HAL open science

Heuristics for a matrix symmetrization problem

Bora Uçar

To cite this version:

Bora Uçar. Heuristics for a matrix symmetrization problem. Proceedings of Parallel Processing and Applied Mathematics (PPAM'07), 2008, Gdansk, Poland. pp.718–727. hal-00803470

HAL Id: hal-00803470

https://inria.hal.science/hal-00803470v1

Submitted on 15 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Heuristics for a matrix symmetrization problem*

Bora Uçar

CERFACS,
42 Avenue Gaspard Coriolis,
31057, Toulouse, Cedex 1, France
ubora@cerfacs.fr

Abstract. We consider the following problem: given a square, nonsymmetric, $(0,1)$-matrix, find a permutation of its columns that yields a zero-free diagonal and maximizes the symmetry. The problem is known to be NP-hard. We propose a fast iterative-improvement based heuristic and evaluate the performance of the heuristic on a large set of matrices.

Key words: unsymmetric sparse matrix; bipartite matching; matrix symmetrization

1 Introduction

We consider the matrix symmetrization problem for square $(0,1)$-matrices. A square matrix A is symmetrizable if its columns can be permuted to yield a symmetric matrix. Deciding whether a given matrix is symmetrizable is NP-complete [4]. The variation of the problem which restricts the permutations such that the permuted matrix has a zero-free diagonal is also NP-complete [2]. In this work, we are interested in the optimization version of the matrix symmetrization with zero-free diagonal problem, i.e., given a square $(0,1)$-matrix, find a permutation of the columns that yields a zero-free diagonal and maximizes the symmetry. We assume that there is at least one permutation that yields a zero-free diagonal.

The problem arises in a preprocessing phase of some other algorithms. For example, when a given sparse matrix A has an unsymmetric pattern, most of the graph partitioning and ordering algorithms are applied to the pattern of the symmetric completion $A + A^T$ (ignoring numerical cancellation); see discussions in [8, 10, 15]. A remark which usually accompanies using the pattern of the symmetric completion is that this trick would be appropriate only if the matrix is nearly symmetric. The techniques proposed in this work can be used to make a given matrix more symmetric and obtain a sparser symmetric completion. In other words, the proposed techniques can help improve the running time of the aforementioned algorithms and their solutions' quality.

Since the decision problem is NP-complete, and hence the optimization version that we are interested is NP-hard, we propose a heuristic algorithm. We test the heuristic on a large set of matrices. We also report encouraging experiments on a certain matrix ordering problem.

* This work was supported by "Agence Nationale de la Recherche", ANR-06-CIS6-010.

2 Method description

In this section, A is a square $(0,1)$-matrix of size $n \times n$. As is common, we associate a bipartite graph $G = (R \cup C, E)$ with the matrix A, where R and C are the two sets in the vertex bipartition, and E is the set of edges. Here, the vertices in R and C correspond, respectively, to the rows and the columns of A such that $(r_i, c_j) \in E$ if and only if $a_{ij} = 1$. Although the edges are undirected, we will always specify an edge $e \in E$ as $e = (r_i, c_j)$; the first vertex will always be a row vertex and the second one will always be a column vertex. An edge $e = (r_i, c_j)$ is called to be incident on the vertices r_i and c_j. We have the following notation: $N(r_i)$ denotes the neighbors of a row vertex r_i, i.e., $N(r_i) = \{c_j : (r_i, c_j) \in E\}$; similarly $N(c_j) = \{r_i : (r_i, c_j) \in E\}$ denotes the neighbors of a column vertex c_j; for a set s, $|s|$ denotes its cardinality; $d(\cdot)$ denotes the degree of a vertex, e.g., $d(r_i) = |N(r_i)|$; and w_{ij} denotes the weight of an edge (r_i, c_j).

