Strategies for scaling and pivoting for sparse symmetric indefinite problems

Iain. S. Duff* and Stéphane Pralet†

Technical Report TR/PA/04/59

July 13, 2004

Abstract

We consider ways of implementing preordering and scaling for symmetric systems and show the effect of using this technique with a multifrontal code for sparse symmetric indefinite systems. After having presented a new method for scaling, we propose a way of using an approximation to a symmetric weighted matching to predefine 1×1 and 2×2 pivots prior to the ordering and analysis phase. We also present new classes of orderings called "relaxed) constrained orderings" that mix structural and numerical criteria.

*i.s.duff@rl.ac.uk, CERFACS, Toulouse, and Atlas Centre, RAL, Oxon OX11 0QX, England.
†Stephane.Pralet@cerfacs.fr. CERFACS, 42, av. G. Coriolis, 31057 Toulouse Cedex 01, France.

Contents

1 Introduction **3**

2 Symmetric indefinite multifrontal solvers and numerical pivoting **4**
2.1 Multifrontal approach 4
2.2 Numerical pivoting 4

3 Experimental environment **5**
3.1 Test machine 5
3.2 Matrices 5
3.3 Measures and methodology 6

4 Scaling **8**
4.1 Existing symmetric scaling 8
4.2 Adaptation of MC64 scaling 8
4.2.1 Maximum weighted matching 8
4.2.2 Symmetrization 9
4.2.3 Structurally singular matrices? 10
4.3 Influence of scaling 11

5 Ordering and weighted matching **15**
5.1 Symmetric maximum weighted matching 15
5.2 Selection of 2 by 2 pivots and symmetric weighted matchings 18
5.3 Coupling detected pivots and orderings 20
5.3.1 Ordering on the compressed graph 20
5.3.2 Constrained ordering 21
5.3.3 Relaxation of constrained ordering 23

6 Experimental results **24**
6.1 General symmetric indefinite matrices (set 2) 25
6.2 Augmented systems (set 1) 27
6.2.1 AMD based approaches 27
6.2.2 MeTiS based approaches 29
6.2.3 AMF based approaches 31
6.2.4 Best approaches discussion. 32

7 Conclusions and future work **33**

1 Introduction

We study techniques for scaling and choosing pivots when using multifrontal methods in the LDL^T factorization of symmetric indefinite matrices where L is a lower triangular matrix and D is a block diagonal matrix with 1×1 and 2×2 blocks.

Our main contribution is to define a new method for scaling and a way of using an approximation to a symmetric weighted matching to predefine 1×1 and 2×2 pivots prior to the ordering and analysis phase. We also present new classes of orderings called "(relaxed) constrained orderings" that select pivots during the symbolic Gaussian elimination using two graphs: the first one contains information about the structure of the reduced matrix and the second one gives information about the numerical values.

Prior to the LU factorization of a matrix A, the package `MC64` [11, 12] can be used to get a maximum weighted matching so that the corresponding permutation will place large entries on the diagonal. The matrix can then be scaled so that diagonal entries have modulus one and off-diagonals have modulus less than or equal to one. This has been found to greatly improve the numerical stability of the subsequent LU factorization. If, however, `MC64` is applied to a symmetric matrix the resulting permutation will not normally preserve symmetry. In this paper, we examine ways in which symmetric preprocessing can be applied and, in particular, how `MC64` can be used while still preserving symmetry. Our preprocessing will be followed by a standard symmetric permutation in order to decrease the fill-in in the factors.

We will use our symmetric preprocessing with a symmetric multifrontal code `MA57` [10] to validate our heuristics on real test problems that we have divided into two sets: the first consists of augmented matrices and the second contains general indefinite matrices.

In Section 2, we present some general characteristics of multifrontal symmetric indefinite solvers that will be useful in understanding our preprocessing strategies and experimental results. In Section 3, we describe our experimental environment. Section 4 shows how `MC64` scaling can be used for the symmetric indefinite case and studies the effects on the factorization of different kinds of scaling. Section 5 discusses orderings to decrease the fill-in and to give good preselected pivots. Our experimental results are given in Section 6 and the best strategies are discussed. Finally, in Section 7, we summarize our results and consider future improvements.

We make extensive use of routines from HSL [21]. Any code with a name beginning with `MA` or `MC` is from HSL or is a derivative of an HSL code, for example `MC64` or `MA57`.

2 Symmetric indefinite multifrontal solvers and numerical pivoting

2.1 Multifrontal approach

Multifrontal methods [13, 14] use an elimination tree [24] to represent the dependencies of the computation. Each node of this tree is associated with a frontal matrix F that is assembled (summed) by contributions from its children and the original matrix. It is of the form:

$$\begin{array}{rc} & \begin{array}{cc} \text{fully summed columns} & \text{partially summed columns} \\ \downarrow & \downarrow \end{array} \\ \begin{array}{r} \text{fully summed rows} \rightarrow \\ \text{partially summed rows} \rightarrow \end{array} & \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}. \end{array}$$

Then elimination operations are performed using pivots from within the fully summed block, F_{11}, and the Schur complement or contribution block $F_{22} \leftarrow F_{22} - F_{21}F_{11}^{-1}F_{12}$ is computed. In the symmetric case, $F_{12} = F_{21}^T$, the matrices F_{11} and F_{22} are symmetric, pivots are chosen from the diagonal as discussed in the following section, and operations and storage are about half that of the general case.

2.2 Numerical pivoting

In the unsymmetric case, at step k of the Gaussian elimination, the pivot (p, q) is selected from the fully summed rows and columns and the entries a_{ij} of the remaining submatrix are updated:

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - \frac{a_{ip}^{(k)} a_{qj}^{(k)}}{a_{pq}^{(k)}}.$$

To limit the growth of the entries in the factors and thus to have a more accurate factorization, a test on the magnitude of the pivot is commonly used. a_{pq} can be selected if and only if

$$|a_{pq}| \geq u \max_j |a_{pj}| \tag{2.1}$$

where u is a threshold between 0 and 1. Thanks to this criterion of selection the growth factor is limited to $1/u$.

In the symmetric indefinite case, we have to perform 1×1 and 2×2 pivoting if we want to keep the symmetry while maintaining stability. Pivot selection can be done using the Bunch-Parlett [4] or Bunch-Kaufman [3] algorithm. In the context of sparse matrices, the criterion of the Duff-Reid algorithm ([13], as modified in [15]) can be used to ensure a growth factor lower than $1/u$ at each step of Gaussian elimination. A 1×1 diagonal pivot can be selected if and only if it satisfies the

inequality (2.1). A 2×2 pivot $P = \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}$ can be selected if and only if it satisfies:

$$|P^{-1}| \begin{pmatrix} \max_k |a_{pk}| \\ \max_k |a_{qk}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix} \tag{2.2}$$

where $|P^{-1}|$ denotes the matrix whose values are the absolute values of P^{-1}. During the elimination, it may not be possible to eliminate some fully summed variables. Elimination of these variables must then be delayed to the parent. This has the effect of causing extra fill-in and thus increases the memory and the number of operations. Too many delayed pivots can severely slow down the factorization.

3 Experimental environment

3.1 Test machine

Our experiments are conducted on one node of a COMPAQ Alpha Server SC45 at CERFACS. There is 4 GBytes of memory shared between 4 EV68 processors per node and we disable three of the processors so that we can use all the memory of the node with the remaining single processor. We use the Fortran 90 compiler, f90 version 5.5 with -O option. Our use of integer pointers restricts our array size to 2 GBytes. If the factorization needs to allocate an array larger than 2 GBytes or requires more than 30 minutes, we consider that the factorization is not successful.

3.2 Matrices

We conduct our experiments on a number of test problems that we divide into two sets. We decided to test our preprocessings on matrices of order between 10000 and 100000, and to restrict the number of matrices of the same type to 2 or 3 in each set in order to avoid class effects. The matrices have an identification number that will be used to represent them on the x-axis of some figures. Except for bloweybl, bloweybq and qpband that come from CUTEr, they are available from `ftp.numerical.rl.ac.uk/pub/matrices/symmetric/indef/` and correspond to a subset of the matrices collected by [20] for testing symmetric sparse solvers. The matrices come from the University of Florida collection (UF) [6], the Maros and Meszanos quadratic programming collection (M2) [25], the CUTEr optimization test set (CUTEr) [19] and from Kumfert and Pothen (KP) [23]. Some problems were generated by Andy Wathen (AW), Mario Arioli (MA), and Miroslav Tuma (MT). The c-* matrices were obtained from Olaf Schenk (OS) and are also available in [6]. These problems are described in Tables 3.1 and 3.2.

Many of the test matrices correspond to augmented matrices of the form

$$\mathcal{K}_{H,A} = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix}$$

Matrix	Id	n	nnz	Origin
A0NSDSIL	1	80016	200021	Linear Complementarity problem (CUTEr)
A2NNSNSL	2	80016	196115	Linear Complementarity problem (CUTEr)
A5ESINDL	3	60008	145004	Linear Complementarity problem (CUTEr)
AUG3DCQP	4	35543	77829	Expanded system–3D PDE (CUTEr)
BLOCKQP1	5	60012	340022	QP with block structure (CUTEr)
BLOWEYA	6	30004	90006	Cahn-Hilliard problem (CUTEr)
BRAINPC2	7	27607	96601	Biological model (CUTEr)
BRATU3D	8	27792	88627	3D Bratu problem (CUTEr)
CONT-201	9	80595	239596	KKT matrix–Convex QP (M2)
K1_SAN	10	67759 (1)	303364	Straz pod Ralskem mine model (MT)
NCVXQP1	11	12111	40537	KKT matrix–nonconvex QP (CUTEr)
NCVXQP5	12	62500	237483	KKT matrix–nonconvex QP (CUTEr)
NCVXQP7	13	87500	312481	KKT matrix–nonconvex QP (CUTEr)
SIT100	14	10262	34094	Straz pod Ralskem mine model (MT)
bloweybl	15	30003 (1)	60000	Cahn-Hilliard problem (CUTEr)
cvxqp3	16	17500	62481	Convex QP (CUTEr)
mario001	17	38434	114021	Stokes equation (MA)
olesnik0	18	88263	402623	Straz pod Ralskem mine model (MT)
qpband	19	20000	30000	QP (CUTEr)
stokes128	20	49666	295938	Stokes equation (MA)
stokes64	21	12546	74242	Stokes equation (AW)
tuma1	22	22967	50560	Mine model (MT)
tuma2	23	12992	28440	Mine model (MT)

Table 3.1: Augmented matrices of $H \neq 0$ type (set 1). (x): x is the structural deficiency of the matrix. Id: identification number.

We performed some earlier experiments [8] with matrices of this form where H is the zero matrix. We note that matrices of this form are structurally singular unless A is square nonsingular. The set of matrices described in Table 3.1 are of the above form but with $H \neq 0$. The second set, in Table 3.2, corresponds to general (nonzero diagonal) indefinite symmetric matrices. Note that we also include the BOYD1 matrix which is an augmented system in this set because its number of constraints (order of the zero block) is negligible.

3.3 Measures and methodology

In our earlier experiments [8], we clearly identified that `MA57` was not tailored to augmented matrices with a zero $(1,1)$ block. These were much better handled by `MA47` [16] that respects the two zero blocks that exist when 2×2 pivots with two zero entries on the diagonal (referred to as *oxo* pivots) are eliminated. This is why we have decided not to try to improve `MA57` on this set of matrices and why we will not use matrices of this type in our experiments. [20] also reported some failures of `MA57` on this class of problems. They found that `MA47` and `MA67` usually performed better on these problems. We ran the `MA57` factorization with a pivoting threshold

Matrix	Id	n	nnz	Origin
BOYD1*	24	93279	652246	KKT matrix–Convex QP (CUTEr)
DIXMAANL	25	60000	179999	Dixon-Maany optimization example (CUTEr)
HELM3D01	26	32226	230335	Helmholtz problem (MA)
LINVERSE	27	11999	53988	Matrix inverse approximation (CUTEr)
SPMSRTLS	28	29995	129971	Sparse matrix square root (CUTEr)
bcsstk35	29	30237	740200	Stiffness matrix–automobile seat frame (UF)
bcsstk39	30	46772	1053717	Stiffness matrix–track ball (UF)
bloweybq	31	10001	30000	Cahn-Hilliard problem (CUTEr)
c-68	32	64810	315403	Optimization model (OS)
c-71	33	76638	468079	Optimization model (OS)
copter2	34	55476	407714	Helicopter rota blade (KP)
crystk02	35	13965	491274	Stiffness matrix–crystal free vibration (UF)
crystk03	36	24696	887937	Stiffness matrix–crystal free vibration (UF)
dawson5	37	51537	531157	Aeroplane actuator system (UF)
qa8fk	38	66127	863353	FE matrix from 3D acoustics (UF)
vibrobox	39	12328	157014	Vibroacoustic problem (UF)

Table 3.2: General symmetric indefinite matrices (set 2). $*$: BOYD1 is in set 2 because the number of constraints is negligible. Id: identification number.

equal to 10^{-2} and obliged its analysis to merge two nodes if both require less that 16 eliminations (`ICNTL(12) = 16`). We compare our codes using the performance profiling system of [7]. In these profiles, if α is the value on the x-axis, then the y-axis measures the fraction of times that each code is within a factor α of the best code. Note that we do not explicitly label the axes of these profile graphs.

