AN ORDERING METHOD FOR THE DIRECT SOLUTION OF SADDLE-POINT MATRICES[*]

R. BRIDSON[†]

Abstract. An ordering method and accompanying factorization for the direct solution of saddle-point matrices is presented. A simple constraint on ordering together with an assumption on the rank of parts of the matrix are sufficient to guarantee the existence of the LDL^T factorization, stability concerns aside. In fact, D may be taken to be a diagonal matrix with ± 1 along the diagonal, and be fully determined prior to factorization, giving rise to a "signed Cholesky" factorization. A modified minimum-degree-like algorithm which incorporates this constraint is demonstrated, along with a simple algorithm to modify an existing fill-reducing ordering to respect the constraint. While a stability analysis is lacking, numerical experiments indicate that this is generally sufficient to avoid the need for numerical pivoting during factorization, with clear benefits for performance possible. For example, a highly efficient Cholesky factorization routine, based on separate symbolic and numerical phases and possibly exploiting supernodes, can be easily adapted to this more general class of matrices.

Key words. saddle-point matrices, KKT systems, symmetric indefinite matrices, direct solvers, LDL^T factorization, minimum degree ordering

AMS subject classifications. 65F05

1. Introduction.

Many applications in scientific computing require the solution of large linear systems of equations involving symmetric indefinite sparse matrices of a special 2×2 block structure:

$$K = \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} \tag{1.1}$$

Here the matrix K is partitioned into a symmetric positive semi-definite A, a possibly rectangular B and its transpose, and the negative of a symmetric positive semi-definite C which is often simply zero. I assume without loss of generality that the dimension of C is less than or equal to the dimension of A. Matrices of this type, sometimes termed saddle-point, KKT or equilibrium matrices, arise naturally in constrained optimization, mixed FEM discretizations, incompressible fluid flow and constrained dynamics to name just a few areas. See the article of Benzi et al. [3] for an excellent review of these problems and existing solution techniques, both direct and iterative. Fully exploiting the special structure of such matrices in direct solvers remains an open problem.

Codes that handle general indefinite matrices can naturally be used for saddle-point problems. However, they typically either rely on some form of numerical pivoting—which poses problems for avoiding fill-in—or need to modify the matrix being factorized to avoid breakdown—making the factorization only approximate, necessitating some kind of iteration and raising the possibility of slow convergence in ill-conditioned cases.

In the special case of a symmetric quasidefinite matrix, i.e. where A and C are both positive definite, Vanderbei [15] shows that in fact the LDL^T factorization exists for any ordering, and Gill et al. [9] provide a stability analysis indicating that B

[*]This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

[†]Dept. Computer Science, The University of British Columbia, Vancouver B.C. V6T 1Z4, Canada, rbridson@cs.ubc.ca

should not be too large compared to A and C and that both A and C should be well-conditioned. Under these restrictions, the full matrix may be ordered to reduce fill-in without concern over indefiniteness, using standard algorithms designed for fill-reduction of SPD matrices. However, this doesn't apply in the common case where C is singular (often just zero).

Another special case of practical interest is the class of $\mathcal{F}$-matrices, where each column of B has exactly two nonzero entries which sum to zero, and $C = 0$. These arise, for example, in Stokes flow problems. For this class, Tůma [14] demonstrated that a fill-reducing ordering of the entire matrix can be efficiently modified to guarantee the existence of the LDL^T factorization, apparently without greatly compromising the quality of the ordering. De Niet and Wubs [11] tackle the same class of matrices with a different approach, first ordering the A-part of the matrix (though basing fill on the structure of $A + B^TB$), then carefully interleaving the remaining nodes into this ordering in a way that guarantees existence of the factorization. De Niet and Wubs further prove that the factorization of a well-scaled $\mathcal{F}$-matrix is stable for any ordering where it exists. Unfortunately, neither of these have yet been adapted to fully general B (and in the latter case, may have serious difficulties if B has a dense row, making $A + B^TB$ dense).

This paper demonstrates a simple constraint on orderings for saddle-point matrices where A is definite and B has full rank, sufficient to guarantee the existence of the LDL^T factorization. I demonstrate how a minimum-degree-like ordering algorithm can easily be modified to respect this constraint, generating high quality fill-reducing orderings which avoid the need for numerical pivoting. As an alternative approach, I demonstrate how an arbitrary ordering can be post-processed to respect the constraint. Finally, once such fill-reducing orderings are available, near-Cholesky-like efficiency is possible for the direct solver; I present timings with a proof-of-concept code for a range of test matrices. While I do not have a stability analysis for the method, it appears in practice that at worst one step of residual refinement is required to attain full accuracy.

