Journal of Computational and Applied Mathematics 12&13 (1985) 551-561
North-Holland

Algorithms for the reduction of matrix bandwidth and profile

W.F. SMYTH
Unit for Computer Science, McMaster University, Hamilton, Ontario, Canada

Abstract: Since 1969 a standard approach to the reduction of matrix bandwidth and profile has been to grow rooted level structures (RLSs) of the adjacency graph of the matrix, and then to use the 'best' RLS to generate a renumbering of the rows and columns. A generally effective, low-cost method for RLS growth is the Gibbs–Poole–Stockmeyer (GPS) algorithm, especially as modified by George and Liu. Recent work by Arany has suggested alternatives to the GPS algorithm. In this paper, algorithms proposed by Arany and several other new algorithms are described, and results of preliminary computer tests on 'difficult' renumbering problems are presented. In particular, RLS width, bandwidth, profile, and CPU time are compared for four algorithms: Minimum Degree GPS, Minimum Degree Arany, Minimum Width Arany, and Maximum Swing.

Keywords: Algorithm, matrix, sparse, bandwith, profile, graph, numbering, peripheral, diameter.

1. Introduction

Since the publication of the Cuthill–McKee (CM) algorithm in 1969 [3], a standard approach to the reduction of matrix bandwidth and profile has been to:

(1) grow *rooted level structures* (RLSs) of the adjacency graph of the matrix;
(2) select the 'best' RLS based on some criterion (maximum height or minimum width);
(3) number the selected RLS on a level-by-level basis.

The reordering of the matrix rows and columns which is induced by the numbering of the RLS may yield reduced matrix bandwidth and profile [6], or otherwise facilitate the computer implementation of various strategies for the solution of sparse (usually also symmetric and positive definite) sets of simultaneous linear equations [5,6]. The idea of an RLS was introduced in [1] and is described in [5,6].

Taking into account both effectiveness and processing time, the Minimum Degree GPS algorithm (developed by Gibbs–Poole–Stockmeyer [7], modified by George and Liu [4]) is perhaps the most satisfactory RLS-selection algorithm invented so far. But this algorithm has serious deficiencies: first, even though it is designed to locate a starting vertex of a 'best' RLS, its effectiveness often depends heavily on the (arbitrary) choice of its own starting vertex; moreover, even though Minimum Degree GPS was originally designed to improve upon the CM algorithm's heuristic choice of a vertex of minimum degree as the starting vertex of the 'best' RLS, the execution of Minimum Degree GPS itself also depends heavily on exactly the same heuristic strategy.

Recent work by Arany [2] has developed the theory of RLSs, thereby suggesting a number of possible new algorithms for selection of the 'best' RLS. Section 2 of this paper reviews this theoretical background, adding some new ideas and results. In this context, Section 3 then describes nine RLS-selection algorithms (two GPS algorithms, two due to Arany, and five new ones), while Section 4 discusses the results of computer tests on three of these algorithms compared with Modified GPS. Section 5 presents tentative conclusions.

2. Theoretical background

With the usual terminology and notation of graph theory (see, for example, [6,8]), let $G=(V, E)$ represent a finite, connected, undirected graph without loops or multiple edges. Then for any vertex $u \in V$, a *level structure* RLS = RLS(u) *of G rooted at u* is a partitioning of the vertices V into subsets $L_k = L_k(u)$, $k=0, 1, \ldots,$ where

(1) $L_0(u) = \{u\}$,

(2) $L_k(u) = \left\{ y | (x, y) \in E;\ x \in L_{k-1}(u);\ y \notin L_j(u),\ j<k \right\},\quad k>0.$

The subsets L_k are called *levels*. $w_k = w_k(u) = |L_k(w)|$ is called the *width of level k*, and $w = w(u) = \max_k w_k(u)$ is called the *width of the RLS*. The largest integer $h = h(u)$ such that $|L_h(u)|>0$ is called the *height* of the RLS. $h(u)$ is also called the *eccentricity of u*, and the *diameter of G* is defined by $\text{diam}(G) = \max_{u \in V} h(u)$. In the case that $h(u) = \text{diam}(G)$, the vertex u is said to be *peripheral*. If, for given $u \in V$, a vertex v of maximum eccentricity in $L_h(u)$ is located—that is, a vertex v such that $h(v) = \max_{x \in L_h(u)} h(x)$—, and if it is true that $h(v) = h(u)$, then u is said to be *pseudo-peripheral*.

