

Extensions of Certain Graph-based Algorithms for Preconditioning

David Fritzsche¹, Andreas Frommer², and Daniel B. Szyld¹

Introduction

We want to solve the linear system

$$Ax = b$$

where $A \in \mathbb{R}^{n \times n}$ is a large and sparse matrix.

We use graph-based algorithms to permute the matrix A and construct preconditioners to standard iterative solvers like GMRES.

PABLO

The PABLO (PArmeterized BLock Ordering, [4]) algorithm uses the structure of a matrix to find a symmetric permutation PAP^T of A . The aim of PABLO is to produce dense diagonal blocks in the permuted matrix. Finding these diagonal blocks is equivalent to finding “clusters” in the graph $D(A)$ of A , which are “well connected”.

PABLO does not try to find the best possible permutation. Instead a “good” permutation is found quickly. The basic principles of PABLO are the following:

- Blocks are build sequentially, i.e. one block at a time. Finished blocks are not touched again later.
- Only vertices adjacent to the current block are tested for possible inclusion.
- Each vertex is tested on its own, independent of the other candidate vertices.

The details can be found in Algorithm 1.

```

1: input: digraph  $D(A) = (V, E)$ 
2: set  $C := \{\text{all vertices in graph}\}$ 
3: set  $Q := \emptyset$  and  $P := \emptyset$ 
4: set  $q := 0$ 
5: while  $C \neq \emptyset$  do
6:   remove a vertex  $v$  from  $C$ , and place it in  $P$ 
7:   move to  $Q$  all vertices in  $C$  adjacent to  $v$ .
8:   while  $Q \neq \emptyset$  do
9:     remove a vertex  $v$  from  $Q$ 
10:    if  $v$  fulfills either PABLO criterion then
11:      insert  $v$  into  $P$ 
12:      move to  $Q$  all vertices in  $C$  adjacent to  $v$ 
13:    else
14:      insert vertex  $v$  into  $C$ 
15:    end if
16:  end while
17:  remove the vertices in  $P$  from the graph, i.e.  $G := G|_C$ 
18:  set  $q := q + 1$ 
19:  designate those vertices in  $P$  to be in a block, i.e. set  $C_q := P$ 
20:  set  $P := \emptyset$ 
21: end while
22: output: a partitioning  $\{C_1, \dots, C_q\}$  of  $V$ 

```

Algorithm 1: The basic PABLO algorithm

Block Sizes

Block sizes are not known a priori in PABLO. The algorithm produces them according to $D(A)$, but there is the option of providing minimum and maximum block sizes.

PABLO Criteria

A key part of PABLO are the criteria used to determine if an eligible vertex v should be included into the current block P . PABLO uses two criteria: The *fullness* and the *connectivity* criteria.

Fullness Criterion:

The “fullness” of $P \cup \{v\}$ is at least α times the fullness of P , i.e.,

$$\varphi_{P \cup \{v\}} \geq \alpha \varphi_P$$

where φ_P , the *fullness* of P , is the number of edges divided by the number of the edges in a clique with the same number of vertices as in P , i.e.

$$\text{fullness} := \frac{\text{number of edges}}{\text{number of all possible edges}}$$

Connectivity Criterion:

A fraction of β or more of all edges of v goes into P , i.e.,

$$\deg|_P(v) \geq \beta \deg(v)$$

where $\deg|_P(v)$, the *degree* (of v) in P , is the degree of v in the induced subgraph $G|_{P \cup \{v\}}$.

TPABLO

PABLO is a combinatorial algorithm taking into account solely the structure of the matrix A and not the values of the entries. TPABLO [1] extends the basic PABLO by taking “large” entries into special consideration, i.e., entries whose absolute value is greater than the given *threshold parameter* γ .

TPABLO comes in two variants, namely TPABLO1 and TPABLO2. They use the following new criteria in addition (not replacing) to the PABLO criteria.

TPABLO1 criterion:

For at least one $i \in P$ we have $|a_{iv}| > \gamma$ or $|a_{vi}| > \gamma$.

TPABLO2 criterion:

For all $i \in P$ we have $|a_{iv}| > \gamma$ and $|a_{vi}| > \gamma$.

XPABLO

In XPABLO we not only consider the graph $D(A)$ of A , but also the graph

$$D^{>\gamma} := \text{remove edges with weight} \leq \gamma$$

Using this *threshold graph* we can formulate more general threshold criteria and can guarantee that XPABLO has a time complexity of $\mathcal{O}(n + \text{nnz}(A))$. We use the $>\gamma$ notation freely to determine degree, fullness and so on in $D^{>\gamma}$.

