Maximum-weighted matching strategies and the application to symmetric indefinite systems

by Stefan Röllin[1], and Olaf Schenk[2]

Technical Report CS-2004-007
Department of Computer Science, University of Basel

Submitted.

[1]Integrated Systems Laboratory, Swiss Federal Institute of Technology Zurich, ETH Zurich, CH-8092 Zurich, Switzerland, email: roellin@iis.ee.ethz.ch

[2]Department of Computer Science, University of Basel, Klingelbergstrasse 50, CH-4056 Basel, Switzerland email: olaf.schenk@unibas.ch

Technical Report is available at
`http://www.computational.unibas.ch/cs/scicomp`

Maximum-weighted matching strategies and the application to symmetric indefinite systems *

Stefan Röllin[1] and Olaf Schenk[2]

[1] Integrated Systems Laboratory, Swiss Federal Institute of Technology Zurich, ETH Zurich, CH-8092 Zurich, Switzerland, roellin@iis.ee.ethz.ch
http://www.iis.ee.ethz.ch

[2] Department of Computer Science Department, University Basel, Klingelbergstrasse 50, CH-4056 Basel, Switzerland, olaf.schenk@unibas.ch
http://www.informatik.unibas.ch/personen/schenk_o.html

Abstract. The problem of finding good numerical preprocessing methods for the solution of symmetric indefinite systems is considered. Special emphasis is put on symmetric maximum-weighted matching strategies. The aim is to permute the large elements to diagonal blocks. Several variants for the block sizes are examined and compared using standard and extended accuracy of the solution. It is shown that these matchings improve the accuracy of direct linear solvers without the additional need of iterative refinements. Especially the use of full blocks results in an accurate and reliable factorization. Numerical experiments validate these conclusions.

1 Introduction

It is now commonly known that maximum-weighted bipartite matching algorithms [5] — developed to place large entries on the diagonal using nonsymmetric permutations and scalings — greatly enhance the reliability of linear solvers [1, 5, 10] for nonsymmetric linear systems. The goal is to transform the coefficient matrix A with diagonal scaling matrices D_r and D_c and a permutation matrix P_r so as to obtain an equivalent system with a matrix $P_r D_r A D_c$ that is better scaled and more diagonally dominant. This preprocessing has a beneficial impact on the accuracy of the solver and it also reduces the need for partial pivoting, thereby speeding up the factorization process. These maximum-weighted matchings are also applicable to symmetric indefinite systems, but the symmetry of the indefinite systems is lost and therefore, the factorization would be more expensive concerning both time and required memory. Nevertheless, modified weighted matchings can be used for these systems. The goal is to preserve symmetry and to define a permutation, such that the large elements of the permuted matrix lie

* This work was supported by the Swiss Commission of Technology and Innovation under contract number 7036.1 ENS-ES, and the Strategic Excellence Positions on Computational Science and Engineering of the Swiss Federal Institute of Technology, Zurich.

on $(p \times p)$ diagonal blocks (not necessarily on the diagonal as for nonsymmetric matrices). This technique based on numerical criterion was first presented by Gilbert and Duff [4] and further extended using a structural metric by Duff and Pralet [6].

We will use these matchings as an additional preprocessing method for the direct solver `PARDISO` [9]. This solver employs a combination of left- and right-looking Level 3 BLAS supernode techniques including supernode Bunch-Kaufmann pivoting for symmetric indefinite systems.

2 Matching strategies for symmetric indefinite systems

We are looking for a symmetric permutation matrix P, such that the large elements of A lie in $p \times p$ diagonal blocks of the matrix PAP^T. The large elements do not necessarily lie on the diagonal as in the nonsymmetric case. As a first step, similar algorithms [5] as in the nonsymmetric case are used to compute the permutation matrix P_r as well as the diagonal scaling matrices D_r and D_c, which are used later to define a symmetric scaling matrix. The permutation corresponding to P_r can be broken up into disjoint cycles $c_1, \ldots, c_k$. The length of the cycles can be arbitrary but they sum up to the dimension of A. These cycles are used to define a symmetric permutation: $\sigma = (c_1, \ldots, c_k)$, which already results into the permutation matrix P, we are looking for. The matrix PAP^T will have k blocks of size $|c_1|, \ldots, |c_k|$ on the diagonal, if the adjacency structure of the nodes corresponding to a cycle are extended to a clique.