A *numbering $n(V)$ of the vertex set V* is a one-one mapping n: $V \to I$ of V onto the integers $I = \{1, \ldots, |V|\}$. The *bandwidth* $\delta = \delta(n)$ and *profile* $p = p(n)$ of a numbering are defined respectively by

$$\delta = \max_{(x, y) \in E} |n(x) - n(y)|,$$

$$p = \sum_{x \in V} \max_{(x, y) \in E} [0, n(x) - n(y)].$$

It is usually desirable to minimize δ or p, hence to provide some sort of reasonable bound on the quantities $n(x) - n(y)$. A *Reverse Cuthill–McKee* (RCM) *numbering $n_u(V)$ of the RLS rooted at u* [11] is a one–one mapping $n_u: V \to I$ such that $n_u: L_k(u) \to L_k$, $k=0, \ldots, h$, are all one–one mappings, where $I_k = \{\Sigma_{j>k}|L_j(u)|+1, \ldots, \Sigma_{j \geqslant k}|L_j(u)|\}$. This level-by-level numbering thus reduces the number of different possible numberings from $|V|!$ to more manageable proportions, and, as observed in [3], at least ensures that $n_u(x) - n_u(y) < w_k(u) + w_{k+1}(u)$ for all $x \in L_k(u)$, $k = 0, \ldots, h-1$. In practice, except for pathological cases, $n_u(x) - n_u(y)$ is not much greater than $\max[w_k(u), w_{k+1}(u)]$, so that, in order to minimize δ or p, it becomes of interest to grow RLSs of minimum width. One heuristic approach to this problem would be to search for a peripheral vertex u [12], so as to be able to grow an RLS(u) of maximum height [13], in the hope that on the average such RLSs would also have small width. But the efficient location of peripheral vertices turns out to be a difficult problem, so that the GPS method [7] and its variants [4,10], which involve a search for a pseudo-peripheral vertex, have become well established.

Given arbitrary $u, v \in V$, Arany [2] defines the *reversible set* $R(u, v)$ to be the set of vertices which lie on a shortest path from u to v; that is,

$$R(u, v) = \{ x | x \in L_k(u);\ x \in L_{d(u,v)-k}(v) \},$$

where $d(u, v)$ represents the distance from u to v. Further, she defines the *middle set* $M(u, v)$ as a subset of $R(u, v)$ which occurs 'midway' between u and v:

$$M(u, v) = \{ x | x \in R(u, v);\ d(u, x) = \lfloor \tfrac{1}{2} d(u, v) \rfloor \}.$$

She then proves the following:

Theorem 1. *Suppose there exist four (not necessarily distinct) vertices u, v, s, t such that $d(s, t) > d(u, v)$. Then for every vertex $a \in M(u, v)$,*

$$\max[d(a, s), d(a, t)] > d(a, u).$$

Proof. Since $a \in M(u, v)$,

$$d(u, v) = d(u, a) + d(a, v) \geqslant 2d(a, u).$$

But by the triangle inequality,

$$\begin{aligned} d(s, t) &\leqslant d(s, a) + d(a, t) \\ &\leqslant 2 \max[d(a, s), d(a, t)], \end{aligned}$$

from which the result follows immediately.

This theorem suggests that peripheral vertices may often lie 'far' from vertices of the middle set, so that the search for peripheral vertices may be expedited by selecting $a \in M(u, v)$, growing RLS(a), and then inspecting $L_h(a)$. This is the basic strategy of the Arany method and its variants.

Another simple result also gives rise to an algorithm:

Theorem 2. *Suppose $u \in V$ is non-peripheral and $v \in L_h(u)$. Then $V - R(u, v)$ contains at least one of every peripheral vertex pair s, t.*

Proof. Suppose on the contrary that there exist $s, t \in R(u, v)$ such that $d(s, t) = \text{diam}(G)$. Then

$$\begin{aligned} 2d(u, v) &= [d(u, s) + d(s, v)] + [d(u, t) + d(t, v)] \\ &= [d(u, s) + d(u, t)] + [d(s, v) + d(t, v)] \\ &\geqslant 2d(s, t) = 2\ \text{diam}(G), \end{aligned}$$

which contradicts the hypothesis that u is non-peripheral.