To compare our different approaches we will look at the factorization time, the memory needed to complete the factorization, and the reliability of the analysis in terms of memory forecasting. We would like to have an estimation of the required memory after the analysis that is not too far from that actually needed by the factorization. The quality of the analysis prediction will be assessed using the ratio between the memory actually used during the factorization and that predicted by the analysis.

It is standard practice to increase the storage estimated by the analysis and allocate rather more storage in the hope that the factorization will be successful even if some additional numerical pivoting occurs. The percentage by which we increase the estimate from the analysis will be called the memory relaxation. In our experiments, we use a memory relaxation of 20% and 50%. We evaluate the analysis prediction as follows: if the memory required by the factorization is larger than the relaxed analysis estimation, we classify the run as a failure.

Firstly, in Section 4, we study the influence of scaling on the `MA57` factorization phase. Secondly, the effect of additional numerical and structural preprocessing are analysed when they are combined with different orderings (`AMD` [1] in 6.2.1,

MeTiS [22] in 6.2.2 and AMF [26, 28] in 6.2.3). The best approaches will be discussed in Sections 6.1 and 6.2.4.

4 Scaling

In this section, we examine the effect of scaling and identify the most robust approach. Firstly, we describe existing scaling approaches; secondly, we propose an alternative based on the symmetrization of an unsymmetric scaling; and finally we analyse our experimental results.

4.1 Existing symmetric scaling

Let A be a square symmetric sparse matrix. MC30 [5] scales A so that the scaled matrix DAD has its nonzero entries near to 1 in absolute value. In practice, it minimizes $\sum_{a_{ij}\neq 0}(\log|a_{ij}|)^2$.

If A is structurally nonsingular, MC77 [29] used with the p-norm, $||\ ||_p$, computes a sequence of matrices $D_r^{(k)}AD_c^{(k)}$ that converge to a matrix that has its column and row norms equal to 1 (which corresponds to a doubly stochastic matrix if $p = 1$). If $p \neq \infty$, this limit is unique. If A is symmetric then this scaling is symmetric ($D_r^{(k)} = D_c^{(k)}$). We will study the effect of MC77 with $p = 1$ and $p = \infty$. These scalings will be identified by MC77one and MC77inf, respectively. Note that, when the matrix is structurally singular, the convergence of the algorithm is not guaranteed. Moreover, the convergence to a doubly stochastic matrix with $p = 1$ is slower than with $p = \infty$. In our experiments, we limit the number of MC77 steps to 20.

4.2 Adaptation of MC64 scaling

4.2.1 Maximum weighted matching

It is common to represent a symmetric matrix A of order n by a weighted undirected graph given by $G = (V, E)$, where $V = \{1, 2, ...n\}$ and each undirected edge (i, j) of E corresponds to the off-diagonal nonzeros a_{ij} and a_{ji} of A and has a weight of $2|a_{ij}|$ if $i \neq j$. If $i = j$, depending on the context we may or may not want to add a self-loop to G of weight $|a_{ii}|$. The matrix can also be represented by a bipartite graph $G = (R, C, E)$ where $(i, j) \in R \times C$ belongs to E if and only if $a_{ij} \neq 0$ and has a weight of $|a_{ij}|$. In the following we will associate a weight ω with a set of edges $\mathcal{S}$, defined by:

$$\omega(\mathcal{S}) = \prod_{(i,j)\in\mathcal{S}} |e_{ij}|, \text{ where } |e_{ij}| \text{ is the weight of the edge } (i, j). \tag{4.1}$$

We define a **matching** on the bipartite graph as a set of edges, no two of which have the same vertex as an endpoint. A maximum matching is a matching with

maximum cardinality that will be n if the matrix is structurally nonsingular. We will later (in Section 5.1) relate matchings on the bipartite graph to matchings on another undirected graph that is a modified version of the undirected graph described above.

4.2.2 Symmetrization

When `MC64` is used with a product metric as in 4.1 to define the weight of a set of edges, it also returns a row and column scaling. We now show how a symmetric scaling can be built from the `MC64` scaling. It will be called `MC64SYM`.

Definition 4.1 *A matrix $B = (b_{ij})$ is said to satisfy the* `MC64` *constraints if and only if*

$$\exists \text{ a permutation } \sigma, \text{ such that } \forall i, |b_{i\sigma(i)}| = ||b_{.\sigma(i)}||_\infty = ||b_{i.}||_\infty = 1.$$

Property 4.1 *Let $\mathcal{M}$ be the maximum matching of the symmetric matrix A returned by* `MC64` *and $D_r = (d_{r_i})$, $D_c = (d_{c_i})$ be the row and column scaling respectively. Let $D = (d_i) = \sqrt{D_r D_c}$. Then DAD is a symmetrically scaled matrix that satisfies the* `MC64` *constraints.*

Proof: The entry of $|DAD|$ in position (i, j) is

$$|d_i d_j a_{ij}| = \sqrt{|d_{r_i} d_{c_j} a_{ij}|}\sqrt{|d_{r_j} d_{c_i} a_{ji}|} \leq 1$$

because it is the square root of the product of two entries of $|D_r A D_c|$. Let σ be the column permutation associated with $\mathcal{M}$. Thus, for all i, $|d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}| = 1$. Let $\lambda_i = |d_{c_i} d_{r_{\sigma(i)}} a_{i\sigma(i)}| = |d_{c_i} d_{r_{\sigma(i)}} a_{\sigma(i)i}|$. $\forall i$, $\lambda_i \leq 1$ because it corresponds to an entry of $|D_r A D_c|$. Then

$$\prod_{1\leq i\leq n} \lambda_i = \left[\prod_{1\leq i\leq n} d_{r_i}\right]\left[\prod_{1\leq i\leq n} d_{c_i}\right]\left[\prod_{1\leq i\leq n} a_{i\sigma(i)}\right] = \prod_{1\leq i\leq n} (d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}) = 1,$$

where we exchange the terms in the products to get the product of the scaled terms in the `MC64` maximum matching. That is, the product of the numbers, λ_i, each of which is less than or equal to one, is one, so that $\lambda_i = 1$, $\forall i$ and so

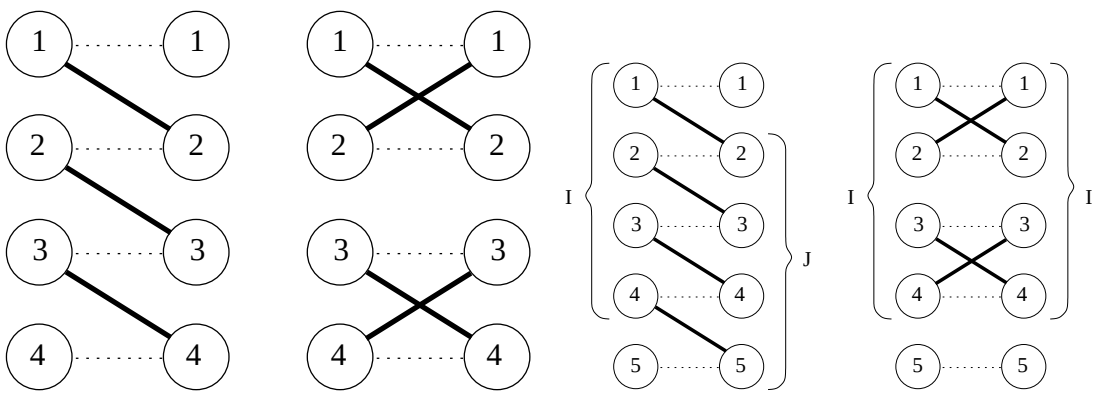
$$\forall i,\ |d_i d_{\sigma(i)} a_{i\sigma(i)}|^2 = |d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}||d_{r_{\sigma(i)}} d_{c_i} a_{i\sigma(i)}| = 1 \times \lambda_i = 1.$$

□

4.2.3 Structurally singular matrices?

Structurally singular matrices are quite common in optimization. The notion of a maximum weighted matching with the product metric is not well defined for `MC64`. All the maximum matchings of `MC64` will have zero weight. Moreover, it is impossible to find scaling factors that satisfy the `MC64` constraints for structurally singular matrices (it is impossible to find a permutation σ that satisfies the relation of Definition 4.1).

Suppose that the weight of a matching on a structurally singular matrix is given by the product of the absolute values of the matched entries (which are nonzeros). Then we can define a maximum weighted matching on a structurally singular matrix as a matching of maximum weight among the matchings of maximum size. We did not modify the `MC64` code to get this kind of matching, but we used a less complicated algorithm to get an approximation to a maximum weighted matching. We first apply `MC64` to A and we get a maximum matching $\mathcal{M}$. Then we extract a structurally nonsingular symmetric submatrix $\tilde{A}$ from A using Property 4.2 below. We then apply `MC64` this time to $\tilde{A}$.



(a) Example of even maximum open path. (b) Example of odd maximum open path.

Figure 4.1: Maximum open paths. Bold edges are edges of the matching. Dashed lines represent the diagonal of the matrix that may not be entries.

Property 4.2 *Let A be a symmetric matrix and let $\mathcal{M} = \mathcal{I} \otimes \mathcal{J}$ be a maximum matching of A. The restriction of A to $\mathcal{I} \times \mathcal{I}$ is structurally nonsingular.*

Proof: We define the bipartite graph $G = (R, C, E)$ where $a_{ij} \neq 0$ if and only if $(i^{\mathcal{R}}, j^{\mathcal{C}}) \in E$. The notations $^{\mathcal{R}}$ (resp. $^{\mathcal{C}}$) will be added to a set of indices when we want to explicitly mention that it refers to a set of rows (resp. columns) included in R (resp. in C). A path in this graph from i_1 to i_k is defined by a sequence $(i_1, \ldots, i_k)$ where $(i_t^{\mathcal{R}}, i_{t+1}^{\mathcal{C}})_{t=1,\ldots,k-1} \in E$. This path is a cycle if $i_k \equiv i_1$. By definition, the length of this path is k. We can similarly define a path using the maximum matching

$\mathcal{M} \subset E$ where $(i_t^{\mathcal{R}}, i_{t+1}^{\mathcal{C}})_{t=1,\ldots,k-1} \in \mathcal{M}$. A maximum open path of the matching is defined as a path $(i_1, \ldots, i_k)$ where there are no v such that $(v^{\mathcal{R}}, i_1^{\mathcal{C}}) \in \mathcal{M}$ or $(i_k^{\mathcal{R}}, v^{\mathcal{C}}) \in \mathcal{M}$. All matching edges are either in cycles or in maximum open paths.

The length of a maximum open path must be odd since, if it were even the symmetry of the matrix would allow us to extend the matching by using the edges $(i_1^{\mathcal{R}}, i_2^{\mathcal{C}}), (i_2^{\mathcal{R}}, i_1^{\mathcal{C}}), (i_3^{\mathcal{R}}, i_4^{\mathcal{C}}), (i_4^{\mathcal{R}}, i_3^{\mathcal{C}}) \ldots, (i_{k-1}^{\mathcal{R}}, i_k^{\mathcal{C}}), (i_k^{\mathcal{R}}, i_{k-1}^{\mathcal{C}})$ thus contradicting that we have a maximum matching. Figure 4.1.a shows this extension for $k = 4$.

We now construct a maximum matching on $\mathcal{I}^{\mathcal{R}} \times \mathcal{I}^{\mathcal{C}}$ of the same cardinality as that on $\mathcal{I}^{\mathcal{R}} \times \mathcal{J}^{\mathcal{C}}$. Clearly any cycle gives corresponding matching edges in $\mathcal{I}^{\mathcal{R}} \times \mathcal{I}^{\mathcal{C}}$. Consider a maximum open path $(i_1, \ldots, i_k)$, $i_1, \ldots, i_{k-1} \in \mathcal{I}$, $i_k \in \mathcal{J} \setminus \mathcal{I}$. Then we can replace the matching edges corresponding to this path by $(i_1^{\mathcal{R}}, i_2^{\mathcal{C}}), (i_2^{\mathcal{R}}, i_1^{\mathcal{C}}), \ldots, (i_{k-2}^{\mathcal{R}}, i_{k-1}^{\mathcal{C}}), (i_{k-1}^{\mathcal{R}}, i_{k-2}^{\mathcal{C}})$ all of whose end points are in $\mathcal{I}$. It is illustrated by the example in Figure 4.1.b for $k = 5$.