In the factorization, the rows and columns corresponding to a diagonal block are treated together. Large diagonal blocks will result in a large fill-in. A first possibility is to keep the k blocks and to put up with the fill-in. Another strategy is to break up long cycles of the permutation P_r to avoid large diagonal blocks. One has to distinguish between cycles of even and odd length. It is possible to break up even cycles into cycles of length two, without changing the weight of the matching due to the symmetry of the matrix. For each cycle, there are two possibilities to break it up. We use a structural metric [6] to decide which one to take. The same metric is also used for cycles of odd length, but the situation is slightly different. Cycles of length $2k + 1$ can be broken up into k cycles of length two and one cycle of length one. There are $2k+1$ different possibilities to do this. The resulting 2×2 blocks will contain the matched elements. However, there is no guarantee that the diagonal element corresponding to the cycle of length one will be nonzero. Another possibility is therefore, to have $k - 1$ cycles of length two and one cycle of length three since there will be more freedom to choose an appropriate pivoting sequence in this case. Again, there are $2k + 1$ different ways to do the division.

As in the nonsymmetric case, it is possible to scale the matrix such that the absolute value of the matched elements is one and the other elements are equal or less than one in modulus. However, the original scalings cannot be used, since they are not symmetric, but they are used to define a symmetric scaling.

The matrix $PDADP^T$ will have the desired properties if we define D as follows: $D = \sqrt{D_r D_c}$, where D_r and D_c are the diagonal matrices mentioned above.

For structurally singular matrices, a permutation matrix P_r and the diagonal scaling matrices D_r and D_c with the desired properties do not exist. In this case, we compute a matching as large as possible and then expand the corresponding permutation to a full permutation, basically by adding the identity permutation. The scaled matrices will have elements less or equal to one.

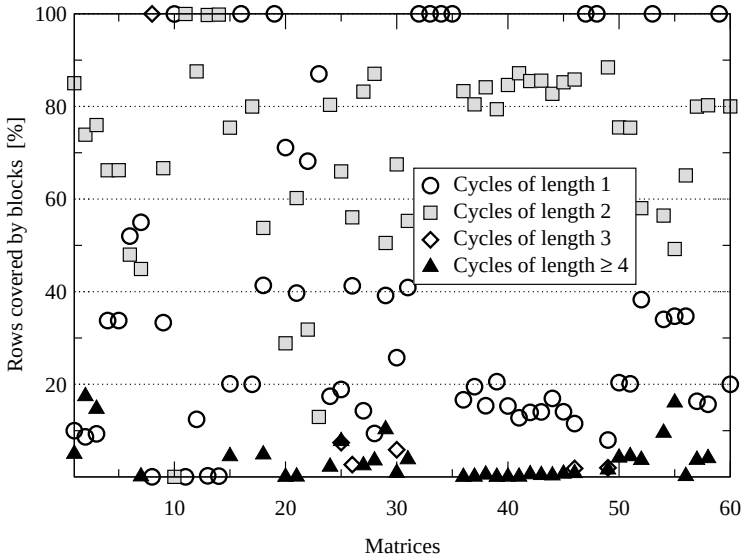

Fig. 1. Distribution of the cycle lengths for the tested matrices.

In section 3 we have used 60 sparse symmetric indefinite matrices for the experiments. These matrices are described in [7]. Figure 1 shows how long the cycles for the matrices are if the cycles are not broken up. In theory, they could be arbitrarily long. However, in practice, most of the cycles c_i are of length one or two. Only a small part of all cycles are longer than two. 40% of all matrices have only cycles of length one and two. In [4, 6], the size of the diagonal blocks has been limited to one or two for the factorization process with `MA57` [3]. Here, we extend these strategies and choose the following block sizes:

1. Do not apply any additional preprocessing based on symmetric matching to the factorization algorithm with sparse Bunch and Kaufmann pivoting in `PARDISO`[8]. This strategy will be called **default**.
2. Do not break up cycles of length larger than two. This will probably result in more fill-in, but allows more choices to perform the pivoting within the diagonal blocks. Therefore, we expect to have better accuracy. This strategy will be called **full blocks**.