Theorem 2 thus provides assurance that a peripheral vertex may be found by searching the *non-reversible set* $V - R(u, v)$. This set may however be quite large, and it therefore becomes of interest to try to identify a subset of it which is at least likely to contain peripheral vertices.

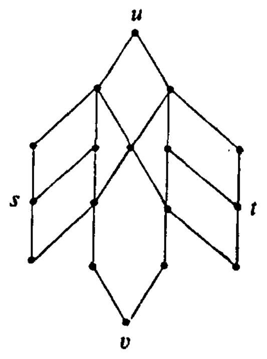
Consider then an arbitrary vertex u and any vertex $x \in L_k(u)$, $k = 0, \ldots, h(\text{u})$. The *set of consequents of x with respect to u* is defined by

$$C_u(x) = \{ y | (x, y) \in E;\ y \in L_{k+1}(u) \}.$$

Note that $C_u(x) = \emptyset$ whenever $k = h(u)$. Next denote by $Q(u)$ the *set of vertices x with no consequents with respect to u*:

$$Q(u) = \left\{ x | C_u(x) = \emptyset \right\}.$$

Note that $L_h(u) \subset Q(u)$ and note further that for every $v \in L_h(u)$, $Q(u) - \{v\} \subset V - R(u, v)$. It is tempting to conjecture that $Q(u)$ necessarily contains a peripheral vertex, but this can be shown by example to be false:



RLS(u)

In fact, as the above example shows, it is not even true that $Q(u) \cup Q(v)$ necessarily contains a peripheral vertex. Nevertheless, it appears that 'often' or 'usually' either $Q(u)$ or $Q(u) \cup Q(v)$ will indeed contain a peripheral vertex, and this notion gives rise to two more heuristic search algorithms (Section 3), each of which involves searching only a subset of $V - R(u, v)$.

Given arbitrary vertices u, $v \in V$, the *swing of a vertex x with respect to u, v* may be defined by

$$s_{u,v}(x) = d(u, x) + d(v, x) - d(u, v).$$

Observe that $s_{u,v}(x) = 0$ if and only if $x \in R(u, v)$, a remark which suggests that the vertices x of maximum swing may well include one or more peripheral vertices. Even this conjecture is not necessarily true, as Arany shows by counterexample [2], but inspection of the vertices of maximum swing nevertheless provides an attractive basis for yet another heuristic algorithm. To facilitate description of this algorithm, the *swinging set*

$$S(u, v) = \left\{ z | s_{u,v}(z) = \max_{x \in V} s_{u,v}(x) > 0 \right\},$$

is introduced. Note that $S(u, v) = \emptyset$ if and only if $V - R(u, v) = \emptyset$.

3. RLS-selection algorithms

In this section, based on the preceding discussion, the algorithms to be tested are described.

Original GPS

(1) Choose arbitrary $u \in V$ and grow RLS(u).

(2) For each $x \in L_h(u)$, grow RLS(x). Choose v such that

$$h(v) = \max_{x \in L_h(u)} h(x).$$

(3) If $h(v) > h(u)$, replace u by v and go to (2).
(4) [u is pseudo-peripheral.] Exit.

Minimum Degree GPS

To avoid growing additional RLSs, step (2) of Original GPS is changed to locate instead a vertex v of minimum degree:

(2) Choose v such that

$$\deg(v) = \min_{x \in L_h(u)} \deg(x),$$

then grow RLS(v).

Observe that the vertex u chosen by this algorithm is no longer necessarily pseudo-peripheral.

Original Arany

(1) Choose arbitrary $u \in V$ and grow RLS(u).
(2) For arbitrary $v \in L_h(u)$, grow RLS(v) up to level $\lfloor \frac{1}{2}h(u) \rfloor w(u)$):.
(3) Choose $a \in M(u, v)$ and grow RLS(a).
(4) For each $x \in L_h(a)$, grow RLS(x). Choose u such that

$$h(u) = \max_{x \in L_h(a)} h(x).$$

(5) [u is 'often' peripheral.] Exit.

Minimum Width Arany

Realizing that minimum width of the RLS is normally a more important criterion than maximum height, Arany [2] proposes changing step (4) of Original Arany to abort RLS growth whenever the width of the current level exceeds or equals the minimum RLS width found so far (initially $w(u)$):

(4) For each $x \in L_h(a)$, grow RLS(x), aborting if at any level $L_k(x)$, $w_k(x) \geqslant w(u)$. If $w(x) < w(u)$, replace by u by x.

