RAL-TR-2006-001

CCLRC

Towards an automatic ordering for a symmetric sparse direct solver

Iain S. Duff and Jennifer A. Scott

December 2005

Council for the Central Laboratory of the Research Councils

© Council for the Central Laboratory of the Research Councils

Enquires about copyright, reproduction and requests for additional copies of this report should be addressed to:

Library and Information Services
CCLRC Rutherford Appleton Laboratory
Chilton Didcot
Oxfordshire OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

CCLRC reports are available online at:
http://www.clrc.ac.uk/Activity/ACTIVITY=Publications;SECTION=225;

ISSN 1358-6254

Neither the Council nor the Laboratory accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

RAL-TR-2006-001

Towards an automatic ordering for a symmetric sparse direct solver[1,2]

Iain S. Duff and Jennifer A. Scott

Abstract

In recent years, nested dissection has grown in popularity as the method of choice for computing a pivot sequence for use with a sparse direct symmetric solver. This is particularly true for very large problems. For smaller problems, minimum degree based algorithms often produce orderings that lead to sparser matrix factors. Furthermore, minimum degree orderings are frequently significantly cheaper to compute than nested dissection. In this report, we look at whether we can predict which ordering will be better, using only the sparsity pattern of the matrix. Our aim is to choose efficiently a good ordering for a wide range of large problems from different application areas.

Keywords: large sparse linear systems, symmetric matrices, ordering, minimum degree, nested dissection.

---

[1] Current reports available from "http://www.numerical.rl.ac.uk/reports/reports.html".

[2] This work was supported by the EPSRC grant GR/S42170.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

December 23, 2005.

1 Introduction

A direct method for solving a sparse linear system $Ax = b$ involves the explicit factorization of the system matrix A (or, more usually, a permutation of A) into the product of lower and upper triangular matrices L and U. In the symmetric case, for positive-definite problems $U = L^T$ (Cholesky factorization) or, more generally, $U = DL^T$, where D is a block diagonal matrix with 1×1 and 2×2 blocks. Many sparse direct solvers have three distinct computational phases: analyse, factorize, and solve. The analyse phase (which is sometimes referred to as the symbolic factorization or ordering step) determines a pivotal sequence and data structures for the factorization. During the factorization phase, this sequence is used to compute the matrix factors. Forward elimination followed by back substitution for each given right-hand side b is performed during the solve phase using the stored factors.

The efficiency of a direct method, in terms of both the storage needed and the work performed, is dependent on the order in which the elimination operations are performed, that is, the order in which the pivots are selected. For symmetric matrices that are positive definite, the pivotal sequence may be chosen using the sparsity pattern of A alone and so the analyse phase involves no computation on real numbers and the factorization phase can use the chosen sequence without modification. For symmetric indefinite problems, many codes again select a tentative pivot sequence based upon the sparsity pattern (generally treating zeros on the diagonal as entries) and then modify the sequence if necessary during the factorization phase to maintain numerical stability.

Finding the optimal ordering is an NP-complete problem (Yannakakis 1981) so, instead of looking for this, the aim is to find an ordering that is "good enough" using heuristics. Since the original work of Markowitz (1957), this has been the subject of much research. For symmetric problems, an important class of ordering methods is based upon the minimum degree algorithm, which was first proposed as algorithm S2 by Tinney and Walker (1967). This uses a local strategy: at each stage of the elimination process, the diagonal entry of the row in the remaining submatrix with the least number of entries is chosen as the next pivot. Variants include the multiple minimum degree algorithm (Liu, 1985) and the approximate minimum degree algorithm (Amestoy, Davis and Duff, 1996). Other local strategies are based on selecting at each stage the pivot that will introduce the least amount of fill-in, or an approximation to this (for example, algorithm S3 of Tinney and Walker, 1967).

An alternative to this kind of local strategy is to choose the ordering using a global strategy. In recent years, methods based on nested dissection, an approach pioneered by George (1973), have become increasingly popular. Nested dissection has its roots in finite-element substructuring. The central concept is the removal of a set of nodes (called a *separator*) from the graph of the matrix that leaves the remaining graph in two (or more) disconnected parts. These parts are themselves further divided by the removal of sets of nodes, with the dissection nested to any depth. The quality of a nested dissection ordering depends crucially upon the quality of its separators; the ordering of the nodes in the different levels of the separator is also important. Nested dissection can be generalised to a multisection approach (Ashcraft and Liu, 1998), in which a separator is constructed that splits the graph into a number of subgraphs.

It is worth stressing that both local and global ordering strategies do a very good job of considerably reducing the amount of fill-in and the computational effort for the succeeding factorization and much of the success of direct methods stems from the reliability and availability

of good ordering algorithms and software. In general, the difference between the performance of a direct solver using a good local ordering and a good global ordering is significantly less than the difference between using one of these good orderings and not reordering at all. But for some problems the "best" ordering allows an in-core factorization where a poorer ordering does not. The qualification on the word "best" is a comment on the metric used to determine which ordering is better. An ordering might be considered better if it takes less time to compute and this might be important in a one-off situation, where only a single factorization and solve follows the analysis. This cost is less important if many factorizations or solves are performed and indeed the normal metric is to declare an ordering better if it results in a factorization with less fill-in or fewer operations. In our subsequent discussions, an ordering will be declared "better" if it yields less fill-in.