This then gives us a maximum matching on $\mathcal{I} \times \mathcal{I}$ of cardinality $|\mathcal{I}|$. $\square$

We relax the `MC64` constraints on the scaling. Let D_r and D_c be the scaling factors returned by `MC64` on $\tilde{A}$. We build D so that if i corresponds to an index of $\tilde{A}$, $d_i = \sqrt{d_{r_i} d_{c_i}}$, otherwise

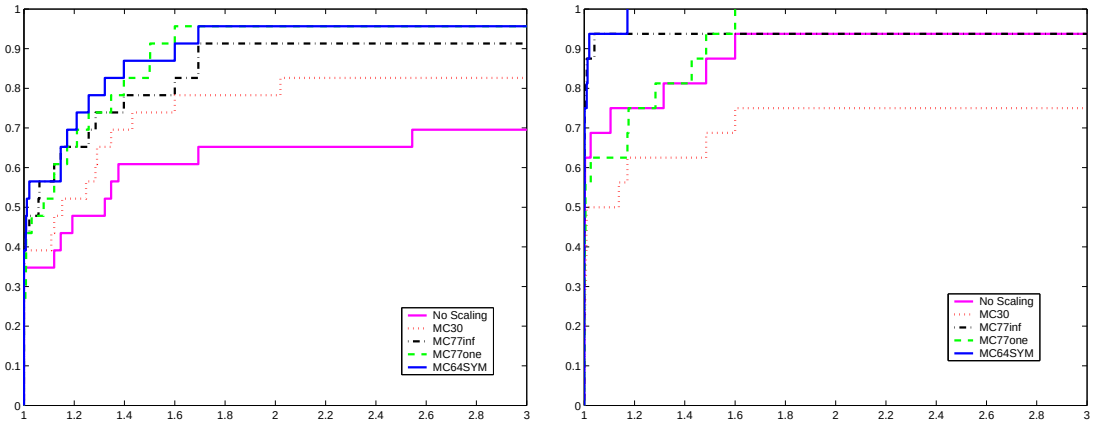
$$d_i = \frac{1}{\max_{k \in index(\tilde{A})} |a_{ik} d_k|} \text{ with the convention } \frac{1}{0} = 1.$$

Note that the entries of $|DAD|$ are less than 1 and entries in the matching are 1. Moreover the maximum entry in absolute value in each non-empty row/column is 1.

4.3 Influence of scaling

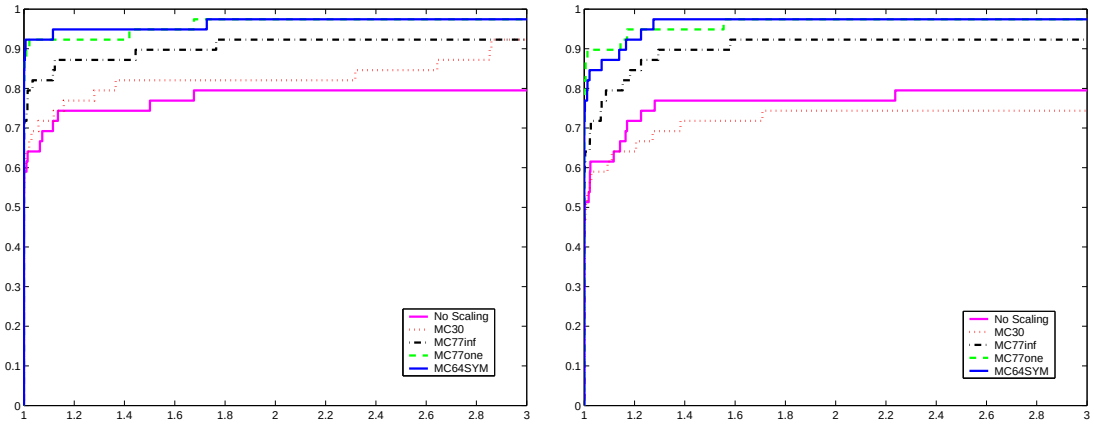
It is important to note that scaling does not change the pivot order returned by the analysis but changes the selection of the numerically stable 1×1 and 2×2 pivots during the factorization. Scaling can have a profound affect on the subsequent factorization. A good scaling can avoid many numerical difficulties whereas a bad scaling can actually cause numerical problems, can produce a lot of delayed pivots when not necessary, can increase the memory requirements, and can consequently severely slow down the factorization.

On sets 1 and 2, the approach without any scaling (No Scaling) fails on 6 matrices, the approaches using `MC30`, `MC77inf`, `MC77one` and `MC64SYM` fail on 3, 3, 1 and 1 matrices respectively (see Table 4.1). The use of scaling thus improves the robustness of the `MA57` factorization in terms of the number of successful computations. Table 4.1 shows that most of the time failures are due to numerical pivoting which increases the CPU time and memory requirement (CPU and MEM failures). It seems difficult to get a good solution for the BRATU3D problem, except with the `MC77one` scaling. Figure 4.2 compares the influence of the different scalings on the CPU factorization time of `MA57` on sets 1 and 2.



(a) Set 1. (b) Set 2.

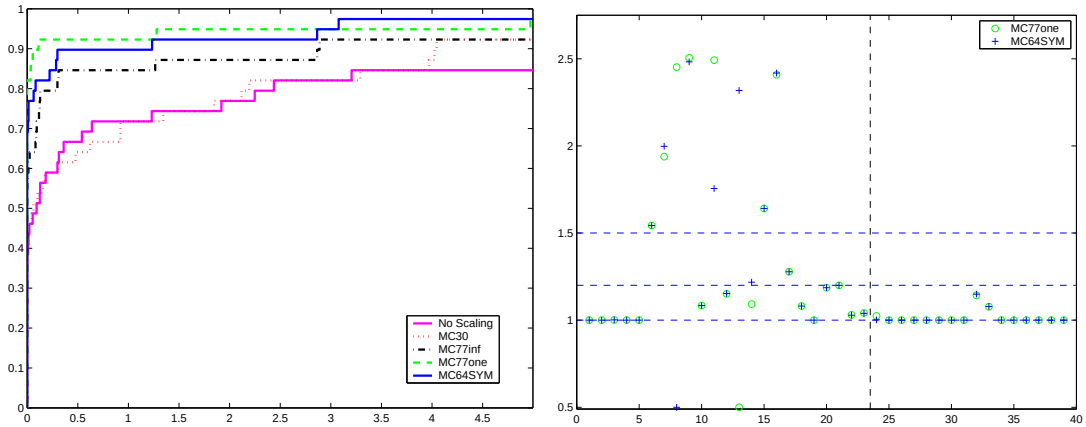
Figure 4.2: Profile showing influence of scaling on CPU factorization time (`AMD` ordering).



(a) Memory profile. (b) Total number of operations profile.

Figure 4.3: Profile showing influence of the scaling on memory and number of operations for the factorization (Sets 1 and 2 together).

On set 2 (Figure 4.2.b), the MC30 scaling degrades the MA57 performance, whereas the other scalings have a positive effect on these general indefinite matrices. The negative effect of MC30 on indefinite problems has also been observed by [20]. Contrary to the other scalings (MC64, MC77), MC30 does not take into account structural information when doing the scaling and uniformly tries to scale all entries to be as close to one as possible. We suspect that the structural decision of the analysis is less compatible with the MC30 scaling. This difference can provide a partial answer about the degradation of performance when using MC30 scaling. We recommend that the MC30 scaling is not used on symmetric indefinite problems. Furthermore, the MC64SYM scaling seems to be the most robust scaling on this set (the MC77inf approach is also good but only for 95% of the time, whereas the MC64SYM scaling is within a factor of 1.17 of the best on 100% of the problems). MC64SYM and MC77one have a similar behaviour on the augmented systems and are slightly better than MC77inf (see Figure 4.2.a).



(a) Number of delayed pivots profile. $\log_{10}$ scale is use for the x-axis (number of delayed pivots).

(b) Ratio between memory used and memory predicted by the analysis on the y-axis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis, set 1 on the left, set 2 on the right.

Figure 4.4: Profile showing influence of scaling on number of delayed pivots and memory (AMD ordering).

The CPU factorization times in Figure 4.2 are linked with the number of operations (Figure 4.3.b), the memory used (Figure 4.3.a) and the number of delayed pivots. Figure 4.4.a shows that the number of delayed pivots is sometimes very large when using the MC30 scaling. The average behaviour of MA57 using the MC30 scaling is similar to applying no scaling. We show in Figure 4.4.b the ratio between the memory used by the factorization and the memory predicted by the analysis. This indicates that the memory predictions of the analysis are sometimes not respected.

It also indicates that most of the numerical problems appear on set 1 (to the left of the vertical line). Indeed there are no problems in set 2 over the 20% limit. If we allocate only 50% more memory than recommended by the analysis instead of 4 GBytes, we would have 7 additional failures with the `MC64SYM` and `MC77one` scalings (points above the highest horizontal dotted line in Figure 4.4.a).

Matrix	NoScaling	`MC77inf`		`MC77one`		`MC64SYM`	
	Facto	Facto	Scal	Facto	Scal	Facto	Scal
Augmented systems (set 1)							
BLOWEYA	CPU	0.08	0.04	0.09	0.04	0.08	0.03
BRATU3D	PREC	PREC	–	77.6	0.06	PREC	–
NCVXQP1	108.	13.6	0.03	19.3	0.01	12.8	0.43
NCVXQP5	MEM	138.	0.19	140.	0.18	139.	2.94
NCVXQP7	MEM	MEM	–	CPU	–	1307	21.9
bloweybl	CPU	0.06	0.03	0.08	0.04	0.06	0.03
cvxqp3	263.	30.6	0.03	37.0	0.04	37.0	0.94
stokes128	2.03	0.81	0.21	0.79	0.16	0.81	0.29
stokes64	0.18	0.14	0.04	0.14	0.01	0.13	0.06
tuma2	0.08	0.08	0.01	0.06	0.01	0.04	0.01
TOTAL solved	18/23	21/23		22/23		22/23	
General symmetric indefinite (set 2)							
BOYD1	CPU	CPU	–	39.9	0.44	33.9	14.8
c-68	24.5	23.1	0.29	28.5	0.21	22.2	0.13
c-71	76.4	76.2	0.48	108.	0.29	76.4	0.28
vibrobox	1.66	1.26	0.04	1.29	0.06	1.28	0.03
TOTAL solved	15/16	15/16		16/16		16/16	

Table 4.1: Factorization time (columns Facto) and scaling time (columns Scal). Times in seconds. `AMD` ordering used. Number of steps for `MC77` set to 20. CPU: the maximum CPU time exceeded. MEM: `MA57` ran out of memory. PREC: problem in precision of the solution.

Table 4.1 summarizes the `MC64SYM`, `MC77inf` or `MC77one` scaling impact on the factorization of symmetric indefinite problems. Using the `MC64SYM` scaling, we get infinite speedup on 5 problems, a speedup greater than 5 on 7 matrices, a speedup of between 2 and 5 on two matrices. This approach is faster than no scaling on 30 matrices and on 9 others it never exceeds the factorization time for no scaling by a factor of more than 1.25. We see also that on set 2, the `MC77one` scaling has problems whereas `MC64SYM` does not (see c-68 and c-71 matrices). As mentioned before, on set 2, the same order of speedup is obtained with the `MC77inf` scaling even if it is less robust in terms of the number of failures.

It may be interesting to use the `MC77inf` scaling on the set 2 and the `MC77one` scaling on the set 1 because they have a comparable influence on the factorization and are less costly to compute (see Table 4.1). Nevertheless, `MC64SYM` remains competitive. Firstly, because the maximum number of iterations of `MC77` has been set to 20 in our experiments and the norms of the rows and columns may still be far

from 1 (for example with `MC77one` on the set 2). A safe approach with `MC77` could be to check the convergence every 20 steps and continue iterating until we are close to the limit. It would involve an additional cost that is not included in Table 4.1. On the contrary, when `MC64SYM` has finished, it guarantees that our scaled matrix satisfies Property 4.1. Secondly, because the scaling time has to be considered relative to the factorization time, Table 4.1 shows that the `MC64SYM` cost is usually compensated for by the gain in factorization time.

In the rest of this paper, we try to improve the CPU factorization time and to obtain a better behaviour with respect to other criteria (for example, the quality of the memory prediction). A first way of improving the `MA57` factorization comes from using other orderings (MeTiS and `AMF`) to decrease the fill-in, the memory and the number of operations. A second way is to influence *a priori* the ordering with numerical and structural information about the 2×2 pivots. The main goal of this approach is to preselect the pivots that will be effectively used during the factorization and thus decrease the number of delayed pivots. As `MC64` is used in the approach that tries to answer the second point (*ie* the use of the `MC64SYM` scaling does not involve additional computation) and as the `MC64SYM` scaling seems to be a robust approach on both sets. We will systematically use this scaling in the remainder of this paper. From now on, A will refer to the matrix symmetrically scaled by `MC64SYM` (its entries are in $[-1, 1]$).