2. Preliminaries.

For the rest of the paper I will restrict the class of matrices to those where A is definite and B has full row rank. This is a sufficient condition for the existence of the LDL^T factorization of the unpermuted matrix, and is frequently met in applications. While the proof-of-concept code that has been developed for this project may and often does work if these restrictions on rank are slightly relaxed, the underlying theory suggests it may fail if these assumptions are broken. For numerical stability of the method further conditions are no doubt needed on A, B and C; however, a precise understanding of stability is left for future work, perhaps in the vein of Gill et al.'s analysis of the quasidefinite case [9]. I conjecture that if A and the negative Schur complement $C+B^TA^{-1}B$ are not ill-conditioned and are of comparable magnitudes, then the factorization (with the orderings presented in this paper) will be stable.

I assume the reader is familiar with the graph-theoretic approach to sparse matrix ordering, for example as outlined in Davis' recent book [7]. I will refer to the nodes in the graph of the matrix corresponding to diagonal entries of A as A-nodes, and the remaining nodes as C-nodes.

It is clear that if all of the A-nodes are ordered before the C-nodes, the block structure of the matrix is preserved and, under our full rank assumptions, the LDL^T

factorization of the (possibly permuted) K must exist:

$$K = \begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} D_{11} & 0 \\ 0 & D_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix}$$

In particular,

$$L_{11} D_{11} L_{11}^T = A$$

is the LDL^T form of the Cholesky factorization of the positive definite A (so D_{11} has a positive diagonal), the off-diagonal block is

$$L_{21} = B L_{11}^{-T} D_{11}^{-1},$$

and

$$L_{22} D_{22} L_{22}^T = -C - BA^{-1}B^T$$

is the LDL^T form of the Cholesky factorization of the negative definite Schur complement (so D_{22} has a negative diagonal).

One well-known problem with this approach is that while fill-in may effectively be avoided in L_{11} and perhaps even L_{21}, the Schur complement will often be fully dense, giving catastrophic fill-in no matter how the C-nodes are ordered. Unfortunately, unlike in the SPD case, a fill-reducing ordering of the entire matrix may destroy the existence of the factorization—for example, if a C-node with a zero diagonal entry is ordered first. A balance must be struck between A-nodes preceding C-nodes and interleaving the two types where helpful for avoiding fill-in.

I will make use of two other results. The first is the inverse of saddle-point matrices satisfying the rank conditions:

$$\begin{pmatrix} A & B^T \\ B & -C \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} - A^{-1}B^T S^{-1} B A^{-1} & A^{-1} B^T S^{-1} \\ S^{-1} B A^{-1} & -S^{-1} \end{pmatrix} \tag{2.1}$$

where the negative Schur complement S is given by:

$$S = C + BA^{-1}B^T \tag{2.2}$$

Observe that since C is positive semi-definite, A is positive definite, and B has full row rank, this matrix S must be positive definite.

The second result is the calculation of row i of the L factor and entry i along the diagonal of D, for a permuted K, assuming the factorization of the first $i-1$ rows exists. Letting $\hat{K}$ be the first $(i-1) \times (i-1)$ principal submatrix of K and $k_{i\diamondsuit}$ be the first $i-1$ entries of row i,

$$\begin{pmatrix} \hat{K} & k_{i\diamondsuit}^T \\ k_{i\diamondsuit} & k_{ii} \end{pmatrix} = \begin{pmatrix} \hat{L} & 0 \\ l_{i\diamondsuit} & 1 \end{pmatrix} \begin{pmatrix} \hat{D} & 0 \\ 0 & d_{ii} \end{pmatrix} \begin{pmatrix} \hat{L}^T & l_{i\diamondsuit}^T \\ 0 & 1 \end{pmatrix}. \tag{2.3}$$

