

## A NEW ALGORITHM FOR FINDING A PSEUDOPERIPHERAL NODE IN A GRAPH\*

ROGER G. GRIMES<sup>†</sup>, DANIEL J. PIERCE<sup>†</sup>, AND HORST D. SIMON<sup>‡</sup>

**Abstract.** A new algorithm for the computation of a pseudoperipheral node of a graph is presented, and the application of this algorithm to reordering algorithms for the solution of sparse linear systems is discussed. Numerical tests on large sparse matrix problems show the efficiency of the new algorithm. When used for some of the reordering algorithms for reducing the profile and bandwidth of a sparse matrix, the results obtained with the pseudoperipheral nodes of the new algorithm are comparable to the results obtained with the pseudoperipheral nodes produced by the SPARSPAK version of the Gibbs-Poole-Stockmeyer algorithm. The advantage of the new algorithm is that it accesses the adjacency structure of the sparse matrix in a regular pattern. Thus this algorithm is much more suitable both for a parallel and for an out-of-core implementation of the ordering phase for sparse matrix problems.

**Key words.** sparse matrices, reordering algorithms, bandwidth reduction, reverse Cuthill-McKee algorithm, Gibbs-Poole-Stockmeyer algorithm, eigenvalues of graphs

AMS(MOS) subject classifications. 65F05, 05C99, 94A20

**1. Introduction.** Algorithms for the numerical solution of sparse linear systems of equations usually start out with reordering the coefficient matrix in order to reduce the fill-in during Gaussian elimination. Several reordering algorithms for sparse matrices require as a first step the determination of a pseudoperipheral node of the graph associated with the adjacency matrix of the problem. For example, the reverse Cuthill-McKee [3] algorithm, the automated nested dissection algorithm, the refined quotient-tree algorithm, and the one-way-dissection algorithm in SPARSPAK [7] all require the determination of a peripheral (or at least pseudoperipheral) node in the associated graph. A widely used algorithm for this purpose is due to Gibbs, Poole, and Stockmeyer [8], and was improved by George and Liu [7], and by Lewis [10]. Other related algorithms have been investigated by Smyth [15]. These heuristic algorithms do not guarantee finding a peripheral node. However, the pseudoperipheral node computed by these algorithms is usually well suited for the purposes of reordering the sparse matrix.

The idea common to all these algorithms is the concept of a rooted level structure of the graph. All these algorithms make direct use of the level structure in performing some type of search heuristic. Here we consider a new and quite different algorithm for determining a pseudoperipheral node. This new algorithm is based on the dominant eigenvector of the adjacency matrix of the graph. Even though our new algorithm does not yield a significant improvement in the performance of the reordering algorithms for sparse linear systems, there are two reasons for writing this detailed investigation of the new algorithm. First, it is indeed remarkable that an algebraic quantity such as an eigenvector can be used in the solution of a discrete graph problem. Eigenvalues of graphs have been studied extensively [4]. Aspvall and Gilbert [2] have used

\* Received by the editors February 17, 1988; accepted for publication (in revised form) May 1, 1989.

<sup>†</sup> Scientific Computing and Analysis Division, Boeing Computer Services, M/S 7L-21, Seattle, Washington 98124.

<sup>‡</sup> Numerical Aerodynamic Simulation (NAS) Systems Division, National Aeronautics and Space Administration (NASA) Ames Research Center, Mail Stop 258-5, Moffett Field, California 94035. (The author is an employee of the Scientific Computing and Analysis (SCA) Division of Boeing Computer Services.)

eigenvectors of the adjacency matrix for the graph coloring problem. Our algorithm, however, appears to be the first application of spectral properties of graphs to sparse matrix reordering problems.

Furthermore, the new algorithm is more suitable for an out-of-core or a parallel implementation. Its key computational requirement is a matrix-vector multiplication, which can be easily implemented, both out-of-core and on a parallel machine. This is in contrast to the algorithms based on a rooted level structure. The generation of a rooted level structure requires repeated access to the adjacency structure of the graph (or the sparse matrix). This involves a large number of random input/output accesses, which make programming an out-of-core version of these algorithms difficult, and their performance inefficient.

The current study of an alternative approach was motivated by the need for an out-of-core reordering algorithm for sparse matrices arising in structural analysis. Since the new reordering capability needed to be implemented in the context of an existing structural analysis package, it was bound by severe core memory limitations. These limitations were imposed rather by the structure of the package, than by actual physical limitations. Details of the implementation are reported by Grimes and Pierce in [9].