5 Ordering and weighted matching

In this section, we present ordering approaches that aim at decreasing fill-in in the factors and at preselecting good pivots during the analysis. Thus we are interested in an algorithm that gives us a good approximation about what will happen during the factorization and that does not generate too much fill-in in the factors. We will first discuss our choices in Section 5.1. Then we present our different approaches. The matrix

$$A33(x) = \begin{pmatrix} 0 & 1 & 1 & x & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ x & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

that has two 3×3 cycles with edges of weight 1 will be used in this section to illustrate our discussions.

5.1 Symmetric maximum weighted matching

In the unsymmetric case, a maximum weighted matching of the **bipartite graph** associated with the matrix is computed to put large entries onto the diagonal of

the permuted matrix. A natural way of adapting this to the symmetric case is to search for a **symmetric maximum weighted matching**, $\mathcal{M}$ (a maximum weighted matching such that if (i, j) is in $\mathcal{M}$ then (j, i) is in $\mathcal{M}$). Property 5.1 presents a method to get a symmetric maximum weighted matching. We will then explain why we decided not to use this method and to build only an approximation of it using the `MC64` maximum weighted matching. The fact that using `MC64` may not return the optimum can be seen in the example $A33(10^{-1})$. `MC64` returns the matching $\mathcal{M} = \{(1, 2), (2, 3), (3, 1), (4, 5), (5, 6), (6, 4)\}$ where we use the convention that, on a bipartite graph, the first index of an edge corresponds to a row vertex and the second index to a column vertex. However, the symmetric maximum weighted matching is $\mathcal{M}_s^{opt} = \{(1, 4), (4, 1), (2, 3), (2, 3), (6, 5), (5, 6)\}$. It is impossible to build $\mathcal{M}_s^{opt}$ using only information in $\mathcal{M}$ because $(1, 4)$ does not appear in it.

Property 5.1 *The problem of finding a symmetric maximum weighted matching is equivalent to the problem of finding a maximum weighted matching on an undirected graph.*

Proof: Let $\mathcal{G}_A = (V_A, E_A)$ be a weighted graph associated with the symmetric matrix A. Note that it is not a bipartite graph. We first define an undirected graph $\mathcal{G}$ with twice the number of vertices as $\mathcal{G}_A$.

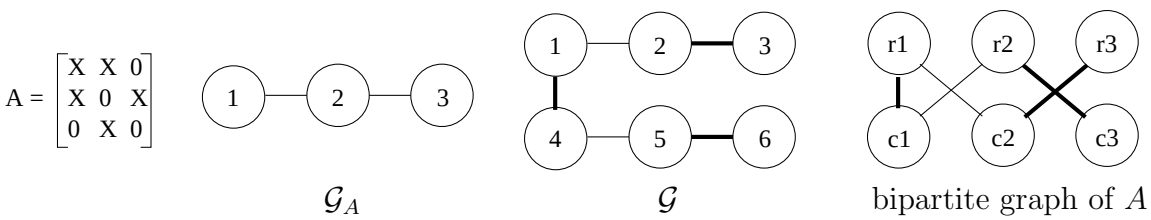


Figure 5.1: Computation of a symmetric maximum weighted matching using a maximum weighted matching on an undirected graph. Bold edges correspond to matching entries.

Let

$$\begin{aligned} V'_A &= \{i + n \text{ such that } i \in V_A\}, \\ E'_A &= \{(i + n, j + n) \text{ with weight } |a_{ij}| \text{ such that } (i, j) \in E_A\} \\ \text{and} \quad E_{diag} &= \{(i, i + n) \text{ with weight } |a_{ii}| \text{ such that } i \in V_A \text{ and } a_{ii} \neq 0\}. \end{aligned}$$

Then $\mathcal{G} = (V_A \cup V'_A, E_A \cup E'_A \cup E_{diag})$ is the required weighted undirected graph. Figure 5.1 illustrates this construction on a small 3×3 example. Note that although $\mathcal{G}$ is not bipartite, there is a simple correspondence between this undirected graph

and the bipartite graph of A. Let $\mathcal{M}_0$ be a maximum weighted matching on $\mathcal{G}$. $\mathcal{M}_0$ can be split into three parts:

$$\begin{aligned} \mathcal{S}_1 = & \ \mathcal{M}_0 \cap E_A, \\ \mathcal{S}_2 = & \ \mathcal{M}_0 \cap E'_A \\ \text{and} \quad \mathcal{S}_d = & \ \mathcal{M}_0 \cap E_{diag}. \end{aligned}$$

We have $\omega(\mathcal{S}_1) = \omega(\mathcal{S}_2)$ (ω as defined in Section 4.2.1), otherwise $\mathcal{M}_0$ would not be a maximum weighted matching (because for example if $\omega(\mathcal{S}_1) > \omega(\mathcal{S}_2)$ then the matching $\mathcal{M}_1$ below has larger weight than $\mathcal{M}_0$). $\mathcal{M}_1 = \mathcal{S}_1 \cup \mathcal{S}'_1 \cup \mathcal{S}_d$ is a maximum weighted matching on $\mathcal{G}$ where

$$\mathcal{S}'_1 = \{(i+n, j+n) \text{ such that } (i,j) \in \mathcal{S}_1\}.$$

Let

$$\mathcal{M}_s = \{(i,j) \text{ such that } (i,j) \in \mathcal{S}_1 \text{ or } (j,i) \in \mathcal{S}_1 \text{ or } (i = j \text{ and } (i, j+n) \in \mathcal{S}_d)\},$$

be a symmetric matching on the bipartite graph of A. Suppose for the purpose of deriving a contradiction that $\mathcal{M}_s$ is not a symmetric maximum weighted matching, and let $\mathcal{M}_{opt}$ be a symmetric maximum matching on the bipartite graph such that $\omega(\mathcal{M}_{opt}) > \omega(\mathcal{M}_s)$. We can build the matching

$$\begin{aligned} \mathcal{M}_2 \ = \ & \{(i,j) \text{ such that } (i,j) \in \mathcal{M}_{opt} \text{ and } i \neq j\} \cup \\ & \{(i+n, j+n) \text{ such that } (i,j) \in \mathcal{M}_{opt} \text{ and } i \neq j\} \cup \\ & \{(i, i+n) \text{ such that } (i,i) \in \mathcal{M}_{opt}\}. \end{aligned}$$

so that $\omega(\mathcal{M}_2) > \omega(\mathcal{M}_0)$ giving us a contradiction. Thus, $\mathcal{M}_s$ is a symmetric maximum weighted matching on the bipartite graph of A and solving the problem of the maximum weighted matching on a non-bipartite graph, enables us to solve the problem of the symmetric maximum weighted matching.

Conversely, if we solve the problem of finding a symmetric maximum weighted matching on the bipartite graph of A, using the same kind of transformations as before, we have the solution of the problem of the maximum weighted matching on the non-bipartite graph of A. $\square$

Property 5.1 gives us a way of computing a symmetric maximum weighted matching and says that the complexity of this problem is of the same order as the complexity of the computation of the maximum weighted matching on a graph with the same number of vertices and with the same number of edges as in the bipartite graph of A. Solving the problem of the maximum matching on a non-bipartite graph is much more complicated than on a bipartite one. A solution to

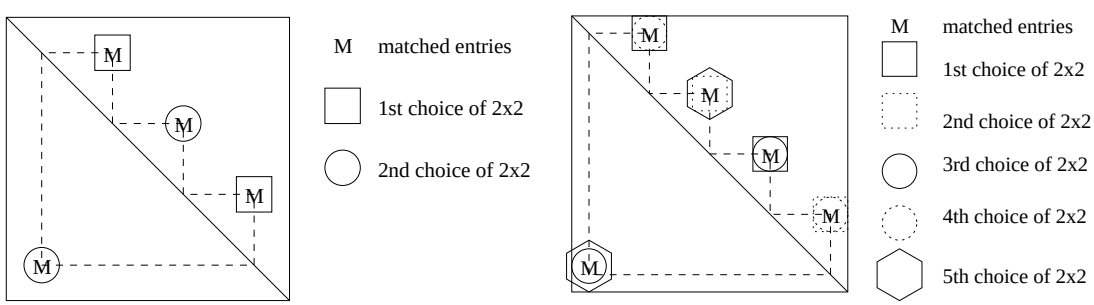
this difficult problem has been found by [17, 18] and its complexity was bounded by $\mathcal{O}(n^4)$. This complexity bound was later decreased to $\mathcal{O}(nnz(A)n \log n)$.

Nevertheless, we will not use Property 5.1 to find good 1×1 and 2×2 pivots. We prefer to find an approximation to the symmetric maximum matching using information returned by `MC64` for the following reasons:

- We want a preprocessing with reasonable complexity with respect to the rest of the analysis and the factorization.
- We do not know an efficient available code for using 5.1.
- In the unsymmetric case, `MC64` has a complexity in the worst case of $\mathcal{O}(nnz(A)n)$ and in practice has a better average behaviour.
- Using `MC64` enables us to compute both the `MC64SYM` scaling and an approximation of a symmetric maximum weighted matching in linear time. We get two complementary preprocessings for the cost of one.

That is why we decided to have a weaker formulation of the problem – we want to find a symmetric weighted matching that is not too far from the optimum – and to use the maximum matching technique on weighted bipartite graphs. [27] shows that, on 33 matrices out of the 39 in our test set, a solution to the weak formulation leads to a solution of the symmetric maximum weighted matching. [27] also gives theoretical bounds as to how far this formulation can be from the symmetric maximum matching.

5.2 Selection of 2 by 2 pivots and symmetric weighted matchings



(a) Example of selection of 2×2 pivots in an even cycle.

(b) Example of selection of 2×2 pivots in an odd cycle.

Figure 5.2: Selection of 2×2 pivots and cycles.

`MC64` is first used to compute a maximum weighted matching $\mathcal{M}$. This will normally have many entries that are not on the diagonal of A. Any diagonal entries

that are in the matching are immediately considered as potential 1×1 pivots and are held in a set $\mathcal{M}_{1\times1}$. We then build a set $\mathcal{M}_{2\times2}$ of potential 2×2 pivots. We use the 2×2 pivot selection strategy suggested by [9], but with a structural metric instead of a numerical criterion. The basis for this strategy is to express the computed permutation, σ, in terms of its component cycles. Because of the scaling, all the entries in the cycles of σ are 1 in absolute value so we choose a structural criterion to select the potential 2×2 pivots. Cycles of length 1 correspond to a matching on the diagonal. k 2×2 pivots can be extracted from even cycles of length $2k$ or from odd cycles of length $2k+1$. For even cycles there are only two possibilities of extraction (see the example in Figure 5.2.a) and for odd cycles of length $2k+1$ there are $2k+1$ possible combinations of 2×2 pivots (see the example in Figure 5.2.b). To resolve the ambiguity present in Figures 5.2 and particularly 5.2.b, for each cycle in σ, we take 2×2 pivots such that the sequence of 2×2 pivots (i_k, j_k) maximizes $\prod metric(i_k, j_k)$ where $metric(i_k, j_k) = |Row_{i_k} \cap Row_{j_k}|/|Row_{i_k} \cup Row_{j_k}|$. This selection can be done in $\mathcal{O}(cn)$ where $\mathcal{O}(c)$ bounds the computation of the metric of each 2×2 pivot (see [27]).

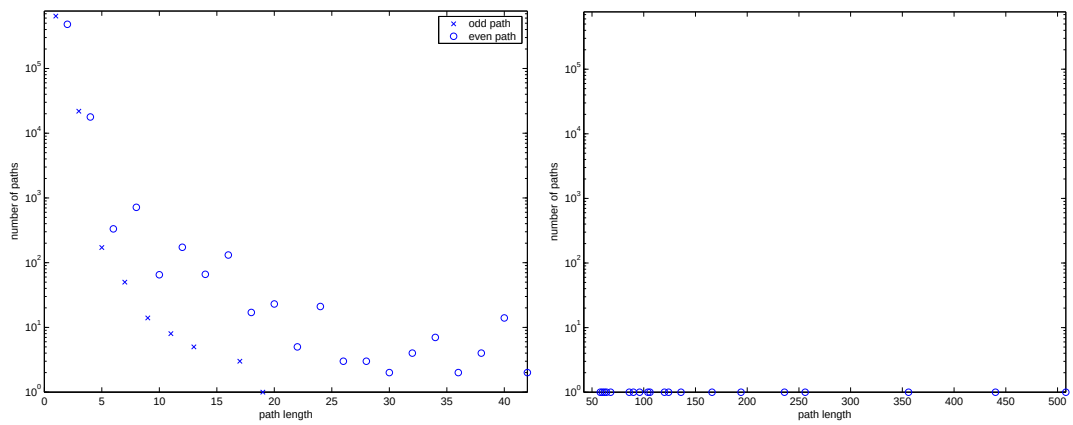


Figure 5.3: Length of `MC64` paths.