In this report, we briefly review the orderings that are currently offered by state-of-the-art symmetric direct solvers. Many of these solvers offer more than one ordering but leave the user to select which is an appropriate ordering for his or her problem. Our aim is to try and automatically choose either a minimum degree or a nested dissection algorithm based on simple characteristics of the matrix. We describe our criteria for selecting an ordering for use in the HSL (HSL, 2004) sparse direct symmetric solver `MA57` (Duff, 2004) and present numerical results for a wide range of large problems from practical applications.

We end this section by describing our test environment and test examples. The experiments were performed on a single Xeon 3.6 GHz processor of a Dell Precision Workstation 670. The NAG Fortran 95 compiler was used with the compiler optimization flag `-O`. All reported timings are CPU times, measured using the Fortran 95 intrinsic `cpu_time` and are given in seconds. The set of test examples is essentially that of Gould and Scott (2004) (see also Gould, Hu and Scott, 2005). However, five problems (`audikw_1`, `SPARSINE`, `AUG2D`, `AUG2DC` and `AUG3D`) have been removed since `MA57` was unable to solve them with the 2.1 GByte memory limit on our test machine. A number of new problems that have been made available to the community as part of the University of Florida Sparse Matrix Collection (`www.cise.ufl.edu/research/sparse/matrices/`) have been added so that the complete set comprises 94 positive-definite problems and 63 numerically indefinite problems, all of order at least $10,000$. Application areas represented by the test set include linear programming, structural engineering, computational fluid dynamics, acoustics, and financial modelling.

We use performance profiles (see Dolan and Moré, 2002) as a means to evaluate and compare the performance of `MA57` with different orderings. The statistics we use are the CPU times required to perform the analyse, factorize, and solve phases, and the number of nonzero entries in the matrix factor. For each ordering and each problem, a CPU time limit of 30 minutes for running the analyse, factorize and solve phases (with a single right-hand side) of `MA57` was imposed; any run of `MA57` with a given ordering that had not completed after this time was recorded as having failed for that problem. All `MA57` control parameters (apart from that which controls the choice of ordering) were used with their default settings.

2 Orderings offered by state-of-the-art solvers

A number of sparse symmetric solvers that were developed in the 1980s and early 1990s used the minimum degree (MD) algorithm. For example, the well known multifrontal code `MA27` of

Duff and Reid (1983) used a minimum degree ordering. The wide acceptance of minimum degree was largely due to its effectiveness in reducing fill-in and the efficiency with which it can be implemented. Minimum degree is still offered by many packages (see Table 2.1) and variants of approximate minimum degree (Amestoy et al., 1996, 2004) perform well on many small and medium sized problems. However, in recent years nested dissection and hybrid algorithms have become increasingly popular. In particular, nested dissection has been found to work well for very large positive-definite problems, including those from 3D discretizations (see, for example, the results presented by Gould and Scott, 2004). As a result, many direct solvers now offer either their own implementation of nested dissection (ND in Table 2.1) or, more commonly, include an explicit interface to the generalized multilevel nested-dissection routine `METIS_NodeND` (or a modification of it) from the MeTiS graph partitioning package of Karypis and Kumar (1998, 1999). Table 2.1 summarizes the ordering options offered by the symmetric solvers that

Code	Ordering options						
	MD	AMD	MMD	ND	MeTiS	MS	MF
`BCSLIB-EXT`	×	×	√	×	√*	×	×
`CHOLMOD`	×	√√*	×	×	√*	×	×
`MA57`	√	√√*	×	×	√*	×	×
`MUMPS`	√	√√*	×	×	√*	√	√
`Oblio`	×	×	√	×	√*	×	×
`PARDISO`	×	×	×	×	√*	×	×
`SPOOLES`	×	×	√	√*	×	√*	×
`SPRSBLKLLT`	×	×	√*	×	×	×	×
`TAUCS`	√	√	√	×	√*	×	×
`WSMP`	×	×	×	√*	×	×	√*

Table 2.1: Orderings offered by state-of-the-art symmetric direct solvers. MD = minimum degree; AMD = approximate minimum degree, MMD = multiple minimum degree; ND = nested dissection; MeTiS = explicit interface to `METIS_NodeND` (or a variant of it); MS = multisection; MF = minimum fill. √√ indicates AMD and one or more variant of AMD are available. * indicates the default.

were compared in the recent study of Gould et al. (2005). An entry marked with * indicates the default (or recommended) ordering. Further details of each of the codes, together with references, are given in the Appendix and in Gould et al. (2005).

We see from Table 2.1 that a number of codes, namely `CHOLMOD`, `MA57`, `MUMPS`, `SPOOLES`, and `WSMP`, have more than one ordering marked as the default. The ordering routine within the

package WSMP computes a minimum fill ordering in addition to an ordering based on recursive graph bisection. It then computes the amount of fill-in that each ordering would generate in a Cholesky factorization and chooses the pivotal sequence based on the better ordering. Similarly, SPOOLES computes orderings using nested dissection and multisection and selects the better. By default, MUMPS automatically chooses the ordering algorithm depending on the packages installed, the size and density of the matrix, and the number of processors being used. On a single processor, a variant of approximate minimum degree is used both for problems of size $n < 10^4$ and for larger but very sparse matrices with quasi-dense rows. Otherwise, METIS_NodeND is used. CHOLMOD also automatically chooses between an AMD and MeTiS ordering and, in some cases, will perform both orderings (further details are given in Section 3.1).