In a connected graph with  $n$  vertices and  $m$  edges an exact peripheral node can be found in  $O(nm)$  time by an obvious algorithm. For sparse matrix applications, what is wanted is an almost peripheral node in  $O(m)$  time. In this paper "pseudoperipheral" means "approximately peripheral," i.e., a heuristic approximation to a peripheral node. In some other contexts [7] a pair of nodes are defined to be pseudoperipheral if they both have eccentricity equal to the distance between them. The SPARSPAK algorithm finds such a pair of nodes, usually in  $O(m)$  time in practice, although there are examples that can make it run for at least  $O(m\sqrt{n})$  time, and perhaps more. A different algorithm gets  $O(m\sqrt{n})$  time in the worst case but is not practical [12].

The current report summarizes some of the initial investigations into an alternative algorithm for determining a pseudoperipheral node. Most of the material is based on an earlier report [14]. In §2 we collect some definitions, and in §3 we present the heuristic algorithm. Section 4 presents some bounds on the dominant eigenvector of a graph, which give additional (albeit weak) justification for the heuristic algorithm. Computational issues and numerical results are discussed in §§5 and 6.

**2. Definitions.** Here we consider an undirected, connected graph  $G = (X, E)$ , where  $X$  is the set of nodes, and  $E$  is the set of edges. The elements  $a_{ij}$  of the adjacency matrix  $A$  of  $G$  are defined by

$$(1) \quad a_{ij} = \begin{cases} 1 & \text{if node } i \text{ and } j \text{ are adjacent, or if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

This definition differs from the common definition of an adjacency matrix for a graph (e.g., in [4]) in that we also set  $a_{ii} = 1$ , whereas usually the diagonal elements are set to be zero. If  $G$  is the ordered graph of a symmetric positive definite matrix  $M$ , this definition proves to be more useful for our purposes. In this case the  $a_{ij}$  could be defined directly by

$$(2) \quad a_{ij} = \begin{cases} 1 & \text{if } m_{ij} \neq 0 \\ 0 & \text{if } m_{ij} = 0, \end{cases}$$

i.e., the adjacency matrix reflects directly the zero-nonzero structure of a given matrix and is therefore the appropriate tool for sparse matrix computations.

Since we assumed  $G$  to be connected, the matrix  $A$  is irreducible. By the Perron-Frobenius theorem,  $A$  has a simple, positive eigenvalue  $\lambda$ . The corresponding eigenvector  $v = (v_1, v_2, \dots, v_n)^T$  has all components  $v_i > 0$ , for  $i = 1, \dots, n$ . Here  $n = |X|$ . Therefore  $v$  can be normalized such that  $\sum_{i=1}^n v_i = 1$ . In the following we will only deal with  $\lambda$  and  $v$ , such that

$$(3) \quad Av = \lambda v, \quad \sum_{i=1}^n v_i = 1, \quad v_i > 0 \quad \text{for } i = 1, \dots, n.$$

No confusion with other eigenvalues and vectors is possible. Since  $G$  is connected, every row sum of  $A$  is at least 2, for  $n > 1$ . Hence  $\lambda \geq 2$ .

We will also use the notation  $A_1 > A_2$ , implying that all elements of the matrix  $A_1$  are larger than the corresponding elements of  $A_2$ . Similarly,  $A > \alpha$  for  $\alpha \in R$  means that all elements of  $A$  are larger than the scalar  $\alpha$ . We will use the same notation for the componentwise comparison of vectors.

The *distance* of two nodes  $x_i$  and  $x_j$ , i.e., the length of the shortest path connecting  $x_i$  and  $x_j$ , is denoted by  $d(x_i, x_j)$ , or for short by  $d_{ij}$ . The *eccentricity* of a node  $x_i$  is the quantity

$$(4) \quad e(x_i) = \max_{j=1, \dots, n} d(x_i, x_j).$$

The *diameter* of  $G$  is then defined by

$$(5) \quad \delta(G) = \max_{i=1, \dots, n} e(x_i).$$

A node  $x_i \in X$  is said to be *peripheral* if its eccentricity is equal to the graph's diameter, i.e., if  $\delta(G) = e(x_i)$ .