Note this takes L to have unit diagonal; later I will show it may be attractive to rescale this L to $L|D|^{-1/2}$ and similarly rescale D to get $|D| = I$. Equating terms in the matrix equation above, row i of the L factor is:

$$l_{i\diamondsuit} = k_{i\diamondsuit} \hat{L}^{-T} \hat{D}^{-1} \tag{2.4}$$

and the i'th pivot on the diagonal of D is:

$$\begin{aligned} d_{ii} &= k_{ii} - l_{i\Diamond}\hat{D}l_{i\Diamond}^T \\ &= k_{ii} - k_{i\Diamond}\hat{K}^{-1}k_{i\Diamond}^T \end{aligned} \tag{2.5}$$

In particular, notice that d_{ii} does not depend on the ordering of the previous $i-1$ nodes: the scalar $k_{i\Diamond}\hat{K}^{-1}k_{i\Diamond}^T$ is invariant under permutations. For the factorization to proceed, it is sufficient for d_{ii} to be nonzero: this paper provides a sufficient constraint on orderings to guarantee this. For stability, d_{ii} should further be well bounded away from zero and $l_{i\Diamond}$ shouldn't grow too large; I do not address these issues in the present paper.

3. The Ordering Constraint.

The proposed constraint is that any C-node can only be ordered after all its A-node neighbours in the graph of K have been ordered. In other words, an ordering should respect the *partial* ordering implied by the directed graph of B.

Note that the first node to be eliminated under this constraint must be an A-node, with pivot $d_{11} = A_{11} > 0$. If a C-node were to come first, it would have to have no A-node neighbours due to the constraint: this would imply a zero row in B, violating the full rank assumption.

Assuming the first $i-1$ steps of factorization have succeeded, proceed inductively to the i'th pivot $d_{ii} = k_{ii} - k_{i\Diamond}\hat{K}^{-1}k_{i\Diamond}^T$. Partition the previous nodes into A- and C-subsets so the i'th principal submatrix appears as:

$$\left(\begin{array}{c|c} \hat{K} & k_{i\Diamond}^T \\ \hline k_{i\Diamond} & k_{ii} \end{array}\right) = \left(\begin{array}{cc|c} \hat{A} & \hat{B}^T & u^T \\ \hat{B} & -\hat{C} & v^T \\ \hline u & v & k_{ii} \end{array}\right) \tag{3.1}$$

Observe that $\hat{A}$ is a principal submatrix of the positive definite matrix A, thus is positive definite itself—and in fact, $\hat{A}$ is at least as well conditioned as A. Also observe that the ordering constraint implies that all nonzeros in the rows of B for previously ordered C-nodes must appear in $\hat{B}$: otherwise there would be a C-node ordered before i with an A-node neighbour ordered at i or later. Since B is assumed to have full row rank, $\hat{B}$ also has full row rank: $\hat{B}$ is a subset of the rows of B with possibly some fully zero columns missing. Again $\hat{B}$ is at least as well conditioned as B. Therefore this partitioning shows that $\hat{K}$ satisfies our rank condition, and has an inverse expressed as in equation (2.1).

Case 1: node i is an A-node. Due to the ordering constraint, i must not have any previously ordered C-neighbours. Therefore $v = 0$ in equation (3.1). Using the form of the inverse in equation (2.1), the pivot is:

$$\begin{aligned} d_{ii} &= k_{ii} - (u \quad 0)\begin{pmatrix} \hat{A} & \hat{B}^T \\ \hat{B} & -\hat{C} \end{pmatrix}^{-1}\begin{pmatrix} u^T \\ 0 \end{pmatrix} \\ &= k_{ii} - u(\hat{A}^{-1} - \hat{A}^{-1}\hat{B}^T\hat{S}^{-1}\hat{B}A^{-1})u^T \\ &= (k_{ii} - u\hat{A}^{-1}u^T) + u\hat{A}^{-1}\hat{B}^T\hat{S}^{-1}\hat{B}A^{-1}u^T \end{aligned} \tag{3.2}$$

Note that

$$\left(\begin{array}{c|c} \hat{A} & u^T \\ \hline u & k_{ii} \end{array}\right)$$

is also a principal submatrix of the positive definite A and hence is positive definite, thus its final pivot is positive: $k_{ii} - u\hat{A}^{-1}u^T > 0$, i.e. the first term of d_{ii} is positive. Also recall that the negative Schur complement S here is positive definite, implying the other term is positive too. Therefore $d_{ii} > 0$. In fact, it's bounded away from zero at least as well as the pivots of A alone are, and including C-nodes earlier in the ordering can only further stabilize it.