The first release of MA57 (Duff, 2004) used, by default, approximate minimum degree. The code also offered the minimum degree ordering as implemented in the earlier MA27 package as well as a variant of AMD called QAMD that is designed to deal with quasi-dense rows (Amestoy, 1997). Our experience with different orderings (Gould and Scott, 2004) supported earlier observations that, in general, the MeTiS ordering gave a faster factorization with less fill-in than an MD or AMD ordering when the matrix was either very large or not very sparse. This led to Version 2.0.0 of MA57 offering an interface to the MeTiS package so that the user could use the METIS_NODEND ordering. To illustrate the potential benefits of using a nested dissection ordering, we present in Table 2.2 the times for analysing, factorizing and solving a number of the positive-definite examples taken from our test set using MA57. The number of entries in the factor are also given. We see that, in some cases, the factorize and solve times, as well as the number of entries in the factors, are substantially reduced by using MeTiS in preference to QAMD.

Identifier	Analyse		Factorize		Solve		Factor entries ($*10^6$)	
	QAMD	MeTiS	QAMD	MeTiS	QAMD	MeTiS	QAMD	MeTiS
3dtube	0.25	0.90	37.3	9.92	0.23	0.12	34.4	18.5
bmwcra_1	0.78	4.04	90.6	47.3	0.65	0.46	96.9	70.6
bodyy4	0.03	0.15	0.14	0.12	0.001	0.001	0.7	0.6
cfd2	0.62	4.18	74.9	26.2	0.47	0.27	70.2	41.1
OBSTCLAE	0.06	0.32	0.24	0.25	0.01	0.01	1.1	1.2
X104	0.45	1.32	10.3	13.0	0.16	0.18	23.6	27.1

Table 2.2: CPU times (seconds) for the different phases of MA57 and the number of entries in the factor ($*10^6$) when run with AMD and MeTiS orderings.

Of course, if the user has a range of ordering algorithms available, he or she can experiment by running the analyse phase with different options and, on the basis of the statistics returned, select which to use. Investing time in running several orderings may be beneficial if a number of matrices with the same (or similar) sparsity patterns are to be factorized since the cost of the repeated calls to the analyse phase can then be amortized over the calls to the factorization phase. However, if only a single or small number of factorizations are required followed by a small number of solves, more than one call to analyse (which not only computes the ordering but also checks the user's data and manipulates it to the form needed by the factorization) can represent an unacceptable overhead. The overhead of performing multiple orderings was noted by Gould

et al. (2005) for the code WSMP which, as we remarked earlier, performs two orderings by default. For some problems, using a nested dissection ordering can be particularly expensive. This is clear from the results in Table 2.2. We observe that, for bodyy4 and OBSTCLAE, the cost of the analyse phase using MeTiS can exceed that of the subsequent factorization. In our experience, many users (particularly those who regard themselves as non-experts) rely on the default settings and are reluctant to try other values, possibly because they do not feel confident about making appropriate choices. These considerations motivated us to look at the automatic selection of an ordering within MA57.

3 Automatically choosing an ordering

3.1 Positive-definite problems

Our experiments with different orderings initially led us to follow the approach used by MUMPS, that is, to decide a priori on the ordering depending upon the order n of the matrix A and its density. In particular, for positive-definite matrices, if n is at least 50,000, Version 3.0.1 of MA57 always chooses MeTiS. For smaller n, it computes the average number of entries per row ($avnum$) in A and chooses the ordering based on n and $avnum$. In particular, if $n \leq 30000$ and $avnum \leq 100$, or if $30000 < n < 50000$ and $avnum \leq 46$, MA57 chooses QAMD; in all other cases, MeTiS is chosen. The critical values of n and $avnum$ were selected on the basis of numerical experiments.

Choosing between the QAMD and MeTiS orderings using only n and $avnum$ is clearly a very straightforward approach and, if minimising the time taken by the analyse phase is important, it has the advantage that only a single ordering is ever performed. In our tests, it generally performed well (that is, it successfully chose the ordering that led to the faster solution times and the sparser factor) but we observed that for a minority of problems, the "wrong" choice was made. In particular, on some large problems (order greater than 50,000) QAMD can outperform MeTiS. The large test examples for which the MeTiS ordering produces factors that are at least 5 per cent denser than those computed using QAMD are presented in Table 3.1.

Problem	n	Analyse time		AFS time		Factors ($*10^6$)	
		QAMD	MeTiS	QAMD	MeTiS	QAMD	MeTiS
Fcondp2	201822	0.56	1.94	35.4	43.4	48.5	56.2
M_T1	97578	0.43	1.50	18.8	18.7	31.6	34.3
Srb1	54924	0.15	0.42	3.83	4.28	10.2	10.8
X104	108384	0.45	1.32	10.9	14.5	23.6	27.1

Table 3.1: Large problems for which the QAMD factor are at least 5 per cent sparser than the MeTiS factor. (AFS denotes analyse plus factorize plus solve with a single right-hand side).

An alternative is to exploit the fact that the QAMD ordering time is small in comparison with the MeTiS ordering time and the overall solution time. Thus, a possible strategy for all problems is to start by computing an QAMD ordering and, for this ordering, to compute the resulting number of entries in the factors and the number of floating-point operations (flops) that will be required by the numerical factorization. Then, based on these statistics, decide either to accept

the QAMD ordering immediately or to compute a METIS ordering and its fill-in. If both orderings are computed, the better (in terms of fill-in) can be selected. This is the approach adopted by Davis in his package CHOLMOD. The algorithm used within CHOLMOD is outlined in Figure 3.1. Here

```
order = QAMD ordering
if (f1 > 5 * nzl .and. flops > 500 * f1) then
   order2 = METIS ordering
   if (f2 < f1) order = order2
end if
```

Figure 3.1: CHOLMOD strategy for selecting the ordering

n is the order of A, nzl is the number of nonzero entries in the lower triangular part of A, f_1 and f_2 are the numbers of entries in the factors for the QAMD and METIS orderings, respectively, and $flops$ is the number of flops required by the factorization when using the QAMD ordering. In our experiments using this strategy within MA57, we found the METIS ordering was computed for 57 out of the 94 test problems and, in 7 cases, this ordering was subsequently rejected and QAMD selected.