The Zoo of XPABLO Criteria

Again, let P be the vertices currently in the block and v the eligible vertex. The current implementation support a large variety of logical combination of some or all of the following criteria:

Fullness criterion:	$\varphi_{P \cup \{v\}} \geq \alpha \varphi_P$
Connectivity criterion:	$\deg _P(v) \geq \beta \deg(v)$
Threshold fullness criterion:	$\varphi_{P \cup \{v\}}^{>\gamma} \geq \vartheta$
Threshold connectivity criterion:	$\deg _P^{>\gamma}(v) \geq \zeta \deg _P(v)$

The *flag parameter* f controls the combination of the criteria:

f	v fulfills the XPABLO criterion $\Leftrightarrow \dots$
0	(fulln. or conn.) or (thr. fulln. or thr. conn.)
1	(fulln. or conn.) and (thr. fulln. or thr. conn.)
2	(fulln. and conn.) or (thr. fulln. or thr. conn.)
4	(fulln. or conn.) or (thr. fulln. and thr. conn.)
3 = 1 + 2	(fulln. and conn.) and (thr. fulln. or thr. conn.)
5 = 1 + 4	(fulln. or conn.) and (thr. fulln. and thr. conn.)
6 = 2 + 4	(fulln. and conn.) or (thr. fulln. and thr. conn.)
7 = 1 + 2 + 4	(fulln. and conn.) and (thr. fulln. and thr. conn.)

We can use this flexibility of XPABLO with the parameters f , ϑ and ζ to find the exact same blocks as PABLO or TPABLO:

- PABLO: $f = 4$, $\vartheta = 2$, $\zeta = 1$
- TPABLO1: $f = 1$, $\vartheta = 2$, $\zeta = 1/2n$
- TPABLO2: $f = 5$, $\vartheta = 1$, $\zeta = 1$

The default behavior of XPABLO is very similar to TPABLO1 and uses $f = 0$, $\vartheta = 2$ and $\zeta = 1/2n$.

To reduce the size of the non-threshold graph we have in XPABLO an additional parameter δ . Edges with weight less than δ are removed from the graph $D(A)$. Thus the two graphs XPABLO works on, are $D^{>\delta}$ and $D^{>\gamma}$, where $D^{>\delta}$ takes the role of the non-threshold graph.

Preprocessing

To enhance performance we perform two preprocessing steps, before (T/X)PABLO tries to find blocks.

Scaling

The matrix is scaled by a row-column scaling $A \rightarrow RAC$, s.t.

$$\max_j \left\{ |(RAC)_{ij}| \right\} = 1 = \max_i \left\{ |(RAC)_{ij}| \right\}$$

Transversals

All PABLO variants perform only symmetric permutations. Hence the diagonal entries are only permuted and can not be exchanged with non-diagonal entries. But small entries on the diagonal can have a negative impact on the performance of direct and iterative solvers. Moreover, with small entries on the diagonal it is more likely that the diagonal blocks found by XPABLO are singular, but we need them to be non-singular. Therefore we find an unsymmetric permutation to put large entries into the diagonal.

Maximum Transversal (MT):

Find permutation $A \rightarrow PA$ s.t. $(PA)_{ii} \neq 0$ for all i .

Bottleneck Transversal (BT): [2]

Find a maximum transversal which maximizes

$$\min_{i=1}^n |(PA)_{ii}|$$

As a future refinement we want to replace bottleneck transversals by weighted matchings with scaling (MPS, see [3]). This algorithm finds an unsymmetric permutation matrix Q and scaling matrices D_r and D_c such that QD_rAD_c is an I -matrix. An I -matrix has entries with absolute value one on the diagonal and all non-diagonal entries are at most one.

Preconditioners

We can use PABLO for preconditioning by using the diagonal blocks found by PABLO as a block Jacobi preconditioner.

Spanning Tree Preconditioner

A more sophisticated approach we use to find a good matrix splitting $A = M - N$ for preconditioning uses the (undirected) quotient graph of the graph of A . In the quotient graph we identify all vertices in a block. A simple example is shown in Figure 1.