Case 2: node i is a C-node. In this case, simply join the i'th row and column to the other C-node parts of the partition in equation (3.1). This $i\times i$ matrix also satisfies the rank condition, following the same argument as above, and thus has inverse as expressed in equation (2.1). The i'th pivot is well known to be the reciprocal of the (i, i) entry of the inverse of the $i \times i$ principal submatrix, which in this case comes from the diagonal of the negative definite $-\hat{S}^{-1}$. Thus $d_{ii} < 0$.

As both cases give nonzero pivots, by induction this ordering constraint ensures that the LDL^T factorization exists. Moreover the pivots associated with A-nodes are guaranteed to be positive and the pivots associated with C-nodes are guaranteed to be negative: we know the signs of the diagonal entries of D in advance. By rescaling $L \leftarrow L|D|^{1/2}$ and $D \leftarrow \mathrm{sign}(D) = \mathrm{diag}(\pm 1)$, the diagonal matrix is fully determined in advance by the structure of the problem, independent of numerical values. I refer to this as the "signed Cholesky" factorization of K, which I will show later allows a direct solver to gain additional efficiency.

4. Constraining Ordering Algorithms.

One of the most successful strategies available for reducing fill is the greedy approach of Minimum Degree and relatives. The heart of these algorithms is maintaining a priority queue of uneliminated nodes, sorted by some cost metric related to fill-in (such as degree, external degree, deficiency, or approximations thereof). At each step a node of minimum cost is eliminated, the graph is updated, and costs of affected nodes are updated. One particularly important enhancement to this basic approach is the detection and compression of supernodes in the graph (cliques of nodes with identical neighbour sets). Many other enhancements are described in more detail in George and Liu's review [8], with more recent developments due to Amestoy et al. [1], Rothberg and Eisenstat [12], and Ng and Radhavan [10]; see also Davis' book [7] for more implementation details.

Modifying these methods to respect the ordering constraint introduced here is straightforward. Initially, rather than putting all nodes in the priority queue, only A-nodes are made available for elimination. Once an A-node is eliminated, its C-node neighbours are tested to see if all of their A-node neighbours have now been eliminated, and if so, are added to the priority queue. Nodes that haven't yet been added to the priority queue are not considered for inclusion in supernodes (and similarly do not take part in other enhancements to the basic algorithm, such as mass elimination).

Occasionally sparse matrices may possess a few fairly dense rows/columns; it is typically far more efficient to detect and strip out these nodes prior to running minimum degree, and then order them last (as AMD, for example, does [2]). Note that if a dense A-node is stripped out in this fashion, then to meet the constraint all its C-node neighbours must follow. However, if it has very many of these neighbours—as is likely given its density—this may cause catastrophic fill-in. Thus it may be worthwhile to ignore the constraint for these nodes, and trust that the row rank of B isn't hurt if these columns are removed.

Typically, instead of using the raw output from minimum degree for factorization, a postordering of the elimination tree is used instead. This simplifies symbolic operations, improves data locality, and allows for supernodes to be exploited. Notice that if the ordering constraint was met initially, every C-node must be an ancestor of its A-node neighbours in the elimination tree, thus a postordering of the tree will not violate the constraint either.

At least for problems based on an underlying mesh, nested dissection and generalizations based on graph partitioning generally outperform Minimum Degree, particularly when hybridized with Minimum Degree. In this paper, I do not consider this class of algorithms, though note in principle it should be straightforward to constrain the search for good vertex separators to only include A-nodes if all their C-node neighbours are also included. Ordering such a separator last would then be compatible with the constraint in this paper.

A cheap alternative to the preceding approach of directly incorporating the constraint into the ordering algorithm is to post-process an arbitrary fill-reducing ordering to satisfy the constraint. This is done by accepting the given sequence of nodes, except delaying any C-nodes until all their A-node neighbours have appeared (again, with perhaps an unsafe option to ignore the constraint for dense A-nodes).

5. The Signed Cholesky Factorization.

Once an ordering respecting the constraint is found (which I will now assume is in fact a post-ordering of the elimination tree—as the previous section argues, this doesn't violate the constraint), and assuming no stability concerns, LDL^T factorization may proceed without need for numerical pivoting.

For very sparse matrices, Davis [7] reports that a point-wise up-looking algorithm for Cholesky factorization may be the fastest choice. The square-root free version of this algorithm may be used unchanged for this case, and no more need be said.