3. Divide blocks larger than two into blocks of size one and two, as described above. It is possible that the block of size one contains a zero element, which can lead to a zero pivot. However, contrary to [6], we do not permute this block to the end, but treat it in the same way as all other blocks. We use the name **2x2/1x1** for this choice.
4. For this possibility, an uneven cycle will be divided into blocks of size two and one block of size three. We name it **2x2/3x3**.

Therefore, we can basically test four preprocessing strategies. In addition all methods can be combined with a symmetric scaling based on $D = \sqrt{D_r D_c}$. The strategies will be applied before performing the numerical factorization with Bunch and Kaufmann pivoting within diagonal $p \times p$ blocks in `PARDISO`. The difference between the eight methods is how the diagonal blocks are chosen and whether the matrix is scaled or not. In total, we have tested `PARDISO` with these eight different preprocessing strategies in combination with the 60 symmetric indefinite matrices. A stated above, it is possible to apply the nonsymmetric permutations and scalings directly to the symmetric matrices, but one loses the symmetry for the factorization. We also list the according results for `PARDISO` in nonsymmetric mode for reference. This method is shown with **default, nonsym** in the discussion of the numerical experiments.

3 Numerical Results

For the computation of the symmetric permutation and scaling, we need to know a nonsymmetric permutation P_r and two diagonal scaling matrices D_r and D_c. We have used our own algorithm for the numerical experiments to compute these matrices. Another choice would have been the algorithm MC64 from HSL (formerly known as Harwell Subroutine Library).

The numerical results were conducted in sequential mode on a COMPAQ AlphaServer ES40 with 8 GByte main memory and with four CPUs running at 500 MHz. A successful factorization does not mean that the computed solution is satisfactory, e.g. the triangular solves could be unstable and thus give an inaccurate solution. With respect to this, we use two different criteria to decide whether the factorization and solution process will be successful:

1. Standard accuracy: a factorization is considered successful if the factorization did not fail and the second scaled residual $\|Ax - b\|/(\|A\|\|x\| + \|b\|)$ that has been obtained after one step of iterative refinement is smaller than 10^{-4}. This corresponds to the same accuracy as used by Gould and Scott in [7].
2. Extended accuracy: We say that the factorization has failed if the first scaled residual $\|Ax - b\|/(\|A\|\|x\| + \|b\|)$ is larger than 10^{-11}. We also consider a factorization as not successful, if the residual grows more than one order of magnitude during one iterative refinement step.

The aim of the extended accuracy is to see, which strategy results in an accurate solution without the need of iterative refinements. Furthermore, we check whether the solution process is stable by testing the convergence of the iterative refinement step.

Strategy	Standard Accuracy $\epsilon = 10^{-4}$	Extended Accuracy $\epsilon = 10^{-11}$
default	58	54
default, scaled	54	50
full cycles	60	60
full cycles, scaled	58	57
2x2/1x1	58	58
2x2/1x1, scaled	58	57
2x2/3x3	58	58
2x2/3x3, scaled	58	56
default, nonsym	43	34

Table 1. Number of solved matrices out of 60 for the eight different preprocessing strategies. All matrices are solved with the sparse symmetric Bunch-Kaufmann factorization pivoting method in `PARDISO`. The second and third column shows the number of systems, which are successfully solved, if standard accuracy or extended accuracy for $\|Ax - b\|/(\|A\|\|x\| + \|b\|) < \epsilon$ is used.

3.1 Standard accuracy

From Table 1 we see that the symmetric strategies successfully solve most matrices used in our experiments with the standard accuracy of the solution. Only **full blocks** is able to solve all examples up to the desired accuracy. The other possibilities are of equal quality and fail only in two situations (with the exception of **default, scaled**, which can solve all system). However, the memory requirements, the time to compute the factorization and the accuracy varies between the eight methods. We compare the different strategies using the performance profiling system presented in [2] and which was also used in [7] to compare sparse direct solvers. In these profiles, the values on the y-axis indicate the fraction $p(\alpha)$ of all examples, which can be solved within α times the best strategy.