For a subset  $Y \subseteq X$ , the adjacency set of  $Y$ , denoted by  $Adj(Y)$ , is

$$(6) \quad Adj(Y) = \{x_i \in X - Y \mid \{x_i, x_j\} \in E \text{ for some } x_j \in Y\}.$$

For a node  $x \in X$ , the *level structure rooted at  $x$*  is the partitioning  $L(x)$  of  $X$  satisfying

$$(7) \quad \begin{aligned} L(x) &= \{L_0(x), L_1(x), \dots, L_{e(x)}(x)\}, \\ L_0(x) &= \{x\}, \quad L_1(x) = Adj(L_0(x)), \\ L_i(x) &= Adj(L_{i-1}(x)) - L_{i-2}(x) \quad \text{for } i = 2, 3, \dots, e(x). \end{aligned}$$

**3. A heuristic algorithm for finding peripheral nodes.** We are trying to find a peripheral node of the graph  $G$ , i.e., a node with maximum eccentricity. Such a node seems likely to have the greatest average distance from all other nodes. Consider now the matrix  $A^k$ . Its  $(i, j)$ th entry denotes the number of different paths (or walks) of length  $k$  leading from  $x_i$  to  $x_j$ , where paths are included, which "stay for a while" at a node, because of  $a_{ii} = 1$ . Now let  $u = (1, 1, \dots, 1)^T$ . Then the  $i$ th component of  $A^k u$  is equal to the number of paths of length  $k$ , beginning at an arbitrary node and ending in  $x_i$ . If a node  $x_i$  is "peripheral," this number will be smaller and if a node  $x_i$  lies in the "center" of the graph, this number will be larger. So for  $k \rightarrow \infty$  one should obtain some average number, which indicates how many paths go "on average" through a node. But with some suitable normalization,  $A^k u$  converges to the largest

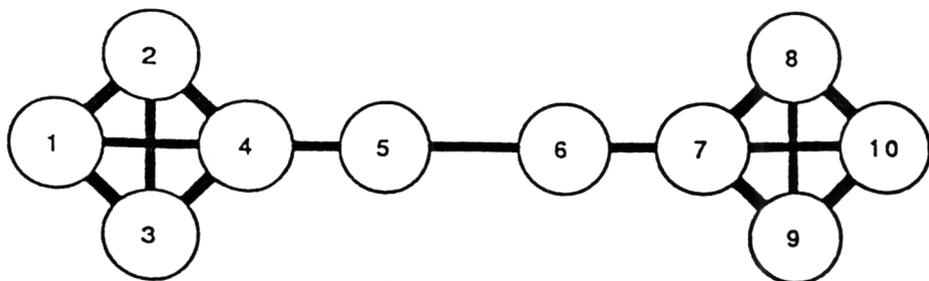


FIG. 1. Counterexample.

eigenvector  $v$  of  $A$ , unless  $u$  were orthogonal to this eigenvector. But this cannot happen; since  $u = (1, \dots, 1)^T$ , we have

$$(8) \quad u^T v = \sum_{i=1}^n v_i = 1 \neq 0.$$

A similar argument has been used in [16] to determine the center of a graph for an application in geography. We use the same method for a different, but closely related application. These arguments suggest the following very simple algorithm for finding pseudoperipheral nodes of a graph:

- (1) Find  $v$ , the dominant eigenvector of the adjacency matrix  $A$ .
- (2) The node corresponding to the smallest component in  $v$  is a pseudoperipheral node.

This algorithm will only determine pseudoperipheral nodes and not necessarily a peripheral node. As a counterexample, consider the graph in Fig. 1. Clearly all the nodes in the two cliques at the end ( $x_1, x_2, x_3$  and  $x_8, x_9, x_{10}$ ) are peripheral. The vector  $v$ , however, is given by  $v \approx (0.1073, 0.1073, 0.1073, 0.1211, 0.0569, 0.0569, 0.1211, 0.1073, 0.1073, 0.1073)^T$ . The smallest components of  $v$  are just corresponding to the "interior" nodes  $x_5$  and  $x_6$ . It is interesting to note that the graph in Fig. 1 also serves as the standard counterexample for a perfect elimination graph for which the minimum degree algorithm does not find a perfect elimination order (see [5, p. 130]).

The proposed method will also fail if the graph is regular, that is, all vertices have the same degree. In this case the components of the dominant eigenvector are all equal. Regular graphs are not common in practice, but it is easy to construct regular graphs in which eccentricities vary widely.

**4. Bounds for the dominant eigenvector.** Although the example above shows that the heuristic algorithm from §3 will not always produce peripheral nodes, we are able to obtain lower bounds on the components of the dominant eigenvector. These bounds indicate that there is a certain inverse relationship between the components of the eigenvector and the eccentricity of the corresponding node.

**PROPOSITION 1.** For  $n > 1$  the components  $v_i$  of the dominant eigenvector  $v$  satisfy

$$(9) \quad v_i \geq \frac{1}{e(x_i)(\lambda - 1)^{e(x_i)} + 1}$$

for  $i = 1, 2, \dots, n$ .