Minimum Degree Arany

The minimum degree heuristic applied by George and Liu [4] to Original GPS may also be applied to Original Arany:

(4) Choose v such that

$$\deg(v) = \min_{x \in L_h(a)} \deg(x),$$

then grow RLS(v). If $h(v) > h(u)$, replace u by v.

The four algorithms remaining to be specified in this section all make use of Arany's criterion of minimum RLS width rather than the maximum RLS height criterion. Moreover, they all depend upon the specification of a vertex set Q, different in each case, as Table 1 shows. The algorithms may then be described as follows:

(u, v) Algorithms

(1) Choose arbitrary $u \in V$ and grow RLS(u).
(2) For arbitrary $v \in L_h(u)$, grow RLS(v).

Table 1

Algorithm	Q
Non-Reversible (u, v)	$V - R(u, v)$
No Consequent (u, v)	$Q(u) \cup Q(v) - \{u, v\}$
No Consequent (u)	$Q(u)$
Maximum Swing (u, v)	$S(u, v)$

(3) Compute $Q = Q(u, v)$. If $w(v) < w(u)$, replace u by v.

(4) For each $x \in Q(u, v)$, grow RLS(x), aborting if at any level $L_k(x)$, $w_k(x) \geqslant w(u)$. If $w(x) < w(u)$, replace u by x.

(5) [u is the selected vertex.] Exit.

No Consequent (u)

(1) Choose arbitrary $u \in V$ and grow RLS(u).

(2) Compute $Q = Q(u)$.

(3) For each $x \in Q(u)$, grow RLS(x), aborting if at any level $L_k(x)$, $w_k(x) \geqslant w(u)$. If $w(x) < w(u)$, replace u by x.

(4) [u is the selected vertex.] Exit.

For the 'difficult' test cases described in the next section, some of the above algorithms (specifically, Original GPS, Original Arany, Non-reversible, No consequent (u, v), No consequent (u)) were found to require substantially more computer time than the remaining algorithms, and at the same time yield negligible additional benefit. The algorithms subjected to detailed comparison were therefore the following: Minimum Degree GPS, Minimum Degree Arany, Minimum Width Arany, Maximum Swing. This does not however necessarily imply that the algorithms not tested would not be of value in cases closer to the real world norm.

4. Computer test results

The graphs so far used to test the algorithms described above are randomly generated subject to certain user-specified integer parameters:

$\|V\|$:	the number of vertices;
$r = \|E\|/\|V\|$:	the average degree of the vertices;
d:	the permissible variation of the degree about r.

The triple $(|V|, r, d)$ then describes a graph on $|V|$ vertices, whose degrees are uniformly distributed over the range $[r-d, r+d]$. The test cases described here are for values $|V| =$ 100, 300, 500, 700, 900, 1100 with (r, d) equal to (4, 1) and (5, 2). For *each* test case—for example, for (500, 4, 1)—the user may specify the number of graphs to be generated (the results described here relate to 10 graphs for each test case), and in addition the number of randomly-selected starting vertices u from which to grow the first RLS (the results described here relate to 10 choices of starting vertex for each of the 10 graphs for each test case). Thus the results given here for each test case are actually averages taken over 100 executions of each algorithm tested.

Table 2
Average height and width of "optimal" RLS (over 4 algorithms, 100 executions per algorithm).

Test case	RLS height	RLS width
(900, 4, 1)	8.8	301
(1100, 4, 1)	8.9	367
(900, 5, 2)	7.1	362
(1100, 5, 2)	7.2	439

Typically, for all results but the timing of the algorithm, standard deviations of the sample fall within 5% of the average.