Figure 2 shows the CPU time profile for the tested strategies, i.e. the time for the numerical factorization as well as the time for the overall solution. For the sake of clarity, we do not show the results for both **2x2/3x3** strategies, since there are only minor differences between them and the results of the strategies **2x2/1x1**. The default symmetric version of `PARDISO` appears to be the most effective solver. It can solve all but two systems, it has the smallest factorization time and the smallest total solution time for the complete solution. Comparing the profiles it can also be seen that the factorization time for the **full cycles** methods are the slowest of all symmetric methods, but it is more reliable than the others. This configuration requires the most memory since columns with different row structure are coupled together resulting in larger fill-in during factorization. Figure 3 compares the first and second scaled residual that has been obtained after one step of iterative refinement. The most accurate solution before iterative refinement is provided by the strategy **full blocks**, followed closely by **2x2/1x1**. The **default** strategy gives the least accurate results of all symmetric version. However, after one step of iterative refinement, the accuracy of all three methods

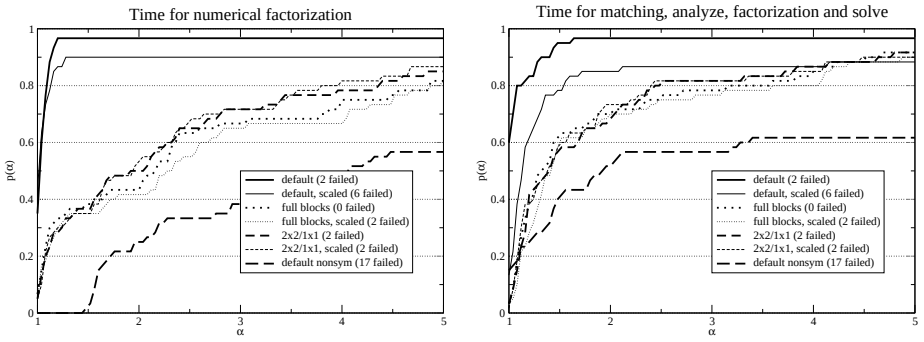

Fig. 2. Performance profile for the CPU time of the numerical factorization, and the CPU time for the complete solution including matching, analysis, factorization and solve.

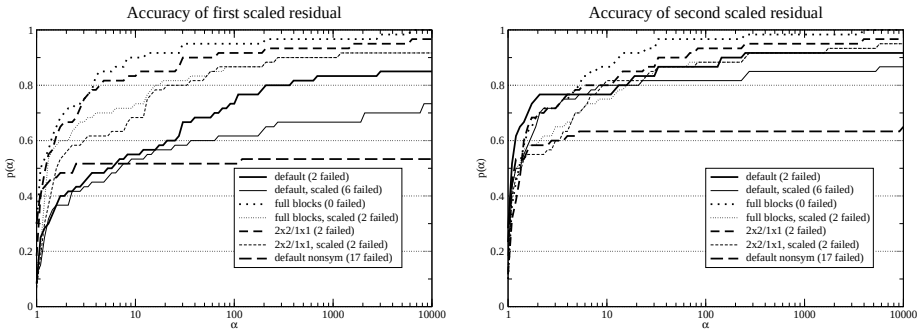

Fig. 3. Performance profile for standard accuracy of the solution before and after one step of iterative refinement. The profile shows the results for the first and second scaled residual up to $\alpha = 10^5$.

is of the same order. It can be concluded that if the factorization time is of primary concern and if one step of iterative refinement can be applied, then the strategy **default** is superior. On the other hand, if the solution process and the stability are important, then the preprocessing method **full cycles** is the most effective method in terms of accuracy.

As expected, the results for the nonsymmetric mode are inferior to the symmetric strategies. Much more time is used for this mode. Some of the matrices are structurally singular, which means that it is not possible to find a permutation such that the diagonal elements are nonzero. However, the nonsymmetric solver relies on a explicit stored (even zero) diagonal element. A part of the failures are due to these circumstances.