We have modified the analyse phase of MA57 to allow us to compare using different ordering strategies (the rest of the analyse phase plus the factorize and solve phases of MA57 are unchanged). The strategies we consider are:

- The default ordering within Version 3.0.1 of MA57 that automatically selects QAMD or METIS ; we call this the MA57 ordering.
- METIS ordering.
- QAMD ordering.
- Always running both METIS and QAMD and selecting the better based on the number of entries in the factor; we call this METIS +QAMD ordering.
- The CHOLMOD strategy.

We note that a number of implementations of minimum degree-based algorithms that attempt to deal with quasi-dense rows are available (including the C version of the AMD algorithm of Amestoy et al., 2004) but in this report, all reported experiments involving QAMD use the MA57 implementation. In particular, the CHOLMOD strategy uses the MA57 version of QAMD, and not the C implementation that is within the CHOLMOD solver. In Figures 3.2 to 3.4 we present the performance profiles for the analyse, factorize, and solve times for MA57 run with each of these strategies. As expected, computing a minimum degree ordering is significantly faster than computing a nested dissection ordering. However, with the QAMD ordering there were two problems (inline_1 and nd12k) that exceeded the time limit of 30 minutes for the complete solution and so are reported as having failed. The analyse time for the default MA57 ordering is slightly faster than for the CHOLMOD strategy (because MA57 allows a METIS ordering to be computed without always first computing a QAMD ordering) and Figures 3.3 and 3.4 show that both are essentially equally successful. Performing both orderings for each problem leads to the fastest factorize and solve times and the sparsest factors (Figure 3.6). However, the additional analyse cost results in the complete solution (AFS) time being slower than for the MA57 and CHOLMOD strategies (Figure 3.5). It is clear that if a single or small number of factorizations and

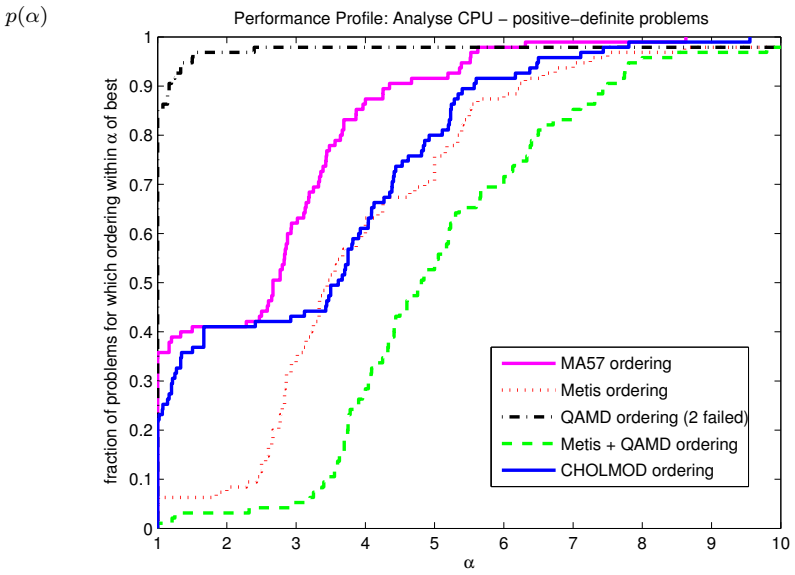

Figure 3.2: Performance profile, $p(\alpha)$: CPU time for the analyse phase (positive-definite problems)

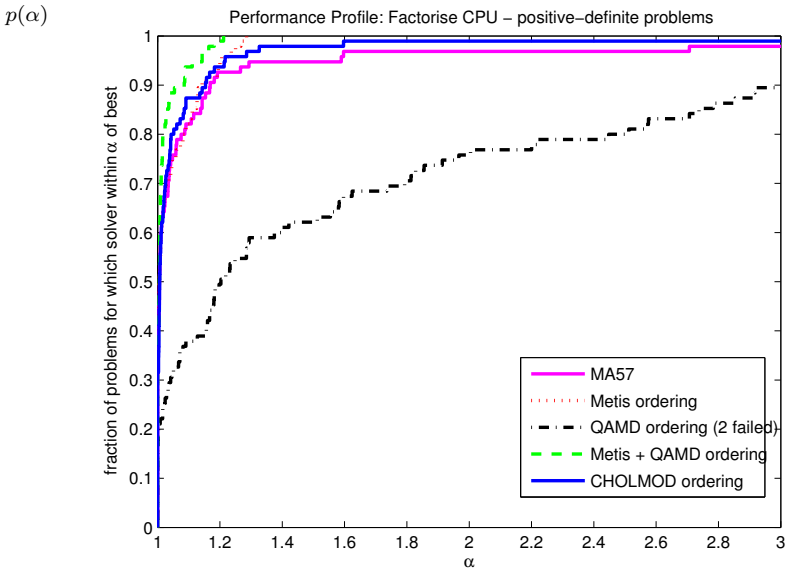

Figure 3.3: Performance profile, $p(\alpha)$: CPU time for the factorize phase (positive-definite problems)