*Proof.* Let  $Av = \lambda v$  and let  $L(x_i) = \{L_0(x_i), L_1(x_i), \dots, L_{e(x_i)}\}$  be the level structure rooted at  $x_i$ . Furthermore, for brevity let

$$(10) \quad \sum_{Adj(x_i)} v_j$$

denote the sum of all  $v_j$  over all indices  $j$ , such that  $x_j \in Adj(x_i)$ , and similarly  $\sum_{L_k(x_i)} v_j$ , etc.

Now  $Av = \lambda v$  implies that (for  $n > 1$ )

$$(11) \quad v_i = \frac{1}{\lambda - 1} \sum_{Adj(x_i)} v_j = \frac{1}{\lambda - 1} \sum_{L_1(x_i)} v_j, \quad i = 1, \dots, n.$$

Substituting (11) into itself and taking into account that  $x_i \in Adj(x_j)$  for  $x_j \in L_1(x_i)$ , we obtain

$$(12) \quad v_i \geq \frac{1}{(\lambda - 1)^2} \sum_{L_2(x_i)} v_j, \quad i = 1, \dots, n.$$

This process can be repeated  $e(x_i)$  times so that we obtain

$$(13) \quad v_i \geq \frac{1}{(\lambda - 1)^k} \sum_{L_k(x_i)} v_j \quad \text{for } i = 1, 2, \dots, n \quad \text{and } k = 1, 2, \dots, e(x_i).$$

Summing up the  $e(x_i)$  inequalities (13), it follows that

$$(14) \quad e(x_i)v_i \geq \sum_{k=1}^{e(x_i)} \frac{1}{(\lambda - 1)^k} \sum_{L_k(x_i)} v_j \geq \frac{1}{(\lambda - 1)^{e(x_i)}} \sum_{j=1; j \neq i}^n v_j = \frac{1 - v_i}{(\lambda - 1)^{e(x_i)}}.$$

Here  $\lambda \geq 2$  was used, which is correct for  $n > 1$ , as mentioned after formula (3). Therefore

$$(15) \quad v_i \geq \frac{1}{e(x_i)(\lambda - 1)^{e(x_i)} + 1} \quad \text{for } i = 1, 2, \dots, n.$$

This is also correct for  $n = 1$ .  $\square$

**PROPOSITION 2.** Let  $\delta$  be the diameter of the graph. Then

$$(16) \quad \lambda \geq 1 + \sqrt[\delta]{\frac{n-1}{\delta}}.$$

*Proof.* From (9) it follows that

$$(17) \quad v_i \geq \frac{1}{\delta(\lambda - 1)^\delta + 1} \quad \text{for } i = 1, 2, \dots, n.$$

Summing up for  $i = 1, 2, \dots, n$  and rearranging yields the result.  $\square$

For the proof of Proposition 3, the following lemma is needed.

LEMMA 4.1. Let  $a_{ij}^{(k)}$  be the  $(i, j)$ th entry of the matrix  $A^k$ ,  $k = 1, 2, 3, \dots$  and let  $n > 1$ . Then it holds that

$$(18) \quad a_{ij}^{(k)} \geq 1 \quad \text{for all } i, j \text{ with } d(x_i, x_j) = k$$

$$(19) \quad a_{ij}^{(k)} \geq k \quad \text{for all } i, j \text{ with } d(x_i, x_j) < k.$$

*Proof.* Let  $d(x_i, x_j) = p \leq k$ . Now  $a_{ij}^{(k)}$  counts the number of paths of length at most  $k$  steps from  $x_i$  to  $x_j$ . If we follow the shortest path and make exactly  $k$  steps, of which  $p$  go forward and  $k - p$  stay at the same node (go around self-loops), there are  $\binom{k}{p}$  possibilities for the choice of  $p$  forward steps. But  $\binom{k}{p} \geq k$  if  $1 \leq p < k$  giving (19) if  $i \neq j$ ; the case  $i = j$  is treated similarly, and  $\binom{k}{p} = 1$  if  $p = k$  giving (18).  $\square$

PROPOSITION 3.

$$(20) \quad v_i \geq \frac{1}{\lambda^{e(x_i)} + 1} \quad \text{for } i = 1, \dots, n.$$