We recall some standard definitions and well-known results. An even cycle contains an even number of vertices. A set of edges $\mathcal{M}$ is a matching if no two edges in $\mathcal{M}$ are incident on the same vertex. Given a matching $\mathcal{M}$, an $\mathcal{M}$-alternating cycle is a simple cycle whose edges are alternately in $\mathcal{M}$ and not in $\mathcal{M}$. A matching is called perfect if for any vertex v in G, there is an edge in $\mathcal{M}$ incident on v. If the edges are weighted, then the weight of a matching $w(\mathcal{M})$ is equal to the sum of the weights of its edges, i.e., $w(\mathcal{M}) = \sum_{(r_i,c_j)\in\mathcal{M}} w_{ij}$. A maximum weight perfect matching on a weighted graph is a perfect matching with maximum weight. Both the perfect matching problem and the maximum weight perfect matching problem are efficiently solvable [7, 13]. We use mate(v), to denote the vertex matched to the vertex v in a matching $\mathcal{M}$. That is if $(r_i, c_j) \in \mathcal{M}$, then mate$(r_i) = c_j$ and mate$(c_j) = r_i$. It is well known that perfect matchings in the bipartite graph G correspond to permutations which yield zero-free diagonals, see for example [7]. A matching edge (r_i, c_j) is used to permute the column c_j to the ith position, yielding a zero-free diagonal. This is achieved by defining the permutation matrix M as

$$m_{ij} = \begin{cases} 1 & (r_i, c_j) \in \mathcal{M} \\ 0 & \text{otherwise} \end{cases} ,$$

and then by multiplying A on the right by M, i.e., by forming AM. We will use calligraphic letters for perfect matchings, e.g., $\mathcal{M}$, and the corresponding italic, Roman letters, e.g., M, for the associated permutation matrices.

For a given square $(0,1)$-matrix A, we define the symmetry score as

$$S(A) = \sum_{a_{ij} \neq 0} a_{ij} a_{ji} \ . \tag{1}$$

As seen from the formula each nonzero entry a_{ij} contributes either 0 or 1 to the score. Hence, for a symmetric matrix A, $S(A)$ is equal to the number of its nonzeros. For a given column permutation M, $S(AM)$ measures the symmetry score of the permuted matrix.

2.1 The heuristic

We propose an iterative-improvement-based heuristic. The proposed heuristic works on the bipartite graph representation of a given matrix. It starts with a perfect matching to guarantee a zero-free diagonal, and then iteratively improves the current matching to increase the symmetry while maintaining a perfect matching at all times.

Algorithm 1 Compute the symmetry score

Input: a bipartite graph $G = (R \cup C, E)$ corresponding to an $n \times n$ matrix A
Input: a perfect matching $\mathcal{M}$
Output: score $= S(AM)$

```
1: mark(r) ← 0 for all r ∈ R
2: score ← 0
3: for each (r_i, c_j) ∈ M do
4:    for each c ∈ N(r_i) do
5:       mark(mate(c)) ← j        ▷ mark r_i too
6:    for each r ∈ N(c_j) do
7:       if mark(r) = j then
8:          score ← score + 1     ▷ increase by one for (r_i, c_j),
                                     ▷ also for a symmetric entry (r_k, c_j) ∉ M
                                     ▷ with r_k = mate(c) for a c ∈ N(r_i)
```

Given a perfect matching $\mathcal{M}$, the symmetry score of the permuted matrix AM can be computed as shown in Algorithm 1. The algorithm runs in $O(E)$ time, since the edges incident on a vertex v are visited only when the matching edge incident on v is processed at line 3.

Note that for a matching edge $(r_i, c_j) \in \mathcal{M}$, the test at line 7 holds, and the matching edge (a diagonal entry in the permuted matrix AM) contributes one to the score. Consider two matching edges (r_i, c_j) and (r_k, c_l) such that $(r_i, c_l) \in E$ and $(r_k, c_j) \in E$. These four edges form an $\mathcal{M}$-alternating cycle of length four, and herald two off-diagonal symmetric entries in addition to the two diagonal entries in AM. The score is incremented by 2 for those two off-diagonal symmetric entries in two steps; by 1 when the "for loop" at line 3 is processing (r_i, c_j), and by 1 while the "for loop" at line 3 is processing (r_k, c_l). Let $C4$ be the set of unique alternating cycles of length four, then the score will be

$$S(AM) = n + 2 \times |C4| \ . \tag{2}$$

Note that all perfect matchings will result in a symmetry score of at least n, therefore the number of alternating cycles of length four is the important term.