The remark has been made above that these test cases are 'difficult'. This may be demonstrated by reference to statistics for the average, over all four algorithms tested, of the height and width of the 'optimal' RLS (Table 2). By contrast, the statistics for maximum-height RLSs related to two common real world structures (a 32×32 rectangular two-dimensional grid and a $10 \times 10 \times 10$ rectangular three-dimensional grid) are very different (Table 3). As suggested in Table 3, these grids are comparable in terms of the parameters ($|V|$, r) to those of Table 2, but the characteristics of the associated RLSs differ widely. The randomly-generated test cases are characterized by optimal RLSs which are consistently low and wide no matter which algorithm or which starting vertex is employed. In fact, for the Minimum Degree GPS and Minimum Degree Arany algorithms, the starting vertex u was selected as optimal in about 60% of all runs, indicating that at least for these algorithms the choice of starting vertex made very little difference to the height and width of the RLS. It is arguable that, for the kind of test cases generated here, the optimal strategy is simply to select an arbitrary starting vertex and stick with it. On the other hand, for the two grids in Table 3, the selection of the root node of the RLS makes a very great difference to height, width, bandwidth, and profile; indeed, for these grids, even *minimum* height RLSs do not yield RLS widths as large as those of Table 2.

Tables 4–6 show the average RLS width, bandwidth, and profile, respectively, achieved by the four RLS-selection algorithms tested. For the determination of bandwidth and profile, the RCM numbering was used (the King numbering algorithm [9] was also programmed, but for the test cases considered here was found to offer no advantage).

The tabulated results show very little difference among the four algorithms with respect to bandwidth and profile. In terms of RLS width, however, the Minimum Width Arany and Maximum Swing algorithms achieve reductions of up to 10% over the two Minimum Degree algorithms, an effect particularly noticeable for test cases ($*$, 5, 2). These reductions remain consistent over repeated runs for ($*$, 5, 2) and ($*$, 5, 1).

Table 3
Maximum height RLS statistics for grids.

Grid	RLS height	RLS width
32×32 (1024, 3.875, $*$)	62	64
$10 \times 10 \times 10$ (1000, 5.400, $*$)	27	91

Table 4(a)
Test cases (*, 4, 1): average RLS width.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	37	106	174	243	308	377
MINDEG ARANY	37	105	174	240	309	377
MINWD ARANY	33	98	162	229	293	355
MAXSWING	34	97	164	228	292	359

Table 4(b)
Test cases (*, 5, 2): average RLS width.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	42	125	214	305	382	464
MINDEG ARANY	43	123	221	298	366	445
MINWD ARANY	39	115	189	269	352	425
MAXSWING	39	115	194	274	346	423

The algorithms were programmed in FORTRAN 77, then compiled (optimization level 3) and executed on a CYBER 173/815 computer under NOS II. Table 7 displays approximate storage requirements and Table 8 gives average execution times for the test runs whose results are shown in Tables 4–6.

With the exception of Minimum Width Arany, the timing of the algorithms is as expected approximately $O(r|V|)$. The average timing of Minimum Degree Arany is generally slightly greater than that of Minimum Degree GPS, but on the other hand Minimum Degree Arany's

Table 5(a)
Test cases (*, 4, 1): average bandwidth.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	40	113	186	256	329	403
MINDEG ARANY	41	114	187	258	332	403
MINWD ARANY	38	107	180	252	320	393
MAXSWING	38	107	181	251	322	394

Table 5(b)
Test cases (*, 5, 2): average bandwidth.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	48	137	229	318	407	494
MINDEG ARANY	49	138	233	316	408	497
MINWD ARANY	47	134	225	314	403	488
MAXSWING	46	134	227	311	404	493

Table 6(a)
Test cases (*, 4, 1): average K profile.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	2.1	17.4	48.1	92.9	154	230
MINDEG ARANY	2.1	17.6	48.5	93.7	155	231
MINWD ARANY	2.0	16.8	47.0	92.1	151	226
MAXSWING	2.0	16.8	47.4	91.2	151	226

Table 6(b)
Test cases (*, 5, 2): average K profile.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	2.4	20.6	57.5	112	186	275
MINDEG ARANY	2.4	20.8	58.2	112	186	277
MINWD ARANY	2.4	20.3	56.9	112	183	272
MAXSWING	2.3	20.3	57.1	110	184	275

timing has the advantage of a very low standard deviation (the algorithm always grows exactly $3\frac{1}{2}$ RLSs). By contrast, the timing of Minimum Width Arany typically has a standard deviation which is of the same order of magnitude as the average, reflecting the fact that the number of RLSs started and/or completed can vary widely depending on the number of vertices in $L_h(u)$.

Apparently as a result of idiosyncrasies of the CYBER 173 FORTRAN 77 optimizing compiler, the timings given here were found to be very sensitive to seemingly minor coding changes. Timing variations may therefore be expected in other implementations of these algorithms, or in implementations on other computers.