*Proof.* Let  $a_{ij}^{(k)}$  be the  $(i, j)$ th entry of  $A^k$  as before, and let  $D$  be the distance matrix of the graph, i.e.,  $D = (d_{ij})$ , where  $d_{ij} = d(x_i, x_j)$ . Then the following statements about  $a_{ij}^{(k)}$  and  $d_{ij}$  can be made for  $k = 1, 2, \dots$  using (18) and (19):

$$(21) \quad \left. \begin{array}{l} a_{ij}^{(k)} \geq k \\ d_{ij} \geq 1 \end{array} \right\} \quad \begin{array}{l} \text{for all } i, j \text{ with } d_{ij} < k \text{ except} \\ \text{for the diagonal elements where } d_{ii} = 0 \end{array}$$

$$(22) \quad \left. \begin{array}{l} a_{ij}^{(k)} \geq 1 \\ d_{ij} = k \end{array} \right\} \quad \text{for all } i, j \text{ with } d_{ij} = k$$

$$(23) \quad \left. \begin{array}{l} a_{ij}^{(k)} = 0 \\ d_{ij} \geq k + 1 \end{array} \right\} \quad \text{for all } i, j \text{ with } d_{ij} \geq k + 1.$$

Taking (21) - (23) together in matrix form it holds that every element of the matrix  $I + A^k + D$  is greater or equal to  $k + 1$ , where  $I$  is the  $n \times n$  identity matrix. Let  $J$  be the  $n \times n$  matrix with all entries equal to one. Then this fact can be written as

$$(24) \quad I + A^k + D \geq (k + 1)J.$$

Therefore

$$(25) \quad v + A^k v + Dv \geq (k + 1)Jv = (k + 1)u,$$

where  $u = (1, 1, \dots, 1)^T$ . The  $i$ th component of  $Dv$  can be bounded as follows:

$$(26) \quad (Dv)_i = \sum_{l=1}^n d_{il}v_l \leq e(x_i) \sum_{l=1}^n v_l = e(x_i).$$

TABLE 1  
Eigenvector bounds for example graph.

i	$v_i$	$e(x_i)$	Lower bound from Prop. 1	Lower bound from Prop. 3
1	0.0364	4	0.00235	0.00317
2	0.1111	3	0.00998	0.01322
3	0.0756	4	0.00235	0.00317
4	0.0414	4	0.00235	0.00317
5	0.1144	3	0.00998	0.01322
6	0.1315	3	0.00998	0.01322
7	0.1330	3	0.00998	0.01322
8	0.1232	3	0.00998	0.01322
9	0.0871	3	0.00998	0.01322
10	0.1480	3	0.00998	0.01322

Using (25) and  $A^k v = \lambda^k v$ , one obtains for the components in (24)

$$(27) \quad v_i + \lambda^k v_i + e(x_i) \geq k + 1 \quad \text{for } i = 1, 2, \dots, n \quad \text{and } k = 1, 2, 3, \dots$$

If  $k$  is chosen to be  $e(x_i)$ , then (20) follows.  $\square$

Note that the choice  $k = e(x_i)$  in (27) makes the bounds the best possible, since  $k < e(x_i)$  yields trivial bounds and  $k > e(x_i)$  yields in general some worse bounds because of the rapidly growing denominator.

PROPOSITION 4.

$$(28) \quad \lambda \geq \sqrt[\delta]{n-1}.$$

*Proof.* Set  $k = \delta$  in (27). Then

$$(29) \quad v_i \geq \frac{\delta + 1 - e(x_i)}{\lambda^\delta + 1} \geq \frac{1}{\lambda^\delta + 1} \quad \text{for } i = 1, \dots, n.$$

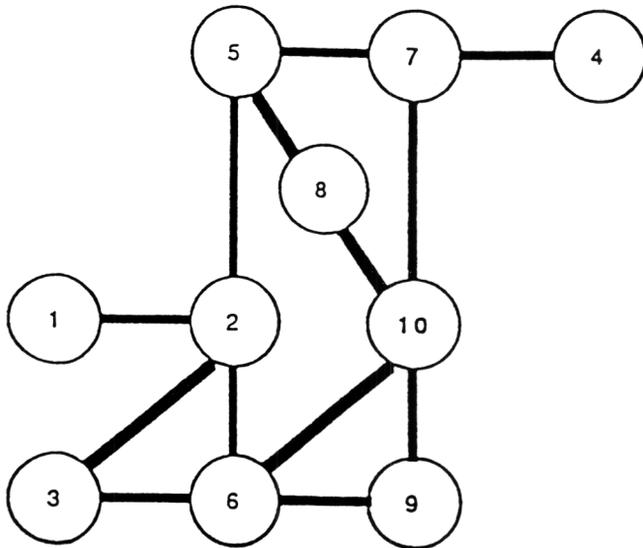
Summing over  $i$  and rearranging yields (28).  $\square$

All the bounds in the propositions above are rather weak. But this is to be expected, since they were proven for general graphs without any further assumptions. The bounds of Proposition 1 are better for some smaller graphs, whereas the bounds of Proposition 3 are better for larger graphs (for larger  $e(x_i)$ ). It should also be noted that the bounds of Proposition 3 are almost sharp, if the graph is a clique. Therefore, there is not much hope to improve these bounds in all generality. However, all bounds on the components of the eigenvector show that there is an inverse relationship between eccentricity  $e(x_i)$  and the corresponding  $v_i$ .