Let $\mathcal{M}^\star$ and $\mathcal{M}$ be, respectively, an optimal perfect matching maximizing the symmetry score, and another perfect matching on the bipartite graph G. Then the symmetric difference $\mathcal{M}^\star \oplus \mathcal{M} = (\mathcal{M}^\star \setminus \mathcal{M}) \cup (\mathcal{M} \setminus \mathcal{M}^\star)$ contains only isolated vertices and even cycles. This is because a given vertex is incident on exactly one edge of $\mathcal{M}$ and one edge of $\mathcal{M}^\star$; if those edges are the same, then the vertex becomes isolated, otherwise it becomes a part of a cycle (intrinsically

an even cycle as the graph is bipartite). Note that each such cycle is an $\mathcal{M}$-alternating cycle. Therefore, an optimal solution $\mathcal{M}^\star$ can be obtained from any perfect matching $\mathcal{M}$ by finding $\mathcal{M}$-alternating cycles and then reversing the membership of the edges along those cycles. Note that this does not imply an efficient algorithm, as there are combinatorially many alternating cycles with respect to a given matching.

Having observed the role of the alternating cycles in improving a given matching, and having noted Eq. 2, we propose to work on the alternating cycles of length four to improve a given perfect matching. Similar to some other iterative-improvement-based heuristic, such as that in [12], we organize the refinement process in passes. At each pass, we build the set $C4$ of unique alternating cycles of length four using an algorithm much like Algorithm 1. Then, we visit the unique alternating cycles of length four in a random order. Among the cycles those whose vertices are not affected by a previous operation and with a nonnegative effect on symmetry score are reversed. That is, in a four cycle, the matching edges are replaced by the non-matching ones. Algorithm 2 shows the actions taken within a pass.

Algorithm 2 Refine a perfect matching

Input: a bipartite graph $G = (R \cup C, E)$ corresponding to an $n \times n$ matrix A
Input: a perfect matching $\mathcal{M}^{(0)}$
Output: another perfect matching $\mathcal{M}^{(1)}$ where $S(AM^{(1)}) \geq S(AM^{(0)})$

```
M^(1) ← M^(0)
C4 ← {(r_1, c_1, r_2, c_2) : (r_1, c_1) ∈ M^(1) and (r_2, c_2) ∈ M^(1) and
          (r_1, c_2) ∈ E and (r_2, c_1) ∈ E}
while C4 ≠ ∅ do
    pick a cycle C = (r_1, c_1, r_2, c_2) ∈ C4
    if isReversible(C) and gain(C) ≥ 0 then
        M^(1) = M^(1) ⊕ C
    remove the cycle C from C4
```

The test isReversible($\mathcal{C}$) returns true if none of the vertices $\{r_1, c_1, r_2, c_2\}$ has been moved before. In other words, this test returns true if the current matching contains the edges (r_1, c_1) and (r_2, c_2). The gain computations are done by using the main "for loop" of Algorithm 1 for the edges (r_1, c_1) and (r_2, c_2), and then for the edges (r_1, c_2) and (r_2, c_1). The difference between the returned scores gives the gain of reversing the edges in the cycle $\mathcal{C}$.

In the worst case, all gain computations return negative, Algorithm 2 evaluates the gain of all alternating cycles of length four. We first note that a vertex v can be in at most $d(v) - 1$ alternating cycles of length four, because one of its neighbors in all such cycles should be its mate. For each such cycle, the edges incident on v are visited during gain computations. Therefore, the gain computation operations can spend at most $O(d(v)^2)$ time on a vertex v. Hence the worst case total time can be bound as $O\left(\sum_{r \in R} d(r)^2 + \sum_{c \in C} d(c)^2\right)$. In practice a faster worst case running time can be expected. This is because of two reasons. First, there are negative terms adding up to $O(|E|)$ that we omit. Second, the

number of length four alternating cycles containing a vertex v cannot be larger than $d(\text{mate}(v)) - 1$.