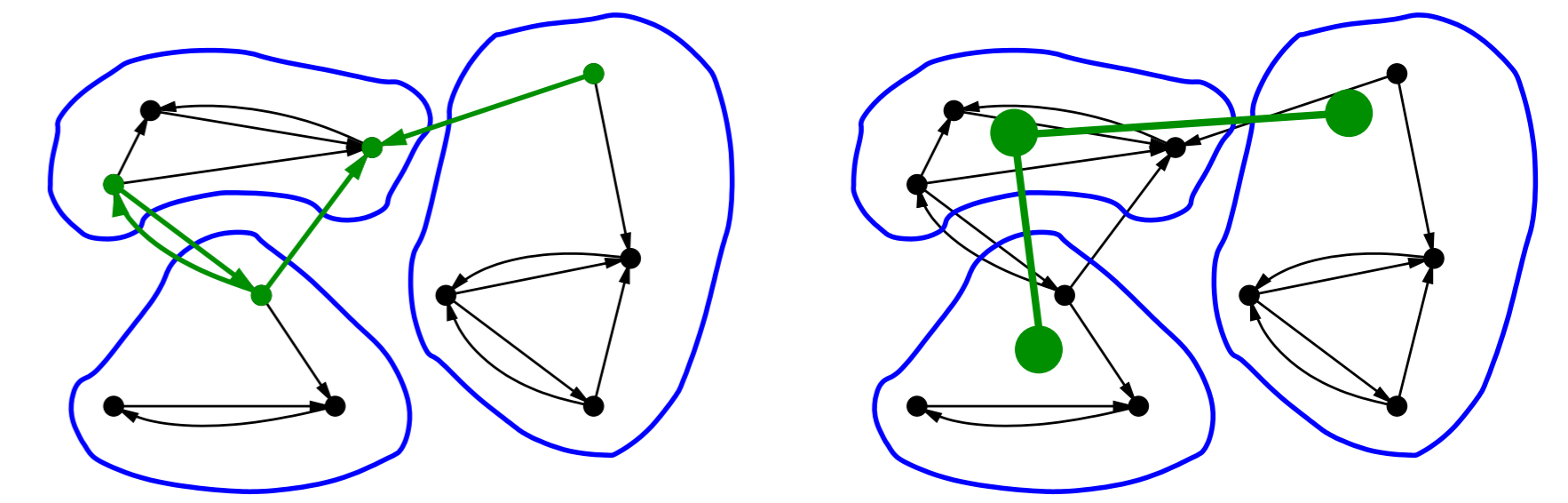


Figure 1: Clusters found by PABLO (blue) and the quotient graph (green)

As the edge weight of an edge in the quotient graph we use the Frobenius norm of the matrix block represented by the edge.

In the quotient graph we find a spanning tree, e.g. by using Kruskal's algorithm. The preconditioner M then consists of the diagonal blocks and the off-diagonal blocks which are represented by an edge in the spanning tree.

We take advantage of the fact that the preconditioner M can be factored without producing any fill-in outside the non-zero blocks, i.e., those represented by the spanning tree.

In our implementation we assume that the non-zero blocks in M are dense and use efficient BLAS3 routines for computations involving M .

Numerical Results

For our experiments we set γ to be the average of the absolute values of the non-zero entries after we have scaled the matrix. The maximal block size is set to be 50, and we do not enforce a minimal block size. The other parameters are set to $\alpha = 1.1$, $\beta = 0.6$, $\delta = 0.05$, $f = 0$, $\vartheta = 2$ and $\zeta = 1/2n$.

Matrix	Times in seconds			
	PAB1	PAB3	ILU1	ILU2
CIRCUIT_4	20.4	6.74	36.41	–
HCIRCUIT	18.5	6.4	1.62	1.02
MEMPLUS	0.16	0.17	0.23	0.15
MULT_DCOP_01	–	6.12	1.5	1.5
MULT_DCOP_02	3.02	1.88	1.24	1.22
MULT_DCOP_03	0.8	1.18	1.2	1.19
NMOS3	–	9.48	0.98	0.68
ZHA01	1.39	2.32	0.36	0.6

Figure 2: Solving times

In Figure 2 we can see the total times needed for solving linear systems using GMRES(50) with several different preconditioners. The convergence tolerance was set to $\sqrt{\epsilon} \approx 1.49 \cdot 10^{-8}$. The preconditioners used are the following:

- PAB1 = XPABLO + block diagonal preconditioner
- PAB3 = XPABLO + spanning tree preconditioner
- ILUT1 = RCM + ILUT(1e-2,5)
- ILUT2 = RCM + ILUT(1e-3,10)

While in most cases, some form of ILUT preconditioner gives better timings, there are cases where XPABLO can not only be competitive but perform much faster. Also the block based XPABLO preconditioners, in particular the simple block Jacobi preconditioner should work well on a parallel computer if we enforce a stricter control of the block sizes.

For completeness we also report the number of total GMRES iterations (Figure 3). A dash indicates that no convergence was reached within 500 iterations. All matrices can be found in the University of Florida Sparse Matrix Collection (<http://www.cise.ufl.edu/research/sparse/matrices/>).

Matrix	n	nnz	Number of iterations			
			PAB1	PAB3	ILU1	ILU2
CIRCUIT_4	80209	307604	152	37	335	–
HCIRCUIT	105676	513072	102	27	15	7
MEMPLUS	17758	126150	2	2	19	9
MULT_DCOP_01	25187	193276	–	93	14	9
MULT_DCOP_02	25187	193276	101	18	13	9
MULT_DCOP_03	25187	193216	19	8	9	7
NMOS3	18588	237130	–	237	35	15
ZHA01	33861	166453	29	25	6	5

Figure 3: Iteration numbers

References

- [1] Hwajeong Choi and Daniel B. Szyld. Application of threshold partitioning of sparse matrices to Markov chains. In *IEEE International Computer Performance and Dependability Symposium IPDS'96, Urbana-Champaign, Illinois*, pages 158–165, Los Alamitos, California, September 4–6, 1996. IEEE Computer Society Press.
- [2] Iain S. Duff and Jacko 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, October 1999.
- [3] Iain S. Duff and Jacko 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, October 2001.
- [4] James O'Neil and Daniel B. Szyld. A block ordering method for sparse matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):811–823, September 1990.

¹Department of Mathematics, Temple University, Philadelphia, PA 19122-6094, USA

²Faculty of Mathematics and Science, Bergische Universität Wuppertal, D-42097 Wuppertal, Germany