The weakness of the bounds can be seen in the following example. The graph is taken from [7], where it is also used to illustrate several algorithms and concepts. For this graph the figures in Table 1 were obtained. Clearly the bounds from Propositions 1 and 3 are an order of magnitude smaller than the corresponding components of the eigenvector.

Here  $\delta = 4$  and  $\lambda \approx 4.21$ . The bounds of Propositions 2 and 4 yield

$$(30) \quad \begin{aligned} \lambda &\geq 1 + \sqrt[4]{\frac{9}{4}} \approx 2.225 \\ \lambda &\geq \sqrt[4]{9} \approx 1.732. \end{aligned}$$

FIG. 2. *Example.*

### 5. Application to bandwidth and profile reduction for sparse matrices.

In §2 we proposed a new algorithm for computing a pseudoperipheral node in a graph. Since this algorithm is only a heuristic, and since the term “pseudoperipheral” is only defined in the context of this heuristic algorithm, there is only one way to assess the efficacy of such an algorithm: to compare it to other algorithms in an application to a practical problem. The application of the new algorithm that we are most interested in is sparse matrix computations.

The solution of sparse linear equations of the form

$$(31) \quad Mx = b$$

by direct methods has been an area of intensive research during the past 15 years. For symmetric positive definite matrices most of the effort has been directed toward a combination of Gaussian elimination with some reordering of the equations and unknowns in (31). The goal is to obtain a permutation such that the solution of the permuted system incurs less fill-in than the solution of the original system. The actual numerical entries of  $M$  are irrelevant for this reordering phase, because if  $M$  is positive definite, then so is the permuted system, and a Cholesky factorization can always be computed. Thus the reordering can be based on the structural information for the matrix, i.e., the graph of the adjacency matrix alone. For a detailed discussion of the topic see the book by George and Liu [7].

Several reordering heuristics discussed in [7] ideally require the computation of a peripheral node. The practical Fortran implementation of these algorithms, however, relies on the Gibbs-Poole-Stockmeyer (GPS) algorithm, which computes a pseudoperipheral node. For most practical applications, this node is a good starting node for the reordering algorithms in SPARSPAK. In order to test the new pseudoperipheral

node finder, we replaced the subroutine FNROOT in SPARSPAK by a new subroutine which computed the dominant eigenvector of the adjacency matrix using the power method. Then the node corresponding to the component with the smallest entry in the eigenvector was used as a pseudoperipheral node. We could have used a more powerful algorithm such as the Lanczos algorithm, as is argued in [13]. But for our purposes here a few steps of the power method were sufficient, as our results in the next section will show.

The application of the power method is straightforward. The only question that remains to be discussed is a suitable stopping criterion. We are interested only in the location of the smallest entry of the dominant vector, possibly only in the location of a small, but not necessarily the smallest entry. The numerical results indicate that four steps of the power method were sufficient to obtain a pseudoperipheral node which was efficient for our sparse matrix applications. This number of iteration steps was also chosen in the implementation discussed in [9].

The new algorithm can be applied in the context of profile and bandwidth reduction algorithms for the reordering of sparse matrices. Recent research results [1], [11] indicate that sparse Gaussian elimination based on profile and bandwidth is no longer competitive with general sparse and multifrontal methods. However, band and envelope methods are widely used in applications in structural engineering, and are used in many software packages for engineers. For these applications the new algorithm is an alternative, since it does not require a general redesign of the package based on a new data structure for the sparse matrices. As in the case of [9] only one extra subroutine is required. Another potential application of the new algorithm is in the context of general sparse schemes, which sometimes require pseudoperipheral nodes as well, e.g., the automated nested dissection algorithm [7].

**6. Numerical results.** In Table 2 we summarize some characteristics of the sparse matrix test problems, which we used to evaluate the new heuristic algorithm. All test problems are available in the Boeing-Harwell sparse matrix collection and are described in [6]. Table 2 lists the problems, the number of equations (nodes), the number of nonzeros in the matrix (edges in the graph), and both profile and bandwidth of the unordered matrix. The first two examples are electric power networks. These are planar graphs, which correspond probably most closely to the model we had in mind, when developing the new algorithm. Problems 3 - 7 are finite-element models of three-dimensional structures. They are probably distinguished by the existence of many cliques. These examples are typical for the type of matrices encountered in structural engineering. The last three examples are finite-difference approximations to problems defined in very regular two-dimensional domains.