3.2 Extended accuracy

Up to now, the **default** strategy of `PARDISO` seems to be the best choice for most of the matrices if the accuracy of the solution is not the first criterion. However, the situation changes if the requirements for the solution are highly

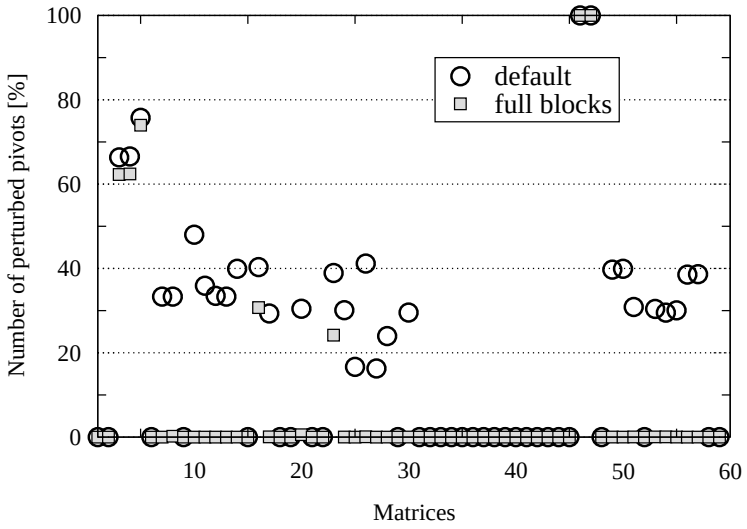

Fig. 4. Number of perturbed pivots during factorization for the tested matrices. The results for the other strategies are similar to **full blocks**.

tightened and extended accuracy is used, as can be seen from Table 1. Again, only the strategy **full blocks** can solve all matrices and the strategies which use matchings solve much more systems than the **default**. From the six matrices which are not successfully solved by `PARDISO` with the default settings without scaling only two can be solved by **default, scaled**. The other strategies **full blocks** or **2x2/1x1** can solve more systems. That the use of blocks is beneficial is not surprising if the number of perturbed pivots during the factorization is considered, depicted in Figure 4. In general, this number decreases by using a strategy with blocks on the diagonal. Especially for those matrices which are not successfully solved by the **default** strategy the perturbed pivots will decrease significantly resulting in high accuracy of the solution by using a **full blocks** or **2x2/1x1** strategy.

In Figures 5 and 6 the performance profiles are given for the CPU time and the extended accuracy of the solution. The best choice is **full blocks** without scalings if the accuracy is important. The strategy **2x2/1x1** is only slightly worse, but on the average requires less memory for the factorization and solves the matrices faster. As before, the strategies with the scalings give less accurate solutions than the others.

In Figure 7 a comparison of the solution time is given. Since the number of nonzeros in the triangular factors for the **default** strategy is slower than for the others, the solution time is also lower. However, the solve time is not significantly affected by the additional preprocessing methods.

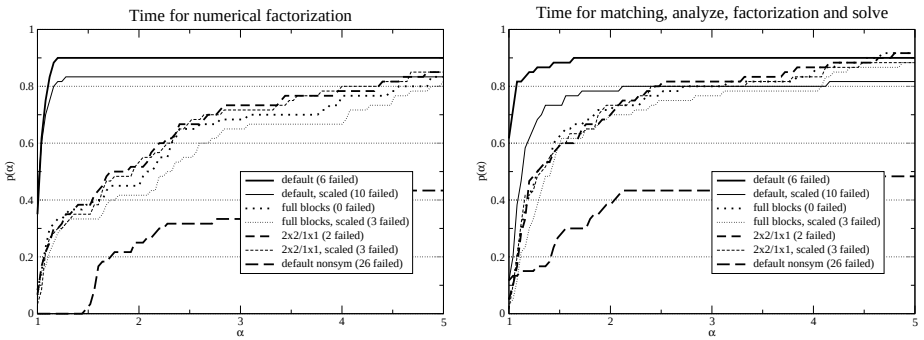

Fig. 5. Performance profile for the CPU time of the numerical factorization, and the time for the complete solution including matching, analysis, factorization and solve using extended accuracy.