We make the following observations: (1) the set of alternating cycles of length four, $C4$, is constructed according to the initial matching; (2) each cycle in $C4$ is considered at most once; (3) due to the nonnegative gain requirement, the algorithm cannot escape from a local minimum. Having observed these deficiencies, we designed two alternatives. The first alternative starts with the same set $C4$ as in Algorithm 2 with more involved data structures and operations. It maintains $C4$ as a priority queue using the gain of a cycle as its key; tentatively modifies the current matching along the best cycle, even along those with a negative effect; at the end, realizes the most profitable prefix of modifications. This approach obtained better results than Algorithm 2, with increases in running time. The second alternative visits the row vertices in a random order, computes the best length four alternating cycle containing that vertex, and modifies the current matching along that cycle if the gain is nonnegative. Algorithm 2 outperformed this alternative in terms of solution quality.

Another point worth mentioning is that computing the block triangular form (btf) of matrix can help reduce the running time of the algorithm. It is well known that the entries outside the diagonal blocks of a btf cannot belong to a perfect matching; see for example [6, Chapter 6] and [14]. Therefore, the corresponding edges in the bipartite graph G cannot belong to an alternating cycle. Those edges can be discarded without any effect on the solution quality of the proposed algorithm to reduce the running time.

2.2 Upper bounds and initial matching

Consider a row r_i and a column c_j where $a_{ij} = 1$. For any matching $\mathcal{M}$ with $(r_i, c_j) \in \mathcal{M}$, the contribution of the matching edge (r_i, c_j) to $S(AM)$ is limited by $\min\{d(r_i), d(c_j)\}$, or, equivalently by the smaller of the number of nonzeros in the corresponding row and column of A. Consider a maximum weight perfect matching $\mathcal{M}^\star_{B1}$, subject to edge weights $w_{ij} = \min\{d(r_i), d(c_j)\}$, in the bipartite graph G. The weight $w(\mathcal{M}^\star_{B1})$ defines an upper bound, referred to as UB1, on the attainable symmetry score. We further obtain an improved upper bound by observing that all neighbors of the vertex r_i may not be matchable to the neighbors of c_j (or vice versa). In fact, two vertices r_i and c_j can contribute at most by the weight of the maximum weight matching $\mathcal{M}^\star_{ij}$, subject to unit edge weights, in the induced subgraph $G_{ij} = (N(c_j) \cup N(r_i), E \cap N(c_j) \times N(r_i))$. Consider a maximum weight perfect matching $\mathcal{M}^\star_{B2}$, subject to edge weights $w_{ij} = w(\mathcal{M}^\star_{ij})$, in G. The weight $w(\mathcal{M}^\star_{B2})$ defines an upper bound, referred to as UB2, on the attainable symmetry score.

Both of the perfect matchings $\mathcal{M}^\star_{B1}$ and $\mathcal{M}^\star_{B2}$ can be used as an initial solution. Note that we have $w(\mathcal{M}^\star_{B2}) \leq w(\mathcal{M}^\star_{B1})$. However, we do not have any relation between the resulting symmetry scores. The CPU cost of finding $\mathcal{M}^\star_{B2}$ can be very high. Therefore, we prefer using $\mathcal{M}^\star_{B1}$ as an initial solution.

3 Experiments

We present results with two sets of square matrices from University of Florida Sparse Matrix Collection [5]. Matrices in the first set originally have symmetric nonzero pattern and full sparse rank, and they satisfy the following assertions regarding the order n and the number of nonzeros nnz: $n \geq 1000$, nnz $\leq 10^6$, nnz $\geq 3 \times n$. There were a total of 420 such matrices (out of 1840) at the time of writing. We ensured that the original matrices have zero-free diagonal. Then, we permuted the rows and columns of those matrices randomly and created unsymmetric matrices. If B is the pattern of an original matrix, then the corresponding matrix A in this set has the pattern of $P(B+I)Q$, where I is the $n \times n$ identity matrix, and P and Q are random permutation matrices. That is, for a matrix A in this set, the optimal symmetry score is equal to the number of nonzeros in A. The matrices in the second set are the 28 public domain matrices used in [7, 15]. These matrices are highly unsymmetric. There are four matrices with an original symmetry score greater than $0.4 \times$ nnz; however, there are zeros in the main diagonal. For these 28 matrices, we do not know the optimal symmetry score. We computed the upper bound UB2 for these matrices.