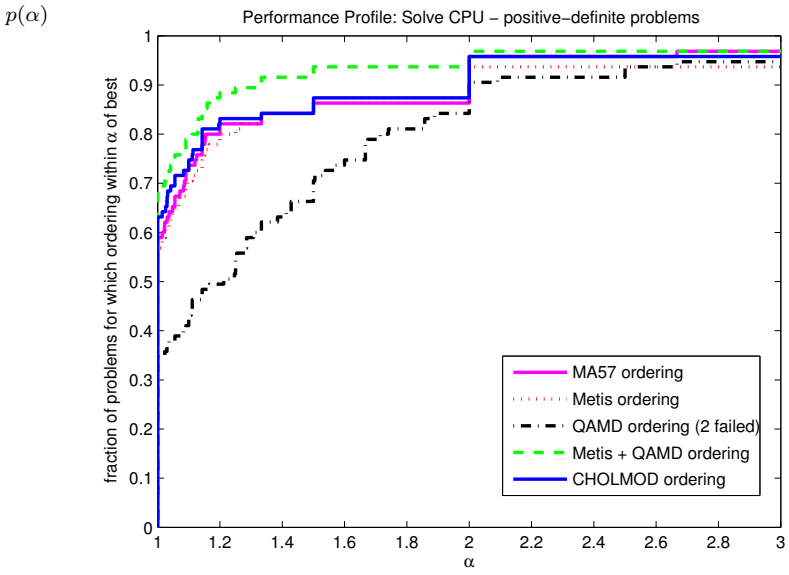

Figure 3.4: Performance profile, $p(\alpha)$: CPU time for the solve phase (positive-definite problems)

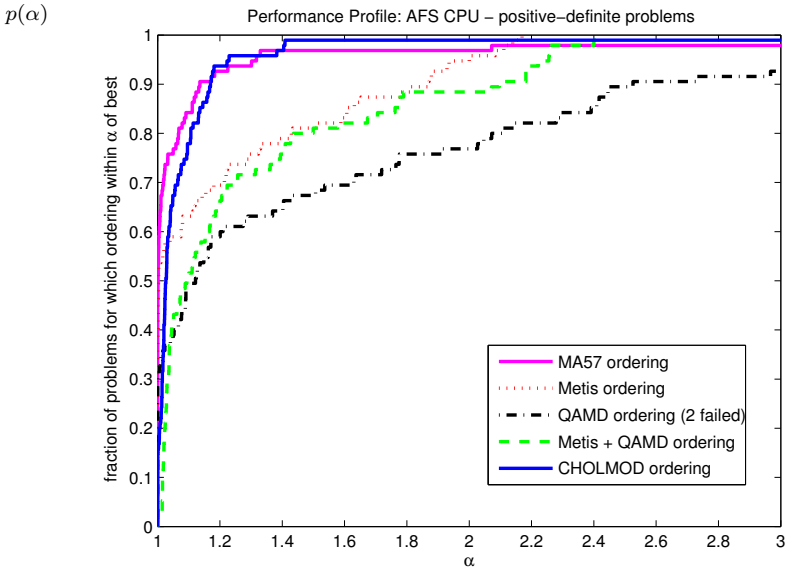

Figure 3.5: Performance profile, $p(\alpha)$: CPU time for the complete solution (positive-definite problems)

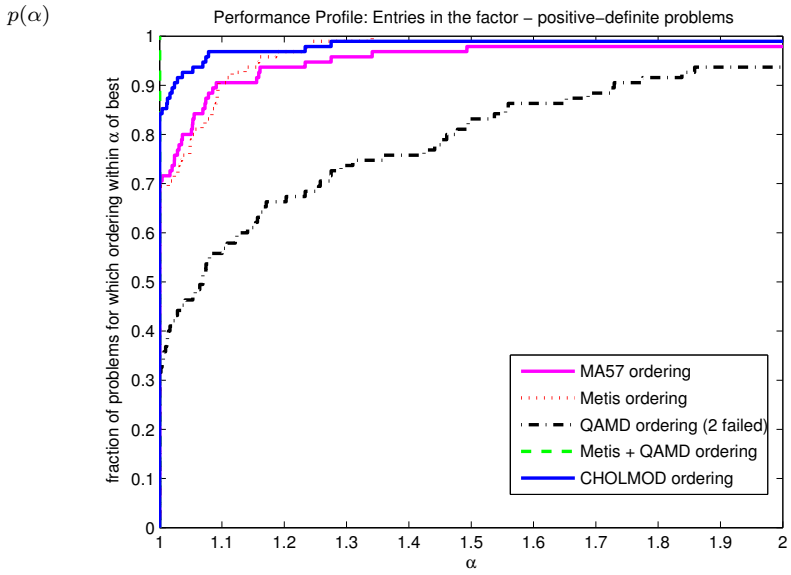

Figure 3.6: Performance profile, $p(\alpha)$: the number of entries in the factor (positive-definite problems)

solves are to follow the analyse phase, it can be advantageous to use one of these automatic ordering selection strategies.