The matrices in Table 2 were first reordered with the reverse Cuthill-McKee (RCM) algorithm as implemented in [7], and then reordered using the new eigenvector algorithm. In order to evaluate the change in efficiency in the reordering, we computed the smallest component of the iteration vector in the power method for each of the first 25 iterations of the power method, and then at each iteration step the resulting RCM ordering. In Table 3 we list the results of this numerical experiment. We give the best result obtained with the eigenvector method, and the number of iterations required to obtain this result. In most (but not all) cases more iterations of the power method did not change the results in Table 3.

Table 3 demonstrates that the node corresponding to the smallest component of the iteration vector in the power method is a suitable alternative as a pseudoperipheral node. The RCM method yields about the same reduction in profile and bandwidth

TABLE 2  
Test matrices.

	Title	Equations	Nonzeros	Profile	Bandwidth
1	Western US Power Network	1,723	6,511	472,515	1,663
2	Entire US Power Network	5,300	21,842	6,122,200	5,189
3	TV Studio	1,074	12,960	240,161	590
4	Fluid Flow - Stiffness Matrix	2,003	83,883	434,798	1,250
5	Geodesic Dome	2,132	14,872	188,488	1,805
6	Cannes Matrix	1,072	12,444	277,248	1,048
7	Connection Table	2,680	25,026	587,863	2,499
8	9-Point Operator on 40 × 40 Grid	1,600	13,924	63,960	41
9	9-Point Operator on 80 × 80 Grid	6,400	56,644	511,920	81
10	George's L-shaped Problem	3,466	23,896	363,844	3,434

TABLE 3  
Comparison with SPARSPAK RCM for envelope reduction.

	SPARSPAK RCM		Best power with RCM		Iter.	Time RCM	Time power
	Profile	Bandw.	Profile	Bandw.			
1	79,260	133	74,251	130	4	0.18	0.44
2	667,245	285	626,863	274	9	0.66	3.10
3	282,999	704	246,776	640	1	0.22	0.20
4	502,907	546	522,640	411	2	1.08	1.86
5	171,437	105	172,712	101	8	0.30	1.94
6	56,438	178	75,409	248	1	0.24	0.14
7	102,983	69	105,058	69	15	0.50	5.96
8	81,497	79	81,497	79	1	0.55	0.46
9	666,997	159	666,997	159	1	2.25	1.86
10	158,546	62	158,546	62	1	0.48	0.32

TABLE 4  
Comparison with SPARSPAK RCM as a pseudoperipheral node finder.

	Diameter	Periph. Nodes	SPARSPAK RCM		Best power with RCM	
			Node	Eccen.	Node	Eccen.
1	38	5	418	38	224	38
2	50	6	1436	50	92	48
3	9	4	1063	9	1	9
4	12	90	659	12	34	11
5	35	20	633	35	192	34
6	13	24	203	13	46	12
7	76	7	243	76	240	73
8	40	156	40	40	1	40
9	80	316	80	80	1	80
10	91	2	16	91	16	91

with either the SPARSPAK pseudoperipheral node as starting node or with the node delivered by our algorithm. The execution times (in seconds) for these numerical tests were obtained on a Sun 3/260 with a floating-point accelerator. The new method does require somewhat higher execution times; however, this additional overhead is insignificant when compared to actual numerical factorization times for these types of matrices (cf. [1], [11]).

Table 3 demonstrates that the eigenvector method is suitable for the intended sparse matrix application. The effectiveness of the eigenvector method for finding pseudoperipheral nodes is demonstrated in Table 4. For the graphs corresponding to the matrices in Table 2 we list the diameter, the number of peripheral nodes, and the nodes found by SPARSPAK RCM and the eigenvector method together with their eccentricity. The numbering of the nodes refers to the original ordering of the matrices as given in the sparse matrix test collection [6].

In Table 5 we summarize the reduction in profile obtained by using the SPARSPAK pseudoperipheral node, the node corresponding to the smallest component of the power method iteration vector after 4 steps, and the node corresponding to the smallest component of the dominant eigenvector. In addition, we list the envelope reduction obtained from the GPS algorithm and from the Gibbs-King (GK) algorithm as implemented by Lewis in [10]. Generally the Gibbs-King is known to obtain the best reduction in envelope size, usually at the cost of increasing the bandwidth.