In the experiments, initial matching $\mathcal{M}^{(0)}$ is chosen to be $\mathcal{M}^{\star}_{B1}$, the perfect matching that defines the upper bound UB1. We use `mc64` (described in [7]) from the mathematical software library HSL [9] with `job=4` to compute $\mathcal{M}^{\star}_{B1}$. The worst case time complexity of `mc64` on an $n \times n$ matrix with τ nonzeros is $O\left(n(\tau+n)\log_2 n\right)$. However, in practice it behaves much faster; see our experiments below and some others in [1] and [7].

We present the performance of the proposed algorithm by normalizing the symmetry score by the upper bound. In what follows, $\mathcal{M}^{(p)}$ is the final perfect matching obtained at the end of the pth refinement pass. Table 1 displays the minimum, the average, and the maximum normalized symmetry scores of $\mathcal{M}^{(0)} = \mathcal{M}^{\star}_{B1}$ and $\mathcal{M}^{(p)}$ for $p \in \{1,3,5,7\}$ on 420 originally symmetric matrices. As seen from the average normalized scores, most of the improvements are obtained within the first few passes. In the remainder, we limit the number of refinement passes to five, i.e., we apply Algorithm 2 five times. As seen from the table, the average symmetry score 0.628 of $\mathcal{M}^{(0)}$ is improved by around 38% in five refinement passes, resulting in an average symmetry score of 0.872 for $\mathcal{M}^{(5)}$.

Table 1. Statistics regarding the symmetry scores normalized by the optimal score on 420 matrices. $S(A)$ is the symmetry score without any column permutation.

	$S(A)$	$S(AM^{(0)})$	$S(AM^{(1)})$	$S(AM^{(3)})$	$S(AM^{(5)})$	$S(AM^{(7)})$
min	0.000	0.307	0.407	0.500	0.556	0.556
avg	0.005	0.628	0.754	0.834	0.872	0.891
max	0.125	1.000	1.000	1.000	1.000	1.000

Figure 1 displays the histogram of the normalized symmetry scores for the initial matching $\mathcal{M}^{(0)}$ and the final matching $\mathcal{M}^{(5)}$. The final matching $\mathcal{M}^{(5)}$ obtained the optimal symmetry score for 89 matrices. The initial matching $\mathcal{M}^{(0)}$ obtained the optimal symmetry score only for 4 matrices. There are only 7 instances below 0.628 (the average for $\mathcal{M}^{(0)}$) among the normalized symmetry scores of $\mathcal{M}^{(5)}$.

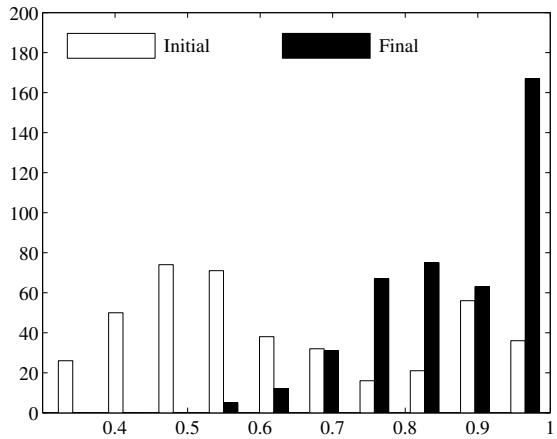

Fig. 1. Histogram of the normalized symmetry scores for the initial matching $\mathcal{M}^{(0)}$ (white bars) and the final matching $\mathcal{M}^{(5)}$ (black bars)