At the end of this $\mathcal{M}_{2\times2}$ computation, we add to $\mathcal{M}_{1\times1}$ the nonzero diagonal entries that were not selected during the pass over odd cycles. We define $\mathcal{M}_s$ as $\mathcal{M}_{1\times1} \cup \mathcal{M}_{2\times2}$ corresponding to a symmetric matching obtained from $\mathcal{M}$. We note that $\mathcal{M}_{1\times1} \cap \mathcal{M}_{2\times2} = \emptyset$. Let $\mathcal{M}_s^c$ be the set $\{(i,i)$ such that i does not belong to any pivots in $\mathcal{M}_s\}$, that is the complementary set to $\mathcal{M}_s$. It may be non empty because of odd cycles with zero diagonals. In practice, we note that this set is small ($\mathcal{M}_s^c = \emptyset$ on all the matrices of sets 1 and 2 except the structurally singular ones). Thus we will not investigate approaches to increase the size of $\mathcal{M}_s$ further.

In practice, most of the cycles from the `MC64` permutation are of length 1 or 2.

On our test matrices, 55% of the cycles are of length 1, 41% are of length 2 and less than 4% have a length greater than 2. We illustrate this in Figure 5.3 where we show the total number of paths of varying lengths for runs over all of our test matrices. That is why we will not try different structural criteria to select the 2×2 pivots.

5.3 Coupling detected pivots and orderings

5.3.1 Ordering on the compressed graph

Let $\mathcal{M}$ be a maximum matching on A from which we have obtained a set of 1×1 and 2×2 candidate pivots, $\mathcal{M}_s$. The undirected graph $\mathcal{G}$ associated with the matrix A has a set of vertices V_A, corresponding to the rows (and columns) of A and a set of unordered edges E_A, where the unordered pair $(i,j) \in E_A$ if and only if $a_{ij} \neq 0$.

We define R, the **reduced matrix of** A **relative to** $\mathcal{M}_s$ as the square matrix of order the number of pivots in $\mathcal{M}_s$ whose associated undirected graph will be referred to as the **compressed graph** (it is a generalization of the compressed graph of [2] who only compress indistinguishable vertices). Each of its vertices $\mathcal{I}_i$ is weighted and either associated with a row/column of A corresponding to a candidate 1×1 pivot (weight of 1) or a pair of rows/columns $(i_k,\ i_l)$ corresponding to a candidate 2×2 pivot (weight of 2). The edge set E_R consists of the unordered pairs $(\mathcal{I}_i,\ \mathcal{I}_j)$ where there exists an edge in E_A between one of the constituent vertices of $\mathcal{I}_i$ and one of the constituent vertices of $\mathcal{I}_j$. In other words, the candidate 2×2 pivots of $\mathcal{M}_s$ are compressed into one vertex and the union of their adjacency defines the adjacency of the new vertex. The weight of the vertices will be used to initialize the size of the supervariables and to compute the appropriate metric. For example, the external degree of a supervariable $\mathbf{i}$ will be initialized to $ext_deg(\mathbf{i}) = \sum_{\mathbf{j}\in Adj(\mathbf{i})} |\mathbf{j}|$, where $Adj(\mathbf{i})$ is the set of vertices adjacent to $\mathbf{i}$ in the compressed graph and $|\mathbf{j}|$ is the weight of the vertex $\mathbf{j}$. Obviously, when two supervariables $\mathbf{i}$ and $\mathbf{j}$ are merged – either in the phase of supervariable detection for a greedy ordering or in the coarsening phase for a partitioning, they form a new supervariable of size $|\mathbf{i}| + |\mathbf{j}|$. Note that the rows/columns of A that are not represented in $\mathcal{M}_s$ will not be represented in the reduced matrix.

Algorithm 1 Main steps of the ordering on the compressed graph.

1 Apply a symmetric `MC64` scaling and get $\mathcal{M}_s$.
2 Compute R, a reduced matrix of A relative to $\mathcal{M}_s$.
3 Set P_{red}, the symmetric permutation returned by an ordering on R.
4 Compute P_{aug}, an extension of P_{red} relative to $\mathcal{M}_s$.
5 Put the components of $\mathcal{M}_s^c$ in the last positions.

Let P_{red} be a symmetric permutation on R. Then it is easy to extend P_{red} to a permutation P_{aug} on A by expanding each component corresponding to a 2×2

composite node to the two rows/columns of A associated with that composite node and by putting the rows/columns of A that were not represented in R ($\mathcal{M}_s^c$ entries) at the end of this permutation. This ensures that zero pivots in $\mathcal{M}_s^c$ will be filled by the previous eliminations if we assume that no numerical cancellation occurs and the matrix is structurally nonsingular. Clearly this expansion can be done in a single pass through P_{red}. Note that P_{aug} is not unique: firstly, when a node corresponding to a 2×2 pivot (i,j) is expanded, we have the choice of taking i or j first in P_{aug}; secondly, when we visit the entries in $\mathcal{M}_s^c$, there is no *a priori* order for placing them in P_{aug}. This approach on the compressed graph generates an ordering that is expected to have preselected good numerical pivots. Algorithm 1 summarizes the main steps of this preprocessing.

To illustrate the compression and the expansion, let us take the following matrices:

$$A = \begin{pmatrix} 2 & -1 & 1 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \text{ and } R = \begin{pmatrix} 1 & X & X \\ X & 1 & 0 \\ X & 0 & 2 \end{pmatrix}$$

In A, we detect a 2×2 pivot in rows/columns 3,4 and two 1x1 pivots on the diagonal. We have $\mathcal{M}_s = \{(1,1),(2,2),(3,4)/(4,3)\}$ and $\mathcal{M}_s^c = \{(5,5)\}$. R is a reduced matrix of A relative to $\mathcal{M}_s$ where row/column 3 corresponds to the 2×2 pivot. If $P_{red} = [3,2,1]$ then $P_{aug} = [3,4,2,1,5]$ or $[4,3,2,1,5]$ is an expansion of P_{red}.

5.3.2 Constrained ordering

The main principle of the orderings presented in this section and Section 5.3.3 is to relax the above preselection of the 2×2 pivots by splitting some of them into two 1×1 pivots before the compression. Moreover, when there is a danger of making a bad numerical decision, we will add some dependency about the precedence between some of the 1×1 pivots coming from split 2×2 pivots. We will propose an algorithm that can be implemented in the context of orderings based on local heuristics like `AMD` or `AMF`.

For each 2×2 pivot, $P_{ij} = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}$, with (i,j) ordered such that $|a_{ii}| \geq |a_{jj}|$ and with $|a_{ij}| = 1$, i is said to be the **leading variable** and j, the **trailing variable**. Let $\theta \in [0,1]$ be a real constant. P_{ij} is said to be :

a **locked pivot** according to θ if and only if $|a_{ii}| \leq \theta$ and $|a_{jj}| \leq \theta$,

a **constrained pivot** if and only if $|a_{ii}| > \theta$ and $|a_{jj}| \leq \theta$,

a **splittable pivot** if and only if $|a_{ii}| > \theta$ and $|a_{jj}| > \theta$.

The set of `MC64SYM` detected 2×2 pivots is separated into three sets, LP_θ, CP_θ and SP_θ, the set of locked pivots, constrained pivots and splittable pivots respectively. Note that when $\theta = 0$, locked pivots correspond to *oxo* pivots (two zeros on the diagonal), constrained pivots to *tile* pivots (one zero on the diagonal) and splittable pivots to full 2×2 pivots. When $\theta = 1$, none of the 2×2 pivots is split and the ordering will behave as the ordering on the compressed graph. The more θ decreases, the more pivots are split and the more the risks of making a bad decision increase. In the rest of our discussion, we will omit the θ from our notation.

During the ordering, we manipulate two kinds of (super)variables, **free (super)variables** and **constrained (super)variables**. At the beginning of the ordering, a supervariable i is said to be a free supervariable if and only if i belongs to a splittable pivot, or is the leading variable of a constrained pivot, or belongs to $\mathcal{M}_{1\times1}$, or is a locked pivot, and a supervariable i is said to be a constrained supervariable if and only if i appears as the trailing variable of a constrained pivot.

During the pivot selection, there are two main rules:

(R1) a free supervariable can be eliminated whenever we want (it does not depend on the elimination of another one),

(R2) a constrained supervariable can be eliminated if and only if a free pivot with which it is associated has already been eliminated.

The second rule is equivalent to marking constrained supervariables as free as soon as (R2) is satisfied. Thus, during the ordering, constrained supervariables can become free. If i is the leading variable of a constrained 2×2 pivot, we will say that i **releases** j where j is the variable associated with i in this pivot (it is a sufficient condition to make the supervariable j free). The free supervariables that correspond to entries in $\mathcal{M}_{1\times1}$ or to locked pivots can be eliminated but do not release any constrained supervariables.

At each step of the ordering, FV and CV will be used to denote respectively the set of supervariables that can be eliminated and the set of pivots which cannot be eliminated. For each free supervariable i belonging to a constrained pivot, we define $ass(i)$, its associated constrained supervariable. For other supervariables $ass(i) = \emptyset$.

Algorithm 2 Constrained ordering scheme.

Determine free and constrained supervariables according to $\mathcal{M}_s$.
$CV \leftarrow \{$ constrained supervariables $\}$ and $FV \leftarrow \{$ free supervariables $\}$.
while there are uneliminated supervariables **do**
  $i \leftarrow arg\min_{p \in FV} metric(p)$
  Do symbolic elimination of i.
  $FV \leftarrow (FV \cup \{ass(i)\}) \setminus \{i\}$
  $CV \leftarrow CV \setminus \{ass(i)\}$
end while

Algorithm 2 describes our constrained ordering. At each step of the symbolic elimination, we select the best pivot in the set FV. Moreover, when the leading supervariable of a constrained pivot is selected, its associated constrained supervariable is released (it is inserted in the FV set and removed from the CV set). Intuitively, if the leading part of a constrained pivot is eliminated then it is possible that the modified value of the trailing supervariable becomes large because our scaling ensures that $a_{pq} = 1$ and $a_{pp} \leq 1$, and thus that the corresponding entry becomes numerically acceptable. This constrained ordering has been implemented with the `AMD` and `AMF` orderings. As in the ordering on the compressed graph, we put the variables in $\mathcal{M}_s^c$ at the end of the permutation.

5.3.3 Relaxation of constrained ordering

Algorithm 3 Relaxed constrained ordering scheme.

Define the free supervariables and the constrained supervariables according to the constraint matrix C and $\mathcal{M}_s$.
$CV \leftarrow \{$ constrained supervariables $\}$ and $FV \leftarrow \{$ free supervariables $\}$.
while there are uneliminated supervariables **do**
  $i \leftarrow arg\min_{p \in FV} Metric(p)$
  Do symbolic elimination of i
  $FV \leftarrow (FV \cup C_{i.}) \setminus \{i\}$
  $CV \leftarrow CV \setminus C_{i.}$
end while

This approach uses the same terminology as the previous one. Here we relax the selection of the pivots, that is we enlarge the set of free pivots. We fix a dropping threshold $\theta_{drop} \in [0, 1]$ and build a matrix $C = (c_{ij})$ that is called the **constraint matrix**. This matrix together with $\mathcal{M}_s$ will be used to define the constrained and free supervariables and the relations between them (who releases whom). Firstly the row and column of C that correspond to indices that appear in locked pivots are set to 0. Then the rest of the entries c_{ij} of the constrained matrix are set to 1 if and only if $a_{ij} \geq \theta_{drop}$, otherwise they are set to 0. C describes the dependency among the free and constrained supervariables: i can release j if and only if $c_{ij} \neq 0$. The above construction prevents a locked pivot from releasing a constrained supervariable.

Let $C_{i.}$ be the ith row of C that intersects the current reduced matrix of the Gaussian elimination. Each nonzero off-diagonal entry of C defines a potential 2×2 pivot. A supervariable i is free if and only if it appears in the indices of a potential splittable pivot in C or it is the leading variable of a potential constrained pivot in C or it corresponds to an entry in $\mathcal{M}_{1\times1}$ or it is a locked pivot. Here again the supervariables of $\mathcal{M}_{1\times1}$ and the locked pivots are systematically included in the set of free variables. During the ordering, a constrained pivot j can be selected if and

only if it is reachable from eliminated supervariables in C. In other words it is possible that it has been updated by large enough entries.

Algorithm 3 describes this approach. At each step, the entries in CV that satisfy the above condition are added to FV. We have implemented both the constrained and the relaxed constrained approaches with an `AMD/AMF` ordering, where small changes are made to allow supervariable detection and locked pivot supervariables are initialized with a weight of 2. The total overhead of the manipulation of these sets is $\mathcal{O}(nnz(C))$ which is negligible with respect to the total complexity of the ordering ($\mathcal{O}(n \times nnz(A))$).

We have presented two alternatives to the ordering based on the compressed graph: a constrained ordering and a relaxed constrained ordering. We expect to have less fill-in using the (relaxed) constrained ordering than using the ordering on the compressed graph. However, we expect that the prediction of the ordering on the compressed graph will be more exact than the prediction of the (relaxed) constrained ordering where there is a greater risk of making a bad numerical decision. We will see in Section 6 that these intuitions are verified and that the constrained and the relaxed constrained ordering have a similar behaviour.