In slightly more dense scenarios, factorizations that can exploit dense subblocks are generally favoured. Both left-looking and right-looking multifrontal variants can handle supernodes (columns of L with identical nonzero structure below the diagonal) with dense operations. Typical fill-reducing orderings generate many such supernodes that can be detected in the symbolic factorization stage. In the context of this paper, a code that computes the standard LDL^T ("square-root free") form of Cholesky factorization can be trivially adapted to the saddle-point case by changing the call to a dense Cholesky factorization of supernodal blocks on the diagonal to an indefinite solver, such as LAPACK's Bunch-Kaufman routine. In fact, some codes such as PARDISO already offer this as an option for general symmetric indefinite matrices. (Of course, without the guarantee of existence offered by the orderings in this paper, perturbation of the diagonal blocks may be necessary to avoid zero pivots, which in turn necessitates several steps of residual refinement [13].)

However, more can be done. In practice, I found supernodes often have a mix of A- and C-nodes; without modifying the nonzero structure of L or the structure of its elimination tree, and without violating the constraint, the ordering can be modified to put all of the A-nodes within the supernode first. Then each such supernode can be efficiently split in two, an A-node only part and a C-node only part. An A-supernode produces only positive pivots, and therefore its diagonal block is positive definite and may be factored with a more efficient dense Cholesky code. Similarly a C-supernode only produces negative pivots, and therefore the *negative* of its diagonal block is positive definite and may also be factored with an efficient dense Cholesky code. This naturally produces the signed Cholesky factorization introduced earlier, and allows an efficient supernodal sparse Cholesky implementation to be adapted to the saddle-point case simply by flipping sign bits for C-supernodes at a few choice points in the code.

Boeing/bcsstm36	GHS_indef/d_pretok	GHS_indef/qpband
FIDAP/ex4	GHS_indef/darcy003	GHS_indef/sit100
FIDAP/ex14	GHS_indef/dtoc	GHS_indef/stokes64
GHS_indef/aug3dcqp	GHS_indef/exdata_1	GHS_indef/stokes128
GHS_indef/boyd1	GHS_indef/k1_san	GHS_indef/tuma1
GHS_indef/boyd2	GHS_indef/laser	GHS_indef/tuma2
GHS_indef/brainpc2	GHS_indef/mario001	GHS_indef/turon_m
GHS_indef/cont-201	GHS_indef/mario002	Grund/meg4
GHS_indef/cont-300	GHS_indef/ncvxqp9	stokes_2d_250
GHS_indef/cvxqp3	GHS_indef/olesnik0	stokes_2d_500

FIG. 6.1. *Saddle-point test matrices. All but the last two are drawn from the University of Florida Sparse Matrix Collection [6]; stokes_2d_250 and stokes_2d_500 are discretizations of the 2D Stokes problem on larger square grids.*

6. Results. To demonstrate the ordering constraint, I implemented a prototype code KKTDirect (available online [4]). While this implementation is far from optimal—in particular the KKTDirect version of external minimum degree is several times slower than the AMD code of Amestoy et al. [2] for SPD matrices, and the symbolic factorization stage currently uses a simple but inefficient algorithm—it is fast enough to produce meaningful comparisons.

My test set of matrices was primarily drawn from the University of Florida Sparse Matrix Collection [6], with some benchmarking done on a subset of the SPD matrices (160 matrices, consisting of those with at least 1000 nonzeros in the Cholesky factor but still fit in memory of the test platform), and more tests on those that appeared to be saddle-point matrices satisfying the rank constraints of the paper. In addition, matrices for solving the 2D Stokes problem on a square staggered grid with the usual central finite difference formulas at 250×250 and 500×500 resolutions were included. See table (6.1) for a list of these 30 matrices.

The test platform was a 2GHz PowerPC G5 Macintosh with 8GB RAM, using the vendor-supplied vecLib BLAS and LAPACK. It was found that limiting supernodes to a maximum size of 63 usually provided the best performance.

