantly more entries in the factor and a higher flop count, leading to a slower solution time. Interestingly, for problem GHS_indef/boyd2, the matching-based ordering is of the same quality as the METIS ordering but the solution time is actually less for the former. Closer examination shows that, for this example, most of the time is used by METIS and, in MC80, METIS is applied to a (smaller) compressed graph, thereby reducing its runtime.

5. SUMMARY OF FINDINGS

For many problems from a wide range of applications, if a nested dissection or minimum degree ordering is used with a sparse indefinite solver and the solver is run with its default settings, few pivots will be delayed. A good scaling further limits the number of delayed pivots. This study has concentrated solely on the problems where the standard approach of threshold partial pivoting leads to a large number of delays: we have reviewed techniques designed to overcome this issue and we have employed a set of hard-to-solve symmetric indefinite systems to examine how well these techniques perform in practice. Our key findings (which are only for these tough problems and assume the system has been prescaled) are the following.

—Using a small pivot threshold parameter with the METIS ordering has a limited effect on the number of delays and, in some cases, FGMRES (which requires a large number of solves) may be needed to recover accuracy.
—For many problems, the `MA47` ordering results in fewer delayed pivots than the METIS ordering but is less efficient (much denser factors and higher flop counts).
—Static pivoting leads to no delays but, when it is combined with threshold partial pivoting and a nested dissection ordering, a large number of solves can be needed to recover accuracy and, in a few cases, we were not able to achieve the requested accuracy using FGMRES. Furthermore, knowledge of the inertia may be lost.
—The constrained ordering of Bridson without pivoting is limited to saddle-point problems and requires the $(1, 1)$ block to be semi-definite. If this is satisfied, the fill-in and flop counts are generally significantly greater than for the unconstrained METIS ordering.
—Matching-based orderings can substantially reduce the number of delayed pivots. In many cases, the computed factors are sparser and flop counts are less than for METIS. Furthermore, these orderings can be successfully used with a small pivot threshold or with static pivoting.
—Although for many examples restrictive pivoting combined with the matching-based ordering works well, it is not robust.

Thus to limit delayed pivots, our general recommendation is that a matching-based ordering should be used. This option is now built into `HSL_MA97`; a matching-based ordering may also be precomputed using `HSL_MC80` and then passed into other direct solvers, such as `HSL_MA77`. If the inertia is not required, incorporating static pivoting removes all delays and, when combined with the matching-based ordering, in our tests gave the required accuracy after refinement. We emphasize, however, that the matching-based approach is only recommended for tough problems such as those we have reported on in this study. As illustrated in Table VIII, if standard threshold partial pivoting gives no delays, scaling may be unnecessary and using a matching-based ordering is not desirable as it can be expensive to compute and may result in denser factors and higher flop counts. For other problems, scaling may be sufficient to reduce the number of delays and, again, a matching-based ordering is then not necessary.

ACKNOWLEDGMENTS

Our thanks go to our colleague Iain Duff for commenting on a draft of this article. We are also grateful to the three anonymous referees for carefully reading and commenting on our manuscript.

REFERENCES

Amestoy, P. R., Davis, T. A., and Duff, I. S. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl. 17*, 886–905.

Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Koster, J. 1999. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl. 23*, 1, 15–41.

Arioli, M. and Duff, I. S. 2009. Using FGMRES to obtain backward stability in mixed-precision. *Electron. Trans. Numer. Anal. 33*, 31–44.

Ashcraft, C., Grimes, R. G., and Lewis, J. G. 1999. Accurate Symmetric Indefinite Linear Equation Solvers. *SIAM J. Matrix Anal. Appl. 20*, 2, 513–561.

Bridson, R. 2007. An ordering method for the direct solution of saddle-point matrices. Preprint http://www.cs.ubc.ca/~rbridson/kktdirect/.

Bunch, J. R. and Kaufman, L. 1977. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp. 31*, 163–179.

Davis, T. A. and Hu, Y. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw. 38*, 1. Article 1.

de Niet, A. C. and Wubs, F. W. 2009. Numerically stable LDLT -factorization of F-type saddle point matrices. *IMA J. Numer. Anal. 29*, 208–234.

Duff, I. S. 2004. MA57—A new code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw. 30*, 118–154.

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

Duff, I. S., Gould, N. I. M., Reid, J. K., Scott, J. A., and Turner, K. 1991. Factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal. 11*, 181–2044.

Duff, I. S. and Koster, J. 2001. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl. 22*, 4, 973–996.

Duff, I. S. and Pralet, S. 2005. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Anal. Appl. 27*, 313–340.

Duff, I. S. and Pralet, S. 2007. Towards a stable mixed pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems. *SIAM J. Matrix Anal. Appl. 29*, 1007–1024.

Duff, I. S. and Reid, J. K. 1983. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw. 9*, 302–325.

Duff, I. S. and Reid, J. K. 1992. MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Tech. rep. R-10533. Computer Science and Systems Division, AERE.