6 Experimental results

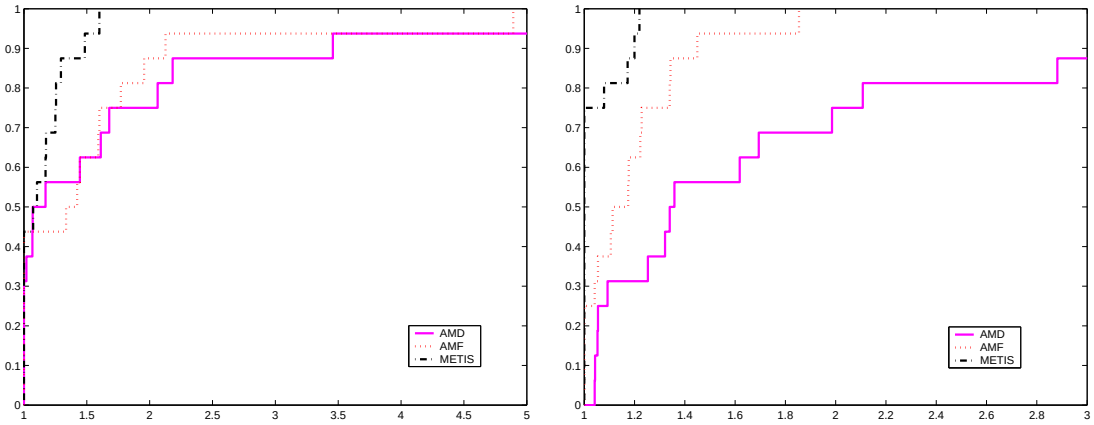
We now consider the effect of our pivoting strategies on `MA57`. We have four approaches from the above strategies.

- `MA57_1` refers to the approach using the `MC64SYM` scaling coupled with an ordering which can be `AMD`, `AMF` or METIS.
- `MA57_2` refers to the approach using the `MC64SYM` scaling, the preselection of the 2×2 pivots and the ordering (`AMD`, `AMF` or METIS) based on the compressed graph (Section 5.3.1).
- `MA57_3` refers to the approach using the `MC64SYM` scaling, the preselection of the 2×2 pivots and the constrained ordering scheme (Section 5.3.2). This strategy is compatible with `AMD` or `AMF`.
- `MA57_4` refers to the approach using the `MC64SYM` scaling, the preselection of the 2×2 pivots and the relaxed constrained ordering scheme (Section 5.3.3). This strategy is compatible with `AMD` or `AMF`.

METIS is called using the routine `METIS_NodeWND` in the `MA57_2` approach and using the routine `METIS_NodeND` in the `MA57_1` approach. In `MA57_3` and `MA57_4`, a threshold of 10^{-2} is used to determine if the diagonal entries correspond to free variables. We tested different values and remark that increasing it from 10^{-2} degrades the fill-in returned by the ordering (there are too many constraints and not

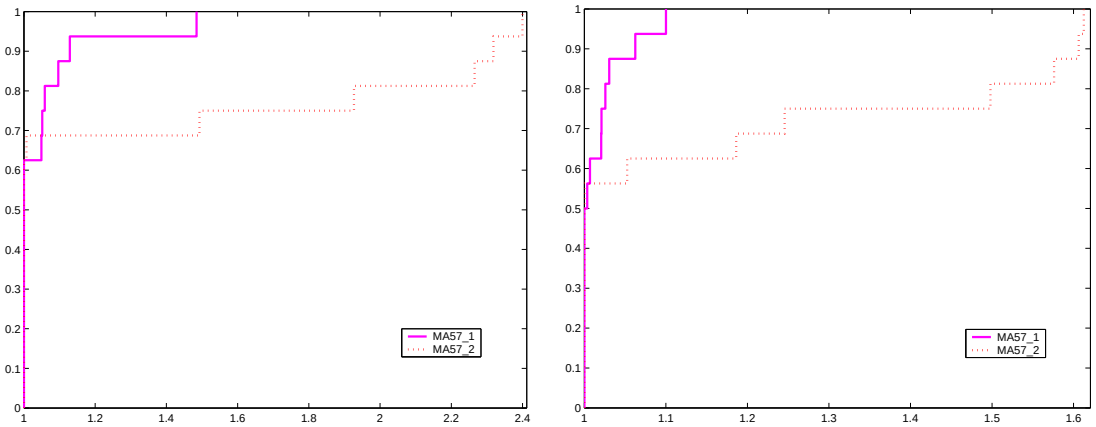
enough free variables), that decreasing it too much from 10^{-2} degrades the memory prediction (on some matrices it did not guarantee good numerical pivots) and did not significantly decrease the fill-in and the number of operations. In `MA57_4`, a threshold of 0.9 is used to determine the constraint matrix. Here also, we tested different values with the same conclusions as above.

6.1 General symmetric indefinite matrices (set 2)



(a) CPU factorization time profile. (b) Memory profile.

Figure 6.1: `AMD`, `AMF`, METIS comparison (set 2). The `MC64SYM` scaling is used.



(a) CPU factorization time profile. (b) Memory profile.

Figure 6.2: CPU factorization time and memory (METIS, set 2).

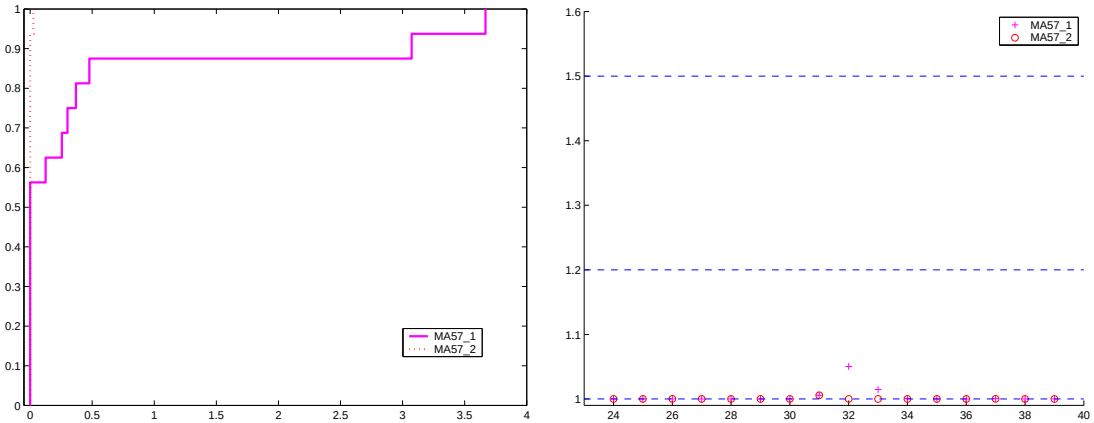
We first determine the best approach on set 2. We will only give details about results for the METIS based ordering for three reasons.

Firstly, METIS is clearly better than AMD and AMF in terms of CPU factorization time (see Figure 6.1.a) and memory (see Figure 6.1.b).

Secondly, in other experiments reported in [27], the relative behaviour between AMD+MA57_1 and AMD+MA57_2 and between AMF+MA57_1 and AMF+MA57_2 are similar to the relative behaviour between METIS+MA57_1 and METIS+MA57_2.

Finally, on this set, we noticed that in early experiments [27] the constrained ordering of Section 5.3.2 and the relaxed constrained ordering of Section 5.3.3 do not significantly improve the CPU factorization time and do not decrease the size of the factors compared with the approach on the compressed graph (Section 5.3.1). On the contrary, we will see in the next section that they clearly improve the factorization on set 2.

Figure 6.2.a shows that MA57_1 is faster than MA57_2 on set 1. This execution time difference is due to the fill-in and the number of operations. Indeed, the approach on the compressed graph merges some rows of the initial matrix and the METIS ordering is called on a coarsened graph. Thus, the MA57_2 pretreatment implies non-negligible constraints on the ordering because of the *a priori* selection of the 2×2 pivots. These constraints severely degrade the quality of the ordering. In particular, they increase the memory requirement (see Figure 6.2.b).



(a) Number of delayed pivots profile (log_{10} scale for x-axis).

(b) Ratio between used memory and memory prediction of analysis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis, set 1 on the left, set 2 on the right.

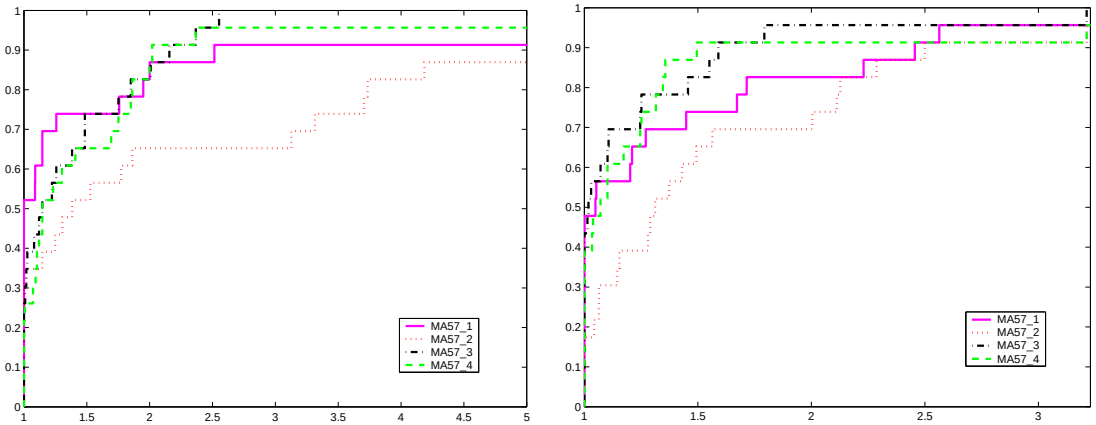
Figure 6.3: Quality of analysis prediction (METIS, set 2).

The MA57_2 analysis succeeds in selecting good pivots for the factorization and delays less pivots than MA57_1 (see Figure 6.3.a). Furthermore, the memory estimation is accurate for both approaches: the ratio between the predicted and the used memory remains close to 1 (see Figure 6.3.b).

6.2 Augmented systems (set 1)

We will now study the behaviour of the MA57 factorization when it is coupled with the three orderings AMD, AMF and MeTiS on set 1 of our test matrices. We will see that the influence of our preprocessing depends on the ordering with which it is associated. We first do relative comparisons between the the four AMD based codes, between the two MeTiS based codes and between the four AMF based codes. Then we select the best approaches and discuss them further.

6.2.1 AMD based approaches



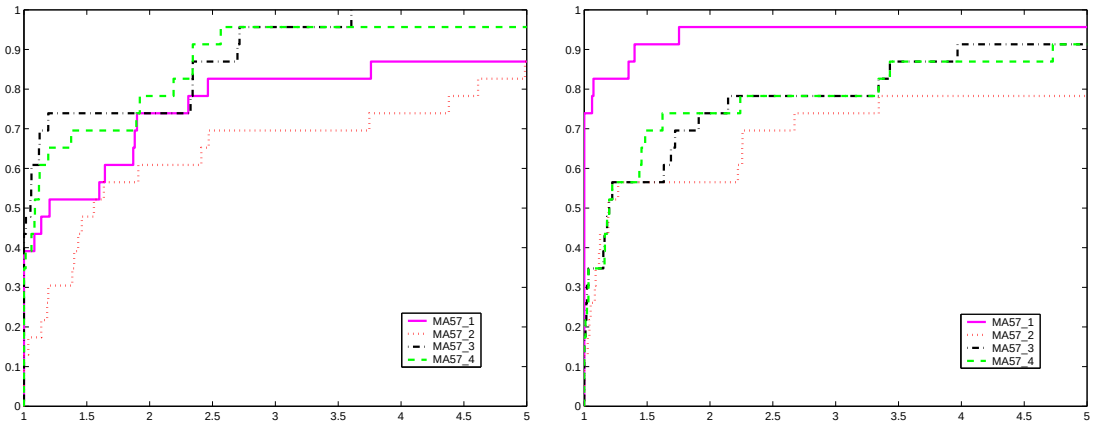
(a) CPU factorization time profile. (b) Memory profile.

Figure 6.4: CPU factorization time and memory (AMD, set 1).

MA57_1, MA57_2 and MA57_4 fail on one matrix (they exceeded the CPU time limit on NCVXQP7) but MA57_3 does not fail. Figure 6.4.a shows that, in terms of CPU factorization time, MA57_1 is within a factor of 1.25 of the best 75% of the time, MA57_3 and MA57_4 are comparable.