Table 7
Storage requirements for main arrays (words).

Description	Amount
For the graph:	$2(\|E\|+\|V\|) = 2(r+1)\|V\|$
EPTR (|V|)	
DEGREE (|V|)	
ADJ (2|E|)	
For the RLSs:	$5.0\|V\|$, for MAXIMUM SWING;
	$4.5\|V\|$, otherwise
RENUM (|V|, 2)	
LEVEL (|V|, 2)	
WSET (|V|/2, 3)	
Q (|V|/2)	
Total	$\sim (2r+7)\|V\|$

Table 8(a)
Test cases ($*, 4, 1$): average CPU ms.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	11	28	52	67	101	119
MINDEG ARANY	13	36	57	78	100	121
MINWD ARANY	31	104	221	247	371	535
MAXSWING	19	57	89	141	191	217

Table 8(b)
Test cases ($*, 5, 2$): average CPU ms.

Algorithm	$\|V\|$					
	100	300	500	700	900	1100
MINDEG GPS	12	31	62	79	91	116
MINDEG ARANY	15	39	63	87	111	133
MINWD ARANY	35	161	218	580	633	542
MAXSWING	23	62	101	160	205	275

Table 9
Effectiveness/Efficiency of four algorithms.

Algorithm	RLS width	timing
MINDEG GPS	w	t
MINDEG ARANY	$\sim w$	$\sim 1.15t$
MINWD ARANY	$< 0.95w$	$> 3t$
MAXSWING	$< 0.95w$	$\sim 2t$

5. Conclusions

Five new RLS-selection algorithms have been described (Minimum Degree Arany, Non-Reversible, No Consequent (u, v), No Consequent (u), Maximum Swing), of which two (Minimum Degree Arany, Maximum Swing) appear to be competitive with existing algorithms on the kind of test cases considered, as summarized in Table 9.

Further tests are planned with sparse graphs more amenable to RLS-selection methods and more similar to cases which arise in practice.

Acknowledgment

The work described here was supported in part by Grant No. 5-26794 of the Science and Engineering Research Board, McMaster University. The author is grateful to Ms. Christine Nazi, who wrote and tested the FORTRAN programs.

References

[1] Ilona Arany, W.F. Smyth and Lajos Szóda, An improved method for reducing the bandwidth of sparse symmetric matrices, Proc. IFIP Congress, 1971, 1246–1250.

[2] Ilona Arany, A heuristic algorithm for finding peripheral nodes, *SIAM J. Numer. Anal.*, to appear.

[3] E. Cuthill and J. McKee, Reducing the bandwidth of sparse symmetric matrices, Proc. ACM 24th Nat. Conf., 1969, 157-172.

[4] Alan George and Joseph W. Liu, An implementation of a pseudo-peripheral node finder, *ACM Trans. Math. Software* **5** (3) (1979) 284–295.

[5] Alan George, An automatic one-way dissection algorithm for irregular finite element problems, *SIAM J. Numer. Anal.* **17** (6) (1980) 740–751.

[6] Alan George and Joseph W. Liu, *Computer Solution of Large Sparse Positive Definite Systems* (Prentice-Hall, Englewoods Cliff, NJ, 1981).

[7] N.E. Gibbs, W.G. Poole and P.K. Stockmeyer, An algorithm for reducing the bandwidth and profile of a sparse matrix, *SIAM J. Numer. Anal.* **13** (2) (1976) 236–250.

[8] Frank Harary, *Graph Theory* (Addison-Wesley, Reading, MA, 1971).

[9] Ian P. King, An automatic reordering scheme for simultaneous equations derived from network systems, *Internat. J. Numer. Methods Engrg.* **2** (1970) 523–533.

[10] John G. Lewis, Implementation of the Gibbs–Poole–Stockmeyer and Gibbs–King algorithms, *ACM Trans. Math. Software* **8** (2) (1982) 180–189.

[11] Joseph W. Liu and Andrew H. Sherman, Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices, *SIAM J. Numer. Anal.* **13** (1975) 198–213.

[12] W.F. Smyth and W.M.L. Benzi, An algorithm for finding the diameter of a graph, Proc. IFIP Congress, 1974, 500–503.

[13] W.F. Smyth and Ilona Arany, Another algorithm for reducing bandwidth and profile of a sparse matrix, Proc. ACM 31st Nat. Conf., 1976, 987–994.