TABLE 5

*Profile reduction using SPARSPAK RCM, GK, GPS, four iterations of the power method (POW4), and dominant eigenvector (EIG).*

	RCM	GK	GPS	POW4	EIG
1	0.17	0.14	0.15	0.16	0.18
2	0.11	0.09	0.09	0.16	0.13
3	1.18	0.80	0.87	1.27	1.27
4	1.16	0.97	1.07	1.20	1.30
5	0.91	0.89	0.92	0.94	0.92
6	0.20	0.18	0.27	0.36	0.20
7	0.18	0.16	0.17	0.25	0.18
8	1.27	1.00	1.00	1.27	1.27
9	1.30	1.00	1.00	1.30	1.30
10	0.43	0.43	0.43	0.43	0.43

Four iterations of the power method were used in [9], and Table 5 demonstrates that this is a reasonable choice. The node thus selected delivers a profile reduction comparable to the GPS node, at a cost which is slightly higher. Note that Table 5 lists the reduction in profile obtained, normalized so that the profile of the original matrix as given in [6] is one. Apparently Problems 3 - 5 are given in a reduced profile form already, since we are not able to obtain any improvements. All algorithms fail in the same way on the regular grid problems. If there is no reduction in the envelope size, GPS and GK are returning the original ordering.

These results demonstrate that the new pseudoperipheral node finder based on the dominant eigenvector, or the computationally more efficient algorithm based on a few steps of the power method, is an alternative to the GPS, GK, and SPARSPAK RCM algorithms. The figures in Table 5 indicate the better performance of GPS and GK on this test set. These are results with the unmodified versions of these

algorithms. We did not merge our eigenvector algorithm with GPS and GK in the same way as we combined it with SPARSPAK RCM. These tests were not carried out, since we expect to see very similar results.

Because of its simplicity the above algorithm has been implemented as an out-of-core alternative in a software package for solving linear systems arising in structural analysis [9]. The advantages of the new algorithm for a parallel implementation are clear, but have not yet been pursued by the authors. More fundamentally, we were able to exploit the algebraic properties of the adjacency matrix of a graph for computational purposes. That it is possible at all to utilize this information in order to uncover structural properties of the graph and the corresponding sparse matrix came as a surprise to us. We believe that spectral properties of the adjacency matrix have more potential use in sparse matrix computations beyond the ideas discussed here.

**Acknowledgment.** We would like to thank John Lewis for making several valuable suggestions for improving the manuscript, as well as for providing the numerical test results with the GPS and GK algorithms.

#### REFERENCES

- [1] C. ASHCRAFT, R. GRIMES, J. LEWIS, B. PEYTON, AND H. SIMON, *Recent progress in sparse matrix methods for large linear systems*, Internat. J. Supercomput. Appl., 1 (1987), pp. 10 - 30.
- [2] B. ASPVALL AND J. GILBERT, *Graph coloring using eigenvalue decomposition*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 526 - 538.
- [3] E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in Proc. 24th National Conference of the Association of Computing Machinery, ACM Publications, 1969, p. P69.
- [4] D. CVETKOVIC, M. DOOB, AND H. SACHS, *Spectra of Graphs*, Academic Press, New York, 1980.
- [5] I. DUFF, A. ERISMAN, AND J. REID, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.
- [6] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM TOMS, 15 (1989), pp. 1 - 14.
- [7] A. GEORGE AND J. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [8] N. GIBBS, W. POOLE, AND P. STOCKMEYER, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM J. Numer. Anal., 13 (1976), pp. 236 - 249.
- [9] R. GRIMES AND D. PIERCE, *The implementation of three resequencing algorithms for MSC/NASTRAN*, Tech. Report ETA-TR-65, Boeing Computer Services, Seattle, WA, 1987.
- [10] J. LEWIS, *Implementations of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms*, ACM Trans. Math. Software, 8 (1982), pp. 180 - 189.
- [11] J. LEWIS AND H. SIMON, *The impact of hardware gather/scatter on sparse Gaussian elimination*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 304 - 311.
- [12] J. K. PACHL, *Finding pseudoperipheral nodes in graphs*, J. Comput. System Sci., 29 (1984), pp. 48 - 53.
- [13] B. PARLETT, H. SIMON, AND L. STRINGER, *Estimating the largest eigenvalue with the Lanczos algorithm*, Math. Comp., 38 (1982), pp. 153 - 165.
- [14] H. D. SIMON, *Bounds for the dominant eigenvector of a graph*, Tech. Report, Dept. of Appl. Math., State University of New York, Stony Brook, NY, 1982.
- [15] W. F. SMYTH, *Algorithms for the reduction of matrix bandwidth and profile*, J. Comp. Appl. Math., 12/13 (1985), pp. 551 - 561.
- [16] P. D. STRAFFIN, *Linear algebra in geography: Eigenvector networks*, Math. Mag., 53 (1980), pp. 269 - 276.