MA57_2 can be far from the best ordering because of the constraints that we impose in the analysis phase. That is why it performs more computation and needs more memory than the other codes (see Figures 6.5.a and 6.4.b) and thus is also slower. MA57_3 and MA57_4 do less operations than MA57_1. Thus the total number of operations cannot explain why MA57_1 is faster on 75% of the problems. Further examination shows that MA57_3 and MA57_4 perform more assembly operations (which are more costly than eliminations) than MA57_1 (see Figure 6.5.b). Indeed the assembly operations involve indirect addressing whereas elimination operations call level 3 BLAS dense kernels, so that assembly operations are slower than eliminations, which explains why MA57_3 and MA57_4 are slower than MA57_1.

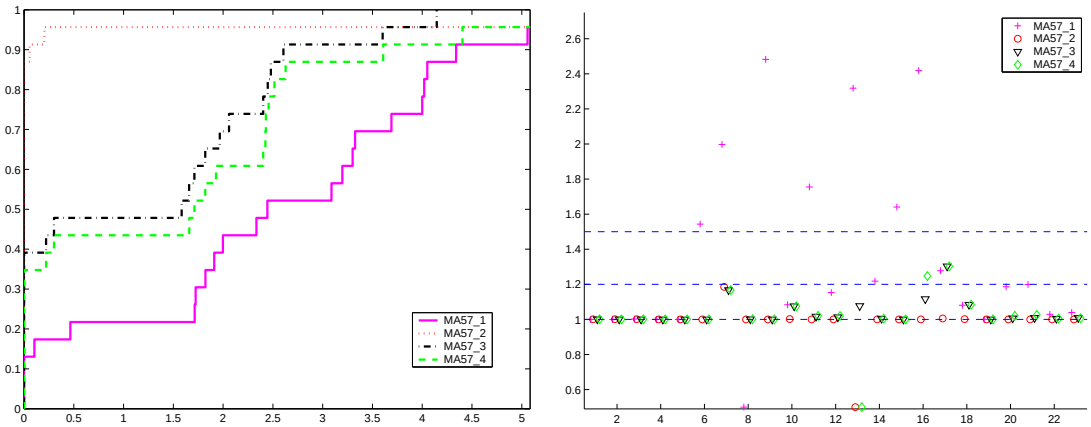
The three new approaches clearly improve the reliability of the analysis predictions. MA57_2 decreases the number of delayed pivots as shown in Figure 6.6.a.



(a) Operations profile.

(b) Assembly operations profile.

Figure 6.5: Number of floating-points operations (AMD, set 1).



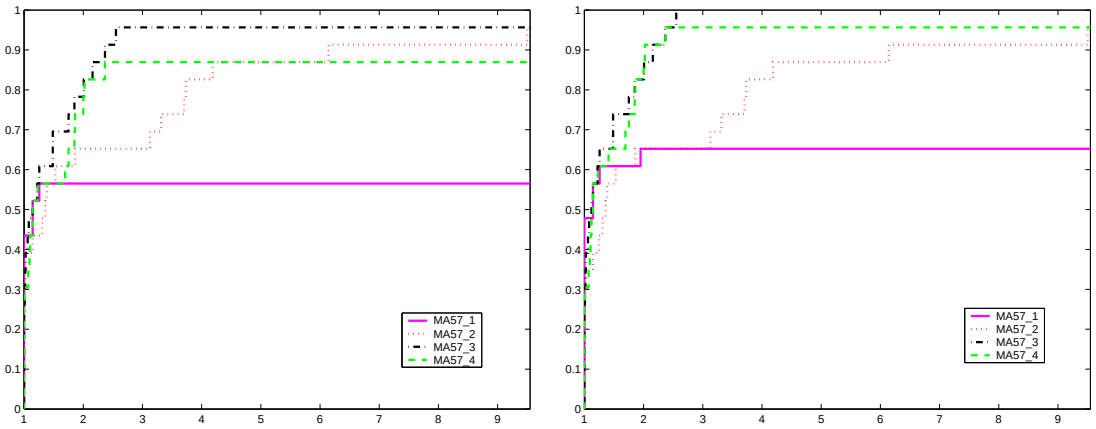
(a) Number of delayed pivots profile (log_{10} scale for x-axis).

(b) Ratio between used memory and memory prediction of analysis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis, set 1 on the left, set 2 on the right.

Figure 6.6: Quality of analysis predictions (AMD, set 1).

Moreover MA57_2, MA57_3 and MA57_4 improve the memory estimations. If a relaxation parameter of 50% is used between the analysis and the factorization the number of failures decreases from 8 to 1, 0 and 1 respectively. (see Figure 6.6.b).

With respect to the above aspects (CPU factorization time, memory, number of delayed pivots and quality of the analysis prediction), MA57_3 seems to be the best if compromises have to be done between CPU time and memory for an AMD based ordering (see Figure 6.7). MA57_1 remains the fastest on most of the problems if a large fixed amount of memory can be allocated for the factorization. That is why,



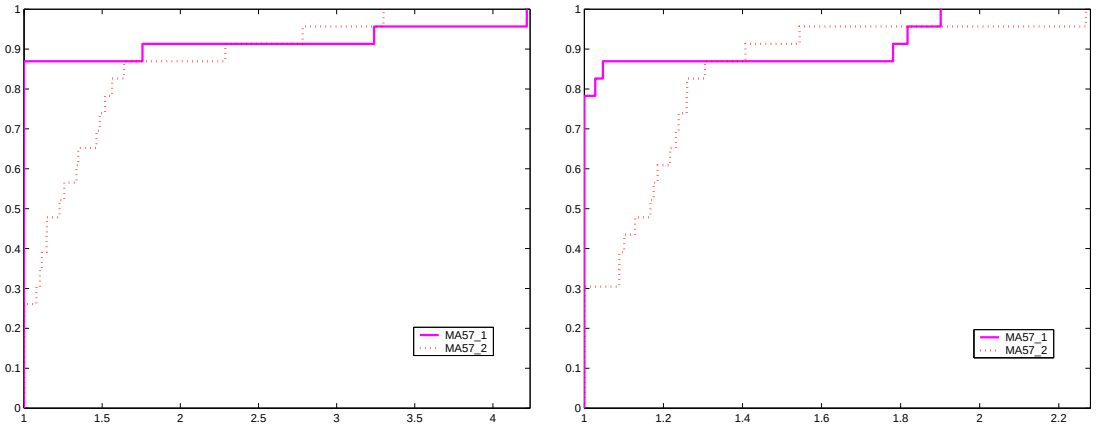
(a) CPU factorization time profile with 20% memory relaxation.

(b) CPU factorization time profile with 50% memory relaxation.

Figure 6.7: CPU factorization time profile with relaxed memory (`AMD`, set 1).

concerning the `AMD` based ordering, we keep the approach with only the `MC64SYM` scaling (`MA57_1`) and the approach with a constrained ordering (`MA57_3`) for our final comparison.

6.2.2 MeTiS based approaches

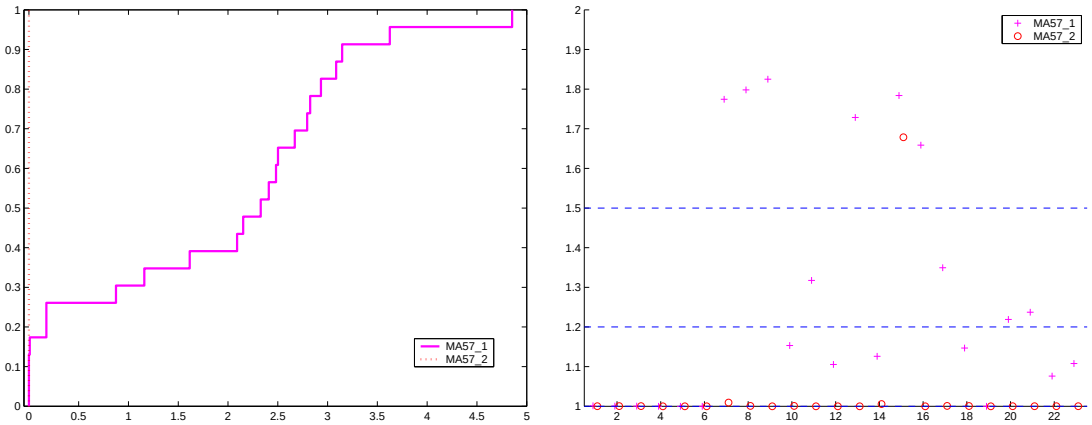


(a) CPU factorization time profile.

(b) Memory profile.

Figure 6.8: Influence on CPU factorization time and memory (MeTiS, set 1).

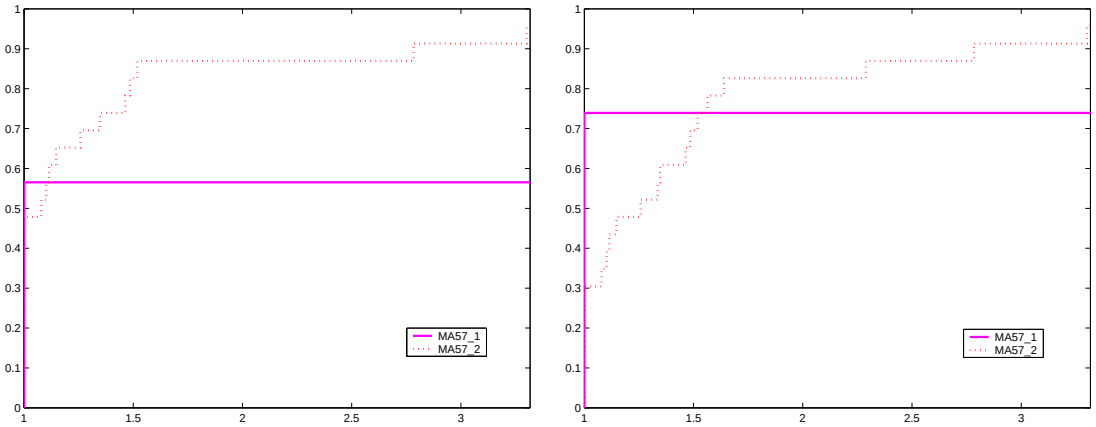
When our approaches are coupled with MeTiS, we did not get any failures. Moreover, `MA57_1` is faster 85% of the time, but `MA57_2` is not too far behind in terms of factorization time and memory usage (see Figure 6.8.a and 6.8.b).



(a) Number of delayed pivots profile (log_{10} scale for x-axis).

(b) Ratio between used memory and memory prediction of analysis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis, set 1 on the left, set 2 on the right.

Figure 6.9: Quality of analysis predictions (METIS, set 1).



(a) CPU factorization time profile with 20% memory relaxation.

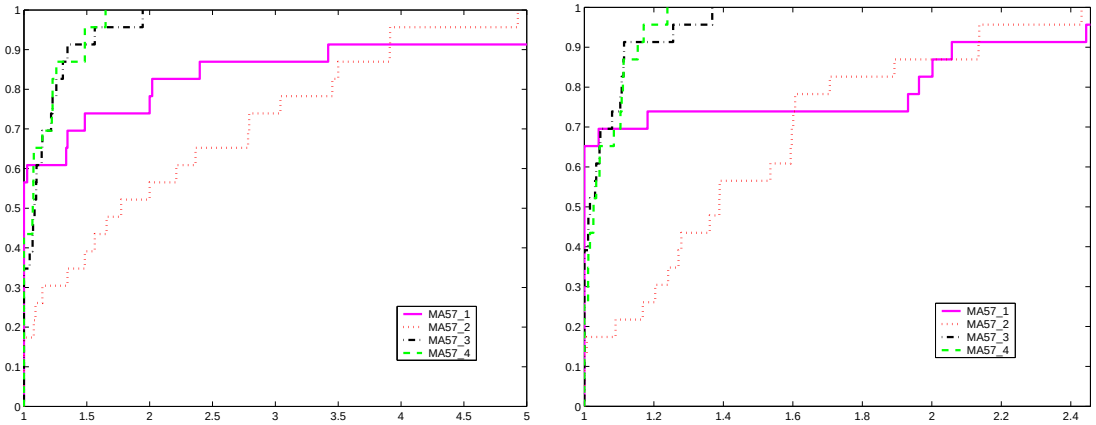
(b) CPU factorization time profile with 50% memory relaxation.

Figure 6.10: CPU factorization time profile with relaxed memory (METIS, set 1).