We argue that the initial choice of using the perfect matching $\mathcal{M}^{\star}_{B1}$ which attains the upper bound UB1 is an important part of the proposed algorithm. We show this by experimenting with two other initial perfect matchings. The first one is an arbitrary perfect matching $\mathcal{M}^{\star}_{a}$ on the bipartite graph G (found using `mc64` with `job=1`). The second one is the perfect matching $\mathcal{M}^{\star}_{B2}$ which attains the upper bound UB2. For the originally symmetric matrices, the upper bound UB1 is exact. Hence, UB2 and the associated matching coincide with UB1 and its matching. With an arbitrary initial perfect matching, the proposed algorithm obtained an average normalized symmetry score of 0.656 (minimum 0.073 and maximum 0.995), well below the average 0.872 given in the rightmost column of Table 1. For the originally unsymmetric matrices, the two upper bounds are different, therefore using the three initial matchings makes sense. The average normalized (with respect to UB2) score after five refinement passes starting from these initial matchings are 0.593 (with $\mathcal{M}^{\star}_{a}$), 0.655 (with $\mathcal{M}^{\star}_{B1}$), and 0.699 (with $\mathcal{M}^{\star}_{B2}$). Using $\mathcal{M}^{\star}_{B2}$ as an initial matching resulted in a better average score with the alternative refinement method which uses priority queues (mentioned towards the end of Section 2.1) too. However, on some matrices, it took hours

to compute $\mathcal{M}^{\star}_{B2}$. That is $\mathcal{M}^{\star}_{B1}$ leads to results similar to those of $\mathcal{M}^{\star}_{B2}$, and it is affordable.

For reproducibility of the results on originally unsymmetric matrices, we present the upper bound UB2 and our results in Table 2. The results are obtained by starting from the initial matching $\mathcal{M}^{(0)} = \mathcal{M}^{\star}_{B1}$. On average, the symmetry score of the initial matching is improved by 20% at the end of the fifth refinement pass.

Table 2. Symmetry scores for originally unsymmetric matrices. Matrices in the top block are from [7]; those in the bottom block are from [15]. The upper bounds (UB2) on the symmetry score are not guaranteed to be attainable. The matrices originally have zeros on the main diagonal, therefore we do not display their original symmetry score.

matrix	n	nnz	UB2	$S(AM^{(0)})$	$S(AM^{(5)})$
av41092	41092	1683902	618198	210806	247112
bayer01	57735	277774	160968	98995	101217
gemat11	4929	33185	31851	26054	31851
goodwin	7320	324784	221955	81901	171292
lhr01	1477	18592	10819	8385	8963
lhr02	2954	37206	21640	16706	17960
lhr14c	14270	307858	173826	146361	151670
lhr71c	70304	1528092	862800	728203	752616
mahindas	1258	7682	3974	2357	2364
onetone1	36057	341088	275064	139182	197513
onetone2	36057	227628	189331	106215	135745
orani678	2529	90158	28550	13875	15509
west1505	1505	5445	3547	1742	1747
west2021	2021	7353	4784	2354	2363
bayer03	6747	56196	28124	13809	14597
circuit_3	12127	48137	39956	30526	34543
extr1	2837	11407	8039	3119	3255
fidapm11	22294	623554	623554	258775	562958
g7jac200sc	59310	837936	554972	228093	266580
hydr1	5308	23752	14450	7254	7518
impcol_d	425	1339	840	507	507
jan99jac020sc	6774	38692	20166	7562	7832
mark3jac140	64089	399735	271990	135451	145771
poli_large	15575	33074	18077	15619	15627
radfr1	1048	13299	8734	6184	6782
rdist1	4134	94408	56804	46238	48152
sinc15	11532	568526	322484	203810	302922
Zhao2	33861	166453	158305	84060	144795

We implemented the algorithm in C, compiled with gcc using option `-O3`, and performed the tests on a Pentium IV 2.80 GHz PC with 2GB main memory and

1MB cache. The running time of the proposed algorithm (with five refinement passes) is, on average, 1.6 seconds for those matrices with a normalized initial symmetry score, $S(AM^{\star}_{B1})/\mathrm{UB2}$, less than 0.9. For others, it takes much larger. The largest running time was obtained for the matrix `ASIC_100k` ($n = 99340$, $\mathrm{nnz} = 940696$) having an initial normalized score of 0.912 (final score is 0.998). For this matrix, the running time is almost an hour. The running time of the HSL subroutine `mc64` is, on average, 0.9 seconds. Therefore, we suggest computing the perfect matching $\mathcal{M}^{\star}_{B1}$, checking its symmetry score, and proceeding with 3-to-5 refinement passes if the initial result is not satisfactory.