Duff, I. S. and Reid, J. K. 1996. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw. 22*, 227–257.

Fourer, R. and Mehrotra, S. 1993. Solving symmetric indefinite systems in an interior-point method for linear programming. *Math. Program. 62*, 15–39.

George, A. 1973. Nested dissection of a regular finite-element mesh. *SIAM J. Numer. Anal. 10*, 345–363.

Gill, P. E., Saunders, M. A., and Shinnerl, J. R. 1996. On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM J. Matrix Anal. Appl. 17*, 35–46.

Gupta, A. 2000. WSMP: Watson Sparse Matrix Package (Part-I: Direct solution of symmetric sparse systems). Tech. rep. RC 21886, IBM T. J. Watson Research Center, Yorktown Heights, NY. http://www.cs.umn.edu/∼agupta/wsmp.

Hagemann, M. and Schenk, O. 2006. Weighted matchings for preconditioning symmetric indefinite linear systems. *SIAM J. Sci. Comput. 28,* 403–420.

Hogg, J. D., Reid, J. K., and Scott, J. A. 2010. Design of a multicore sparse Cholesky factorization using DAGs. *SIAM J. Sci. Comput. 32,* 3627–3649.

Hogg, J. D. and Scott, J. A. 2008. The effects of scalings on the performance of a sparse symmetric indefinite solver. Tech. rep. RAL-TR-2008-007, Rutherford Appleton Laboratory.

Hogg, J. D. and Scott, J. A. 2010a. A fast and robust mixed precision solver for the solution of sparse symmetric linear systems. *ACM Trans. Math. Softw. 37*. Article 17.

Hogg, J. D. and Scott, J. A. 2010b. An indefinite sparse direct solver for large problems on multicore machines. Tech. rep. RAL-TR-2010-011, Rutherford Appleton Laboratory.

Hogg, J. D. and Scott, J. A. 2010c. A note on the solve phase of a multicore solver. Tech. rep. RAL-TR-2010-007, Rutherford Appleton Laboratory.

Hogg, J. D. and Scott, J. A. 2011. HSL MA97: a bit-compatible multifrontal code for sparse symmetric systems. Tech. rep. RAL-TR-2011-024, Rutherford Appleton Laboratory.

Hogg, J. D. and Scott, J. A. 2012a. New parallel sparse direct solvers for engineering applications. Tech. rep. RAL-P-2012-001, Rutherford Appleton Laboratory.

Hogg, J. D. and Scott, J. A. 2012b. A study of pivoting strategies for tough sparse indefinite systems. Tech. rep. RAL-TR-2012-009, Rutherford Appleton Laboratory.

HSL. 2013. A collection of Fortran codes for large-scale scientific computation. http://www.hsl.rl.ac.uk/.

Karypis, G. and Kumar, V. 1998. METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices - Version 4.0. http://www-users.cs.umn.edu/∼karypis/metis/.

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

LIU, J. W. H. 1985. Modification of the Minimum-Degree algorithm by multiple elimination. *ACM Trans. Math. Softw. 11*, 2, 141–153.

MUMPS. 2011. MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. (2011). http://graal.ens-lyon.fr/MUMPS/.

REID, J. K. AND SCOTT, J. A. 2008. An efficient out-of-core sparse symmetric indefinite direct solver. Tech. rep. RAL-TR-2008-024, Rutherford Appleton Laboratory.

REID, J. K. AND SCOTT, J. A. 2009. An out-of-core sparse Cholesky solver. *ACM Trans. Math. Softw. 36*, 2. Article 9.

REID, J. K. AND SCOTT, J. A. 2011. Partial Factorization of a Dense Symmetric Indefinite Matrix. *ACM Trans. Math. Softw. 38*. Article 10.

SAAD, Y. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput. 14*, 461–469.

SCHENK, O. AND GÄRTNER, K. 2004. Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO. *J. Future Generation Comput. Syst. 20*, 475–487.

SCHENK, O. AND GÄRTNER, K. 2006. On fast factorization pivoting methods for symmetric indefinite systems. *Electron. Trans. Numer. Anal. 23*, 158–179.

SCHENK, O., WÄACHTER, A., AND HAGEMANN, M. 2007. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale non-convex interior-point optimization. *Comput. Optim. Appl. 36,* 321–341.

SCOTT, J. A. 2009. A note on a simple constrained ordering for saddle-point systems. Tech. rep. RALTR-2009-007, Rutherford Appleton Laboratory, Chilton, Oxfordshire, UK.

TŮMA, M. 2002. A note on the LDLT decomposition of matrices from saddle-point problems. *SIAM J. Matrix Anal. Appl. 23*, 903–925.

TINNEY, W. F. AND WALKER, J. W. 1967. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE 55*, 1801–1809.

WÄCHTER, A. AND BIEGLER, L. T. 2006. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Math. Program. 106*, 1, 25–57.

Received August 2012; revised February 2013; accepted April 2013