6.1. Testing Ordering Quality. For SPD problems of reasonable size (where AMD took at least 0.5 seconds to run on the test platform, but where numerical factorization didn't run out of memory), the exact external degree algorithm ran on average 10 times slower than AMD but never worse than 15 times. Initial profiling of the code suggests many avenues for optimization, but I felt as it stood the code runs fast enough (in particular, significantly faster than the actual factorization) and consistently enough that the numerical tests are reasonable. In terms of fill, the exact external degree algorithm produced 2% more fill than AMD in the mean (and only 1% more as the median ratio), at worst 23% more (for matrix UTEP/Dubcova3), and at best 32% less fill (for matrix Mulvey/finan512), thus I felt confident in basing comparisons on the quality of this code's ordering.

As an aside and as many authors have observed, the main bottleneck of minimum-degree-like algorithms is the recomputation of node costs (e.g. external degree). It can be observed that often a node will have its cost recomputed dozens or even hundreds of times before it actually is eliminated as a node of minimum cost—leading one to question if all of those recomputations are necessary. While the approach taken in AMD is to make the recomputation as efficient as possible, KKTDirect provides an alternative to cut out a lot of these unnecessary recomputations. When a node is eliminated, its neighbours' costs are replaced with a cheap lower bound (for external degree, their current cost minus the degree of the eliminated node) and a flag is set to indicate this is inexact. When the next node of minimum cost is selected from the priority queue of uneliminated nodes, this flag is tested. Only if the cost is exact does the node get eliminated; otherwise its cost is recomputed exactly, the priority queue is updated, and a new (or possibly the same) node of minimum cost is selected. This can significantly improve performance without adversely affecting ordering quality.

Proceeding to the 30 saddle-point test matrices, I ran three different ordering algorithms to get an estimate of the penalty imposed by the constraint, and the relative merits of incorporating constraints directly into the ordering or as a post-process:

- the plain external degree algorithm (as if the matrices were SPD),
- the external degree algorithm incorporating constraints (excluding dense A-nodes),
- and the plain external degree ordering post-processed to respect constraints (again, excluding dense nodes from the constraint).

Including the dense nodes in the constraint in some cases caused disastrous fill-in, in particular for GHS_indef/boyd2 which contains two nearly dense columns in B, but was found to be unnecessary for the factorization to succeed, thus I ran the tests without constraints applied to dense nodes.

For six problems the constraints had zero effect on fill:

- Boeing/bcsstm36,
- GHS_indef/boyd1,
- GHS_indef/dtoc,
- GHS_indef/exdata_1,
- GHS_indef/laser,
- GHS_indef/qpband.

In three cases ordering with constraints actually reduced fill:

- GHS_indef/aug3dcap,
- GHS_indef/boyd2,
- GHS_indef/brainpc2,

which might have something to do with the special structure (A is diagonal) of these problems.

For the remaining 21 problems, the fill penalty (relative to the unconstrained ordering which assumed the matrix SPD) ranged from a fraction of a percent to an outlier factor of 7.5 for post-processing the ordering of GHS_indef/cvxqp3, a particularly challenging matrix for direct solvers. Post-processing an ordering after the fact usually generated better results: it outperformed external degree with incorporated constraints 14 times out of 21. However, the two methods were quite competitive: on the problems for which post-processing was superior, it was between 3% to 40% better, with a mean 20% improvement; on the other problems, it was up to 185% worse, with a mean 50% degradation. Given that both strategies are relatively cheap compared to the cost of factorization, it may well make sense to try both and select the best.

6.2. Speed of Numerical Factorization. To calibrate the speed of the new factorization code, I compared my supernodal signed Cholesky code to CHOLMOD [5] a well established code that takes the same left-looking supernodal approach, on the SPD problems. On the reasonably sized problems—those for which the codes didn't run out of memory and for which CHOLMOD required at least 0.5 seconds to numerical factor the matrix—the two codes were evenly matched, one never running more than twice as fast as the other, and on average performing identically. This is to be expected as the bulk of computation time for both codes is spent in the same dense BLAS/LAPACK kernels. I excluded the time for symbolic factorization from the tests as currently the new code uses a slow, naïve algorithm; however, as this step is identical for SPD and saddle-point matrices (being a purely structural computation), there is nothing significant to test.

To test the gains in speed possible for the saddle-point problems, I ran the signed Cholesky code with the external degree ordering including constraints (not modified after the fact), as well as UMFPACK with the default options. While UMFPACK is a partial pivoting LU code, and thus ignores symmetry, it is a well established and freely available code appropriate for indefinite problems (and exploits level 3 BLAS for high efficiency)—and indeed, for some problems it turns out unsymmetric permutations are an advantage. To take an extreme example, if B is square and invertible and $C = 0$, an unsymmetric code can reduce the saddle-point matrix to block triangular form with obvious benefits.