3.2 Indefinite problems

For positive-definite problems, the fill-in and the flop counts predicted by the analyse phase are exact but for indefinite problems, numerical pivoting can result in the predictions being significantly less than the actual statistics returned by the numerical factorization. It is therefore interesting to examine whether it is possible to predict which ordering will perform better, based only on the sparsity pattern and statistics computed using the sparsity pattern alone. For indefinite problems, we quickly found that choosing the ordering using only n and $avnum$ is quite effective at the extremes, but there is a large region in matrix characteristic space where this simplistic approach fails to predict the best ordering. This led us to refine our strategy, again with the aim of attempting to predict the better ordering while minimizing how often we compute more than one ordering. For very large problems we continue to use METIS, unless the matrix is very sparse, in which case, we use QAMD. For smaller problems, we first look at the sparsity pattern of A and define the matrix to be an oxo matrix if it is of the form

$$\begin{pmatrix} 0 & A_1 \\ A_1^T & 0 \end{pmatrix},$$

where the (1,1) and (2,2) zero blocks are of order m_1 and m_2 with $m_1 \geq m_2$. Our selection strategy (see Figure 3.7) then depends on whether or not we have an oxo matrix. For oxo matrices, our experience has been that QAMD is the best choice for `MA57` if the (1,1) zero block is significantly larger than the (2,2) zero block; otherwise, we use METIS. For non oxo matrices, we first run the cheap QAMD ordering and compute its predicted fill-in. If this lies below a given

threshold, we use QAMD, even though it is be possible that MeTiS may do even better (we want to avoid the potentially costly overhead of running MeTiS). If QAMD does not do well, we run MeTiS and compare its predicted fill-in with that for the previous QAMD run. If MeTiS predicts a lower fill-in, we use it. Otherwise, we employ QAMD (although to do so within the current Fortran 77 code `MA57` we have first to rerun this ordering because to do otherwise would require additional storage).

We present our algorithm in Figure 3.7. Here n is the order of A, nzl is the number of nonzero entries in the lower triangular part of A, and, for an oxo matrix, m_1 and m_2 are the orders of the (1,1) and (2,2) zero blocks. In addition, p_1 and p_2 are the predicted fill-in for the QAMD and MeTiS orderings respectively; N and c_j, $j = 1, 2, 3$ are parameters. For `MA57` we have selected these parameters using the results of numerical experiments to have the values $N = 10^5$, $c_1 = 3$, $c_2 = 1.8$, $c_3 = 10$. These were chosen on the basis of a single factorization and solving for a single right-hand side. As already noted, if several factorizations or many repeated solves are required, it may be worthwhile to always run both the QAMD and MeTiS orderings and choose the one that predicts the lower fill-in.

```
if (n > N) then
   if (nzl < c1 * n) then
      order = QAMD ordering
   else
      order = MeTiS ordering
   end if
else if (oxo) then
   if (m1 > c2 * m2) then
      order = QAMD ordering
   else
      order = MeTiS ordering
   end if
else
   order = QAMD ordering
   if (p1 > c3 * nzl) then
      order2 = MeTiS ordering
      if (p2 < p1) order = order2
   end if
end if
```

Figure 3.7: Algorithm within `MA57` for automatically selecting the ordering in the indefinite case

In our tests using this algorithm, MeTiS was called for 36 of the 63 test problems. In 24 instances, the call to MeTiS was preceded by a call to QAMD, and in 3 of these cases, the MeTiS ordering was rejected.

We also ran tests using the simpler `CHOLMOD` strategy within `MA57`. Although this strategy was designed with positive-definite problems in mind, it performed well on the indefinite problems. This is clear from the performance profiles in Figures 3.8 to 3.12. The `CHOLMOD` strategy called MeTiS for 34 problems; it was only rejected in favour of the previously computed QAMD ordering in one instance. When used with a QAMD ordering, `MA57` was not able to solve three problems

(Andrews, DTOC and NCVXQP7) within the CPU time limit of 30 minutes. The CHOLMOD strategy selected QAMD for problem DTOC and this accounts for the one reported failure for the CHOLMOD ordering.

In Figures 3.8 to 3.10 we present the performance profiles for the analyse, factorize, and solve times for MA57 run with the different ordering strategies. Again, computing a minimum degree ordering is significantly faster than computing a nested dissection ordering. Over the indefinite

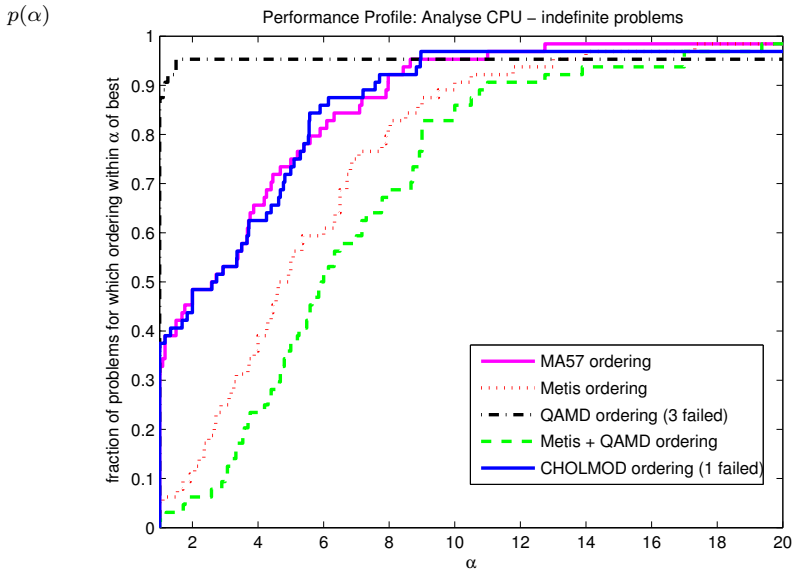

Figure 3.8: Performance profile, $p(\alpha)$: CPU time for the analyse phase (indefinite problems)