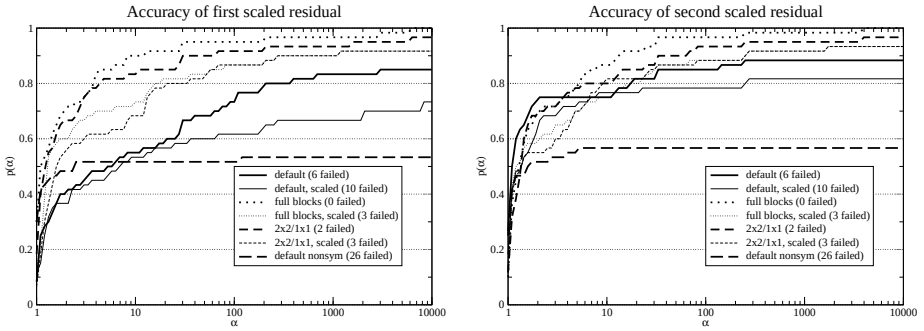

Fig. 6. Performance profile for extended accuracy of the solution before and after one step of iterative refinement. The profile shows the results for the first and second scaled residual up to $\alpha = 10^5$.

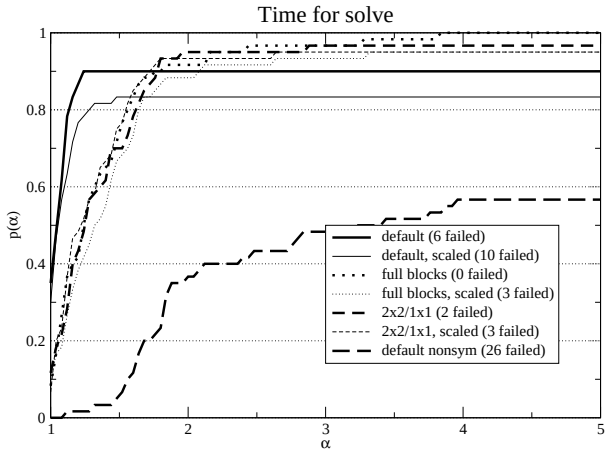

Fig. 7. Performance profile of the CPU time for one solve step using extended accuracy.

4 Conclusion

We have investigated maximum-weighted matching strategies to improve the accuracy for the solution of symmetric indefinite systems. We believe that the current default option in `PARDISO` with one step of iterative refinement is a good general-purpose solver. However, iterative refinement can be a serious problem in cases that involve a high number of solution steps. In these cases matchings are advantageous as they improve the accuracy of the solution without performing iterative refinement. We have demonstrated that the **full blocks** strategy is very effective at the cost of a slightly higher factorization time. Only this strategy is able to solve all examples with the extended accuracy. The solution time increases only slightly by the additional preprocessing methods

References

1. M. Benzi, J.C. Haws, and M. Tuma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Scientific Computing*, 22(4):1333–1353, 2000.
2. E. D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
3. I. S. Duff. MA57 — a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144, June 2004.
4. I. S. Duff and J. R. Gilbert. Maximum-weighted matching and block pivoting for symmetric indefinite systems. In *Abstract book of Householder Symposium XV*, June 17-21, 2002.
5. I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis and Applications*, 22(4):973–996, 2001.
6. I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. Technical Report TR/PA/04/59, CERFACS, Toulouse, France, 2004.
7. Nicholas I. M. Gould and Jennifer A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. *ACM Transactions on Mathematical Software*, 30(3):300 – 325, September 2004.
8. O. Schenk and K. Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. Technical Report CS-2004-004, Department of Computer Science, University of Basel, 2004. Submitted.
9. O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004.
10. O. Schenk, S. Röllin, and A. Gupta. The effects of unsymmetric matrix permutations and scalings in semiconductor device and circuit simulation. *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, 23(3):400 – 411, 2004.