We have tested the effects of the proposed heuristic on the problem of ordering a matrix to doubly bordered block diagonal form. Given a matrix and an integer K, the aim is to permute the matrix into $(K+1) \times (K+1)$ block form such that there is no nonzero in blocks (k, ℓ) and (ℓ, k) for $1 \leq k < \ell \leq K$, the diagonal blocks (k, k) for $1 \leq k \leq K$ are square and have almost equal order. The cost is the size of the border, i.e., the order of the $(K+1, K+1)$st block. The problem is NP-hard [3]. A common tool used for this task is MeTiS [11]. MeTiS finds the block form using symmetric permutations. The heuristics used in MeTiS works on symmetric matrices. Therefore, the trick of using the pattern of $A + A^T$ (no numerical cancellation) instead of A is applicable here. We tested MeTiS on the matrices given in Table 2 with $A + A^T$ and $AM^{(5)} + (AM^{(5)})^T$ for $K = 4, 8, 16, 32, 64$. Using $AM^{(5)}$ resulted in, on average, 43% smaller border size than using A, with, on average, 36% reduction in the running time. In 110 instances out of 140, using $AM^{(5)}$ resulted in better results than using A. The best reduction, 114 versus 8288, is obtained for `bayer01` with $K = 4$. The worst increase, 6726 versus 3997 was obtained for `sinc15` with $K = 64$; this was an odd case—MeTiS gave almost the same border size for $K = 4, 8, 16, 32, 64$ when using A.

4 Conclusion

We proposed a heuristic to solve the matrix symmetrization with zero-free diagonal problem. The heuristic starts from a judiciously chosen initial solution and iteratively improves it. We presented experiments on two sets of matrices. We know the optimal solution for the matrices in one of the sets. The solutions found by the proposed heuristic are, on average, around 0.87 of the exact solutions for the 420 matrices in this set. We do not know an optimal solution for the matrices in the other set. Therefore, we compared the solution quality with respect to a an upper bound described in the paper. The proposed heuristic achieved solutions, on average, within 0.70 of the upper bounds for the matrices in the second set. Compared to the average achieved for the matrices in the first set, we think that the latter results are lower due to having a loose upper bound.

Acknowledgments

I thank M. Benzi, I. S. Duff, O. Parekh, and S. Pralet for helpful discussions, an anonymous referee for constructive comments which helped improve the presentation of Section 3.

References

1. M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM Journal on Scientific Computing*, 22:1333–1353, 2000.
2. E. Boros, V. Gurvich, and I. Zverovich. Neighborhood hypergraphs of bipartite graphs. Technical Report RRR 12-2006, RUTCOR, Piscataway, New Jersey, 2006.
3. T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, 1992.
4. C. J. Colbourn and B. D. McKay. A correction to Colbourn's paper on the complexity of matrix symmetrizability. *Information Processing Letters*, 11:96–97, 1980.
5. T. Davis. University of Florida sparse matrix collection: `http://www.cise.ufl.edu/research/sparse/matrices`. *NA Digest*, 92/96/97, 1994/1996/1997.
6. I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.
7. 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*, 22:973–996, 2001.
8. B. Hendrickson and T. G. Kolda. Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. *SIAM Journal on Scientific Computing*, 21:2048–2072, 2000.
9. HSL: A collection of Fortran codes for large-scale scientific computation. `http://www.cse.scitech.ac.uk/nag/hsl`, 2004.
10. Y. F. Hu and J. A. Scott. Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers. *Numerical Linear Algebra with Applications*, 12:877–894, 2005.
11. 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, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, 1998.
12. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, February 1970.
13. E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover, Mineola, New York (unabridged reprint of Combinatorial Optimization: Networks and Matroids, originally published by New York: Holt, Rinehart, and Wilson, c1976), 2001.
14. A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16:303–324, 1990.
15. J. K. Reid and J. A. Scott. Reducing the total bandwidth of a sparse unsymmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 28:805–821, 2006.