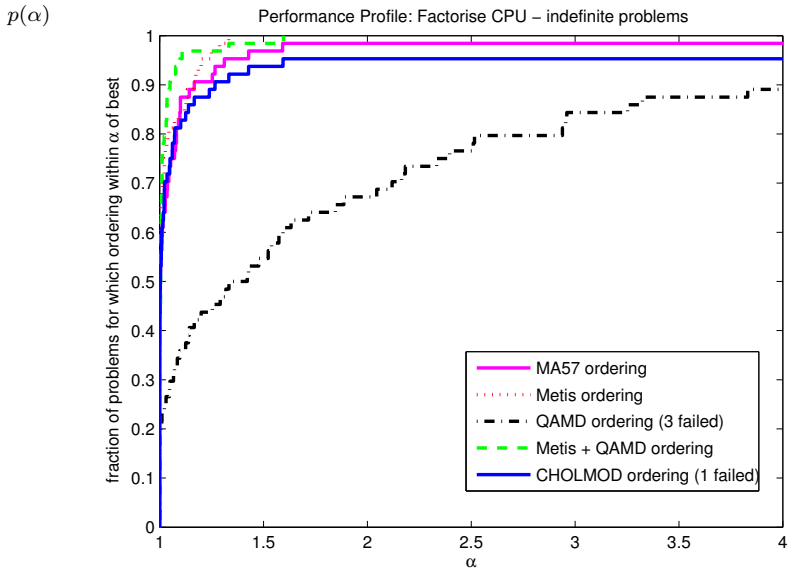

Figure 3.9: Performance profile, $p(\alpha)$: CPU time for the factorize phase (indefinite problems)

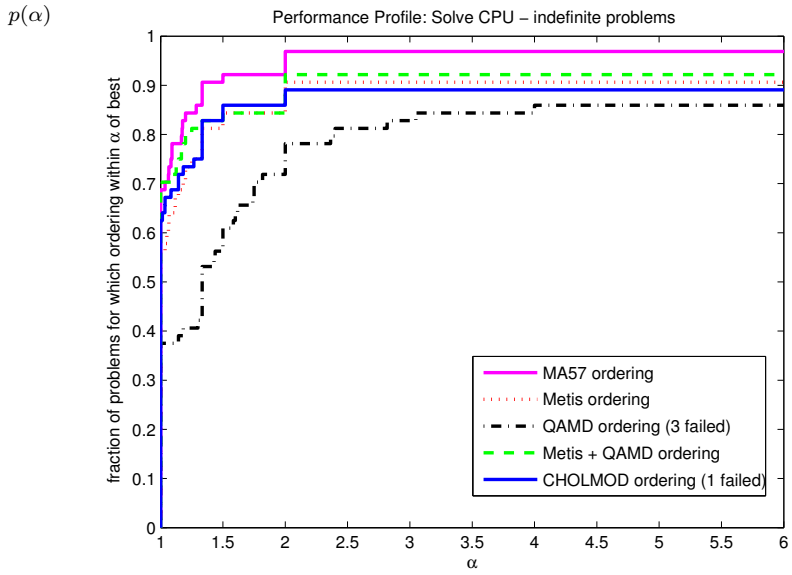

Figure 3.10: Performance profile, $p(\alpha)$: CPU time for the solve phase (indefinite problems)

test set, the analyse time for the default `MA57` ordering is comparable with that for the `CHOLMOD` strategy and Figures 3.9 and 3.10 suggest there is little to choose between the two, with the `MA57` ordering perhaps having a small advantage. Performing both QAMD and MeTiS orderings for each problem leads to the fastest factorize and solve times and the sparsest factors (Figure 3.12). This is interesting since the ordering is picked on the basis of the predicted fill-in based on sparsity alone. `CONT-201` was the only problem in our test set for which the analyse phase selected an ordering that led to the number of entries in the factor being more than 5 per cent greater than for the rejected ordering (and for this problem, the tentative pivot sequences chosen by both QAMD and MeTiS were modified substantially during the factorization). Looking at the profile for the complete solution (AFS) time (Figure 3.11), we again see the potential advantage of using an automatic ordering selection strategy.

4 Concluding remarks

We have described in this paper the logic and rationale behind the automatic choice of ordering strategy used by `MA57` and we have compared this with other automatic and non-automatic strategies. We conclude that it can be worthwhile to have an automatic selection strategy particularly if only few factorizations or solves follow the analysis. The automatic strategy is also a good option for users who wish to treat the software as a black-box and who have no interest in understanding the use of orderings or in tuning the code for their problem. Sadly there are all too many users in this category.

The strategy used by `MA57` seems as good as alternatives although the attempts to use the structure of some indefinite problems do not appear to yield any additional benefits. Indeed it is not clear that differentiating between positive-definite and indefinite systems is of any benefit.