Looking at the 18 problems for which the methods took more than 0.5 seconds to compute the numerical factorization on this platform, there is a wide spread of results. In the worst case (GHS_indef/cvxqp3), the signed Cholesky numerical factorization was 4.3 times slower than UMFPACK; the extremely large bandwith of this problem and the accompanying enormous amounts of fill in the factors for both methods probably indicate this is simply a very challenging problem for any direct solver. For three other problems (GHS_indef/exdata_1, GHS_indef/mario001, GHS_indef/sit100) the new signed Cholesky code was 35% to 60% slower than UMFPACK, again possibly due to unusually dense components or very large bandwidths. However, for the remaining 14 problems, the signed Cholesky code was on average twice as fast, and in some cases orders of magnitude faster: for GHS_indef/boyd1 and GHS_indef/boyd2 the signed Cholesky approach was 1000 and 750 times faster respectively.

6.3. Stability and Accuracy. In terms of stability of the factorization, the signed Cholesky factorization succeeded for all problems, despite several being singular (e.g. for the larger Stokes problems, there is a one dimensional null-space consisting of constant pressure values)—for these problems the right-hand-side was chosen to be consistent, and one of infinitely many solutions was found. Moreover, the computed solutions were relatively accurate for all but four of the nonsingular problems (GHS_indef/boyd2, GHS_indef/cont-300, GHS_indef/cvxqp3, GHS_indef/k1_san) and in all of these cases one step of residual refinement was sufficient to recover full accuracy. Interestingly, even the initial inaccurate solution to GHS_indef/k1_san was several orders of magnitude more accurate than that computed by UMFPACK.

7. Conclusions.

This paper presented a simple ordering constraint sufficient to guarantee the LDL^T factorization of a large class of saddle-point matrices, and gave two possible implementations, either directly incorporating it into a minimum-degree-like ordering or post-processing an arbitrary existing ordering. Both were found to be effective, and in the test problems often did not incur much penalty compared to the idealized situation of a SPD matrix of with the same sparsity pattern.

Since the ordering constraint permits the solver to predict the signs of D in advance, it was shown how a supernodal code can rely on dense Cholesky factorizations (i.e. avoiding numerical pivoting even within supernodes). This permits the full efficiency of a Cholesky direct solver to be retained for more general saddle-point matrices. Comparisons against a pivoting code showed that for many (though not all) problems this can lead to significant performance gains.

REFERENCES

[1] P. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, 1996.

[2] Patrick R. Amestoy, Enseeiht-Irit, Timothy A. Davis, and Iain S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388, 2004.

[3] Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, pages 1–137, 2005.

[4] Robert Bridson. KKTDirect: a direct solver package for saddle-point matrices. available from http://www.cs.ubc.ca/∼rbridson/kktdirect.

[5] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 8xx: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. Technical report, University of Florida CISE Tech. Report, 2006.

[6] T. A. Davis. University of Florida Sparse Matrix Collection. available from http://www.cise.ufl.edu/research/sparse/matrices, NA Digest 92(42) 1994, NA Digest 96(28) 1996, and NA Digest 97(23) 1997.

[7] T. A. Davis. *Direct methods for sparse linear systems.* SIAM, 2006.

[8] A. George and W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989.

[9] Philip E. Gill, Michael A. Saunders, and Joseph R. Shinnerl. On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM J. Matrix Anal. Appl.*, 17(1):35–46, 1996.

[10] Esmond G. Ng and Padma Raghavan. Performance of greedy ordering heuristics for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(4):902–914, 1999.

[11] Arie C. De Niet and Fred W. Wubs. Numerically stable LDL^T-factorization of $\mathcal{F}$-type saddle point matrices. Technical report, Rijksuniversiteit Groningen, 2005.

[12] E. Rothberg and S. C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM J. Matrix Anal. Appl.*, 19(3):682–695, 1998.

[13] O. Schenk and K. Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.

[14] Miroslav Tůma. A note on the LDL^T decomposition of matrices from saddle-point problems. *SIAM J. Matrix Anal. Appl.*, 23(4):903–915, 2002.

[15] Robert J. Vanderbei. Symmetric quasidefinite matrices. *SIAM J. Opt.*, 5(1):100–113, 1995.