Figure 6.9 shows that `MA57_1` does not give an accurate prediction for the subsequent factorization. There are 10 and 6 matrices over the 20% and 50% limits respectively. This can be explained by the huge number of delayed pivots (`MA57_1` delays 10000 times more pivots than `MA57_2` in the worst cases, see Figure 6.9.a). On the contrary, `MA57_2` exceeds the 20% limit only once and otherwise the ratio between the memory predicted and the memory needed is always near to one. This better memory estimation is clearly shown by the profile of Figure 6.10 where `MA57_2`

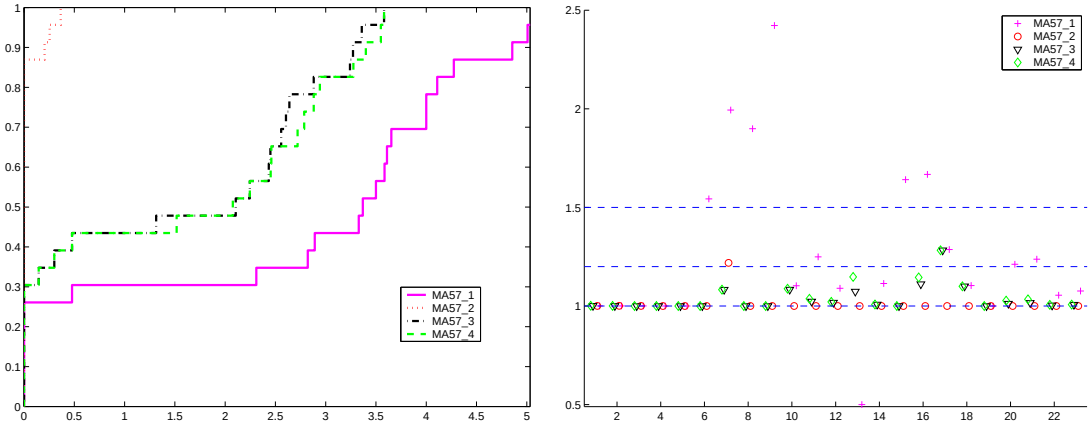
is faster than MA57_1 with 20% memory relaxation. With respect to the above comments, we keep the two MeTiS approaches for our final comparison.

6.2.3 AMF based approaches



(a) CPU factorization time profile. (b) Memory profile.

Figure 6.11: Influence on CPU factorization time and memory (AMF, set 1).

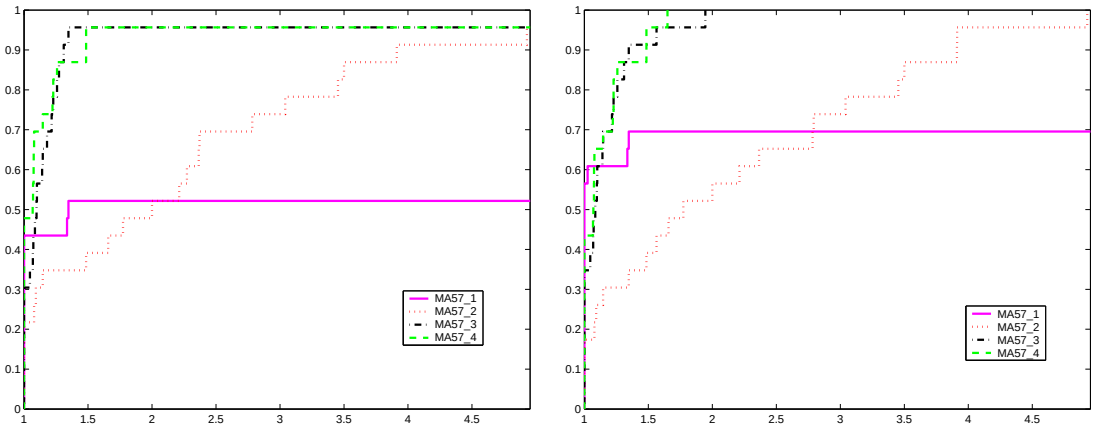


(a) Number of delayed pivots profile (log_{10} scale for x-axis).

(b) Ratio between used memory and memory prediction of analysis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis, set 1 on the left, set 2 on the right.

Figure 6.12: Quality of analysis predictions (AMF, set 1).

We observe a similar relative behaviour between MA57_1 and MA57_2 with an AMF based ordering instead of an AMD based one. The main advantage of an AMF based



(a) CPU factorization time profile with 20% memory relaxation.

(b) CPU factorization time profile with 50% memory relaxation.

Figure 6.13: CPU profile with relaxed memory (AMF, set 1).

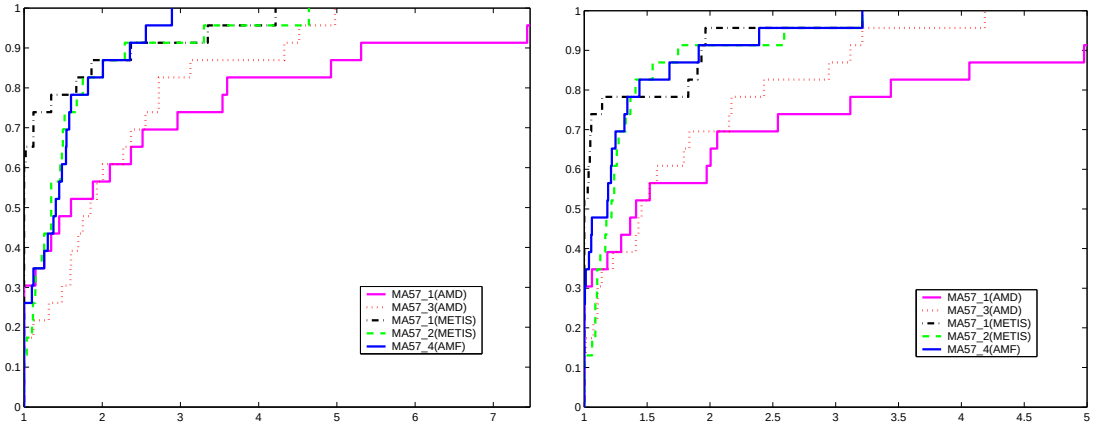
ordering is that MA57_3 and MA57_4 are not penalized by a large number of assembly operations. Thus MA57_3 and MA57_4 are the fastest approaches (Figure 6.11.a) and need less memory than the other approaches (Figures 6.11.b). Moreover they decrease the number of delayed pivots (Figure 6.12.a) and improve the memory estimation (Figure 6.12.b). They decrease the number of failures from seven with MA57_1 to none if 50% relaxation is used between analysis and factorization. That is why they are also the best approaches in terms of CPU factorization time with a fixed memory relaxation as shown in Figure 6.13. For AMF, we only keep MA57_4 for our final comparison.

6.2.4 Best approaches discussion.

In this section, we discuss the best approach for the set 1. We have kept five codes for the comparison because of the earlier discussion: MA57_1/3 with AMD, MA57_1/2 with MeTiS and MA57_4 with AMF.

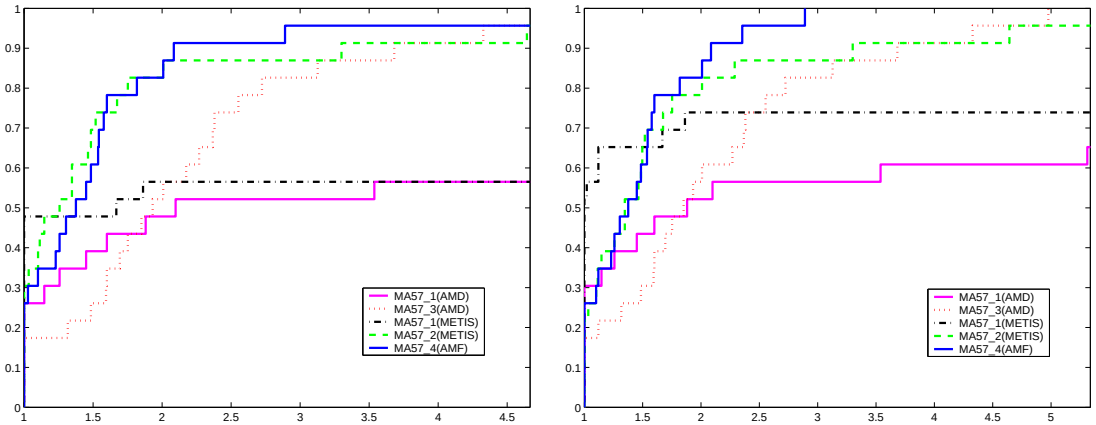
Figure 6.14 shows that the AMD based approaches are the slowest and the most memory consuming. The pure MeTiS based approach is the best on most of the problems and the two other approaches (MA57_2(MeTiS) and MA57_4(AMF)) are comparable. Figure 6.15 presents the factorization time if a relaxation parameter is used between analysis and factorization. With 20% relaxation MA57_2(MeTiS) and MA57_4(AMF) are clearly the best approaches (see Figure 6.15.a). With 50% relaxation MA57_1(MeTiS) is the fastest on many problems and MA57_4(AMF) is the most robust (see Figure 6.15.b).

Thus, if a large amount of memory is available, the approach with only MC64SYM scaling and MeTiS is sufficient. But, if the memory of the factorization needs to be estimated, we recommend the use of the MA57_4+AMF approach.



(a) CPU factorization time profile. (b) Memory profile.

Figure 6.14: AMD/MeTiS/AMF comparison on set 1.



(a) CPU factorization time profile with 20% memory relaxation. (b) CPU factorization time profile with 50% memory relaxation.

Figure 6.15: AMD/MeTiS/AMF comparison (set 1): CPU factorization time profile with relaxed memory.

7 Conclusions and future work

We have shown how MC64 can be used when the matrix is symmetric to effect a symmetric scaling and identify potential 2×2 pivots. We have observed that the use of an appropriate scaling (the MC64 symmetrized scaling, MC64SYM) solves many computational difficulties. We also noticed that MC77 can sometimes be a good alternative and can benefit from its cheap cost. Perhaps this scaling has to be better understood before becoming a default approach.

The performance of MA57 depends very much on the nature of the matrix, but we have shown that it can benefit from this preprocessing, sometimes with no change, or

only minor changes to the analysis. Another benefit of our work is that the analysis phase gives a better indication of the work and storage required by the subsequent factorization. In our future work, we will try to answer the following questions.

How to decrease the fill-in in the MA57_2(MeTiS) analysis while keeping a stable factorization? We have succeeded in decreasing the number of operations in AMD and AMF with a (relaxed) constrained ordering. Fill-in may be lower if we extend this idea to have a tighter coupling with MeTiS. It is not obvious how to do this, so this will be one of our future research directions.

How can we adapt our approaches to a solver like MA47? Indeed, MA47 is designed for augmented systems and will *a priori* do less computation, because it can better manage the *oxo* and *tile* pivots.

How can we design a code with static 2×2 and 1×1 pivoting? Static pivoting can address a large variety of problems in the unsymmetric case when it is coupled with MC64 preprocessing and has the advantage of giving exact memory estimation. It is also more friendly for a parallel distributed implementation. Can we adapt the unsymmetric static pivoting to the symmetric indefinite case and does it address a large range of symmetric indefinite matrices?

Acknowledgments

We are very grateful to Jacko Koster, Daniel Ruiz, and John Gilbert for discussions on various aspects of this work and to Jennifer Scott for her comments on this paper.

References

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

[2] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Comput.*, 16(6):1404–1411, 1995.

[3] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31:162–179, 1977.

[4] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear systems. *SIAM J. Numer. Anal.*, 8:639–655, 1971.

[5] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for Gaussian elimination. *J. Inst. Maths. Applics.*, 10:118–124, 1972.

[6] T. A. Davis. University of Florida sparse matrix collection, http://www.cise.ufl.edu/research/sparse/matrices/, 2002.

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

[8] I . S. Duff and S. Pralet. Experiments in preprocessing and scaling symmetric problems for multifrontal solutions. Technical Report WN/PA/04/17, CERFACS, Toulouse, France, 2004.

[9] I. S. Duff and J. R. Gilbert . Maximum-weight matching and block pivoting for symmmetric indefinite systems. In *Abstract book of Householder Symposium XV, June 17-21*, 2002.

[10] I. S. Duff. MA57 - a new code for the solution of sparse symmetric definite and indefinite systems. Technical report RAL-TR-2002-024, Rutherford Appleton Laboratory, 2002.

[11] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(4):889–901, 1999.

[12] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.

[13] 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.

[14] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.

[15] I. S. Duff and J. K. Reid. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 22(2):227–257, 1996.

[16] Iain S. Duff and John K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL 95-001, Rutherford Appleton Laboratory, 1995.

[17] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards (B)*, 69:125–130, 1965.

[18] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[19] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and sifdec) a constrained and unconstrained testing environment testing environment revisited. Technical Report 2002-009, Rutherford Appleton Laboratory, 2002.

[20] 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. Technical Report RAL-TR-2003-019, Rutherford Appleton Laboratory, 2003. To appear in *ACM Trans. Math. Softw.*

[21] HSL. A collection of Fortran codes for large scale scientific computation, 2004.

[22] 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*. University of Minnesota, September 1998.

[23] Gary Kumfert and Alex Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT*, 37(3):559–590, September 1997.

[24] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.

[25] I. Maros and C. Meszaros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11–12:671–681, 1999.

[26] E. Ng and P. Raghavan. Performance of greedy heuristics for sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 20:902–914, 1999.

[27] Stéphane Pralet. PhD thesis. Technical Report TR/PA/04/xx, CERFACS, 2004.

[28] Edward Rothberg and Stanley C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM Journal on Matrix Analysis and Applications*, 19(3):682–695, 1998.

[29] Daniel Ruiz. A scaling algorithm to equilibrate both row and column norms in matrices. Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, 2001.