We note that many automatic selection strategies assume that the cost of generating the nested dissection ordering is substantially more expensive than that of a local ordering such

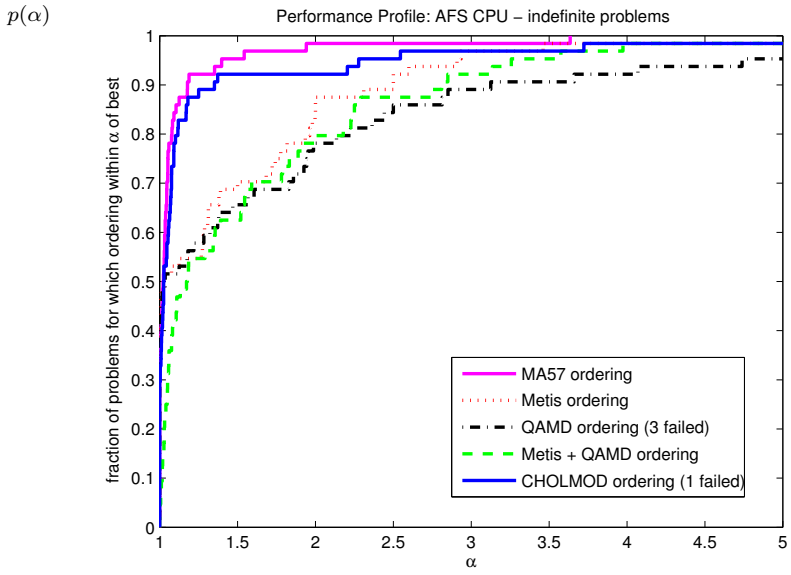

Figure 3.11: Performance profile, $p(\alpha)$: CPU time for the complete solution (indefinite problems)

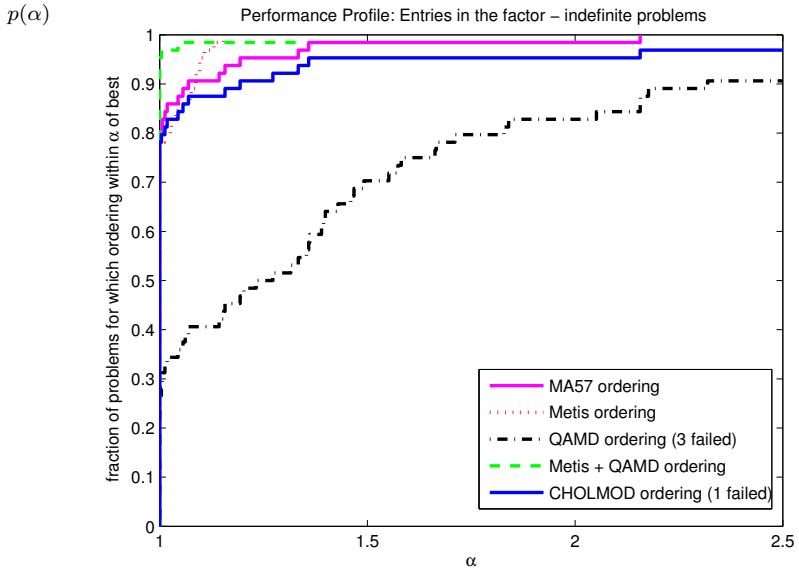

Figure 3.12: Performance profile, $p(\alpha)$: the number of entries in the factor (indefinite problems)

as QAMD. This is certainly the case when the METIS package is used to compute the nested dissection ordering which is used by many sparse direct solvers. We plan to try to develop faster nested dissection orderings and, if successful, may have to alter the parameters or strategy for automatic choice of orderings within `MA57`.

5 Acknowledgements

We are grateful to Tim Davis of the University of Florida for advice on the algorithm used by `CHOLMOD`. We would also like to thank Tim and our colleague John Reid for commenting on a draft of this report.

References

P.R. Amestoy. Recent progress in parallel multifrontal solvers for unsymmetric sparse matrices. *in* 'Proceedings of the 15th World Congress on Scientific Computation, Modelling and Applied Mathematics, IMACS 97, Berlin', 1997.

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

P.R. Amestoy, T.A. Davis, and I.S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Mathematical Software*, **30**(3), 381–388, 2004.

C. Ashcraft and J.W.H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Analysis and Applications*, **19**, 816–832, 1998.

E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–213, 2002.

I.S. Duff. MA57– a new code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Mathematical Software*, **30**, 118–154, 2004.

I.S. Duff and J.K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, **9**, 302–325, 1983.

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

N.I.M. Gould and J.A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. *ACM Trans. Mathematical Software*, pp. 300–325, 2004.

N.I.M. Gould, Y. Hu, and J.A. Scott. A numerical evaluation of sparse direct solvers for the solution of large, sparse, symmetric linear systems of equations. Technical Report 2005-005, Rutherford Appleton Laboratory, 2005. To appear in *ACM Trans. Mathematical Software*.

HSL. A collection of Fortran codes for large-scale scientific computation, 2004. See http://www.cse.clrc.ac.uk/nag/hsl/.

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

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, **20**, 359–392, 1999.

J.W.H. Liu. Modification of the Minimum-Degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, **11**(2), 141–153, 1985.

H.M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, **3**, 255–269, 1957.

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

M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic and Discrete Methods*, **2**, 77–79, 1981.

Appendix

We show, in Table A.1, a list of software for the direct solution of sparse symmetric matrices and either a web or email contact for each.

Code	Webpage / email contact
BCSLIB-EXT	www.boeing.com/phantom/BCSLIB-EXT/index.html
CHOLMOD	www.cise.ufl.edu/∼davis/
MA57	www.cse.clrc.ac.uk/nag/hsl
MUMPS	www.enseeiht.fr/lima/apo/MUMPS/
Oblio	dobrian@cs.odu.edu or pothen@cs.odu.edu
PARDISO	www.computational.unibas.ch/cs/scicomp/software/pardiso
SPOOLES	www.netlib.org/linalg/spooles/spooles.2.2.html
SPRSBLKLLT	EGNg@lbl.gov
TAUCS	www.cs.tau.ac.il/∼stoledo/taucs/
WSMP	www-users.cs.umn.edu/∼agupta/wsmp.html

Table A.1: Contact details for the solvers discussed in Section 2