

COMPRESSED GRAPHS AND THE MINIMUM DEGREE ALGORITHM*

CLEVE ASHCRAFT†

Abstract. The minimum degree algorithm is well known and widely used to find a permutation matrix P such that a sparse symmetric positive definite matrix A can be factored as $PAP^T = LL^T$ to yield a lower triangular matrix L with relatively few nonzero entries. In this paper we show that by compressing the graph of A , when possible, the run time of the minimum degree algorithm can be significantly reduced. Some empirical results for a collection of test matrices are presented.

Key words. minimum degree algorithm, ordering algorithms, compressed graphs

AMS subject classifications. 65F05, 65F50, 68R10

1. Introduction. When a sparse symmetric matrix A is factored into LL^T , it is likely that *fill-in* occurs, i.e., the number of nonzero entries in L is far greater than the number in the lower triangle of A . It is expedient to search for a permutation matrix P such that when PAP^T is factored into $\hat{L}\hat{L}^T$, the number of nonzero entries in $\hat{L}$ is small. The minimum degree algorithm is well known and often used to find a low-fill permutation of A .

We give a brief description of the minimum degree algorithm in §2. The reader should consult [3] for a broad survey of the algorithm's history, evolution, and enhancements. The purpose of this short note is simple: We have noticed that a very minor modification to the algorithm, really a preprocessing step, can reduce the ordering time for certain classes of matrices, including those from structural analysis applications. Our empirical studies are presented in §3. We present a summary and possible extensions in §4.

2. The minimum degree algorithm. Consider the graph $G(V, E)$ of a symmetric matrix A . $V = \{1, \dots, |V|\}$ is the set of *vertices* that corresponds to the rows and columns of the matrix. $E \subseteq V \times V$ is the set of *edges*, where $(i, j) \in E$ if and only if $a_{i,j} \neq 0$. The adjacency set of vertex u is

$$\text{adj}(u) = \{v \mid (u, v) \in E\},$$

the set of all vertices that are connected to u by an edge. (We consider each vertex to be adjacent to itself.) The adjacency set of a set of vertices is $\text{adj}(U) \equiv \bigcup_{u \in U} \text{adj}(u)$. The *degree* of vertex u is the size of its adjacency set, $|\text{adj}(u)|$.

The minimum degree algorithm chooses to eliminate a vertex that has minimum degree of all remaining vertices. Once a vertex u is eliminated, edges are added to E in such a way that simulates the rank-one update from the u 's column to the remainder of the matrix that takes place during the numerical factorization. One hopes that by choosing a vertex of minimum degree (i.e., a column with the fewest nonzero elements) the fill-in will be minimized.

The minimum degree algorithm is given in Fig. 1. There are three major components to a minimum degree code.

- A list of vertices and their degrees must be maintained that supports an efficient search for a vertex of minimum degree as well as updating the degree of any vertex.
- A compact and easily modified representation of the edges.
- The degree of a vertex must be re-computed after a vertex adjacent to it has been eliminated. Since the edge set is usually kept in an implicit form, computing the degrees of the vertices can be the most time-consuming operation.

*Received by the editors August 12, 1993; accepted for publication (in revised form) September 6, 1994.

†Boeing Computer Services, P. O. Box 24346, Mail Stop 7L-22, Seattle, Washington 98124.

```

while  $V \neq \emptyset$ 
  choose  $u \in V$  to eliminate such that  $|\text{adj}(u)|$  is a minimum.
  number  $u$  next in the ordering and remove  $u$  from  $V$ .
  update the edges.
     $E := E \cup (\text{adj}(u) \times \text{adj}(u)) \setminus ((\text{adj}(u) \times \{u\}) \cup (\{u\} \times \text{adj}(u)))$ 
  for  $w \in \text{adj}(u)$ ,  $w \neq u$ 
    update the degree  $|\text{adj}(w)|$ 

```

FIG. 1. Minimum degree defined on $G(V, E)$.

There are three enhancements to the basic minimum degree algorithm that are in common use. Each depends on the relationship between the adjacency sets of two different vertices. (See [3] for more details.)

- Vertices u and v are *independent* if $\text{adj}(u) \cap \text{adj}(v) = \emptyset$. A set $U \subseteq V$ is an *independent set* if its members are pairwise independent. A set U is a *maximal independent set* if U is independent and $U \cup \{w\}$ is not independent for all vertices $w \notin U$.

If u and v are independent and $|\text{adj}(u)| = |\text{adj}(v)|$ is a minimum, then both u and v can be eliminated in the same major step of the algorithm. At any time in the elimination, there may be more than one vertex with minimum degree, and an independent set with one or more vertices must be chosen somehow.

- When $\text{adj}(u) \subset \text{adj}(v)$, v is said to be *outmatched* by u , for v will never have degree less than that of u . The degree update for an outmatched vertex need not be performed until a vertex that outmatches it is eliminated.
- Vertices u and v are *indistinguishable* if $\text{adj}(u) = \text{adj}(v)$. Let us use boldface $\mathbf{u}$ to denote an *indistinguishable set* of vertices whose members are pairwise indistinguishable. A set $\mathbf{u}$ is a *maximal indistinguishable set* if $\mathbf{u} \cup \{w\}$ is not an indistinguishable set for all $w \notin \mathbf{u}$.

There are three major benefits when indistinguishable sets are used in the minimum degree algorithm.

- All vertices in an indistinguishable set $\mathbf{u}$ can be numbered consecutively when a vertex in $\mathbf{u}$ is eliminated.
- The number of degree updates is reduced, since the degree of only one representative vertex in $\mathbf{u}$ needs to be computed.
- The number of active data structures is reduced. This can mean a decrease in the amount of working storage as well as a reduction in the traversals and creation of lists of vertices.

No ordering code known to the author detects *all* outmatching or indistinguishability. Luckily there are simple techniques that enable one to detect most of what is available. See Table 1 of [3] for some statistics on the influence of multiple elimination, outmatching, and recognizing indistinguishable vertices inside a minimum degree code.

Let $G(\mathbf{V}, \mathbf{E})$ be an equivalent representation of $G(V, E)$, where $\mathbf{V}$ is a set of *indistinguishable sets* of vertices, and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ be the edges defined on $\mathbf{V}$. If $\mathbf{V}$ is any partition of the vertices, then $(\mathbf{u}, \mathbf{v}) \in \mathbf{E}$ whenever there exists a $u \in \mathbf{u}$ and $v \in \mathbf{v}$ such that $(u, v) \in E$. Since we insist that $\mathbf{V}$ be a set of indistinguishable sets of vertices, the stronger condition $\mathbf{u} \times \mathbf{v} \subseteq E$ holds.

Figure 2 presents the multiple minimum degree algorithm defined on sets of indistinguishable vertices.¹ The input to the algorithm is the *compressed graph* $G(\mathbf{V}, \mathbf{E})$. The set $\mathbf{V}$

¹The multiple *external* minimum degree algorithm [4] uses an independent set $\mathbf{U}$ where for each $\mathbf{u} \in \mathbf{U}$, $|\text{adj}(\mathbf{u})| - |\mathbf{u}|$ is a minimum.

```

while  $V \neq \emptyset$ 
  choose a (maximal) independent set  $U \subseteq V$  to eliminate
    such that  $|\text{adj}(u)|$  is a minimum for  $u \in U$ 
  for each  $u \in U$ 
    number the vertices in  $u$  next in the ordering and remove  $u$  from  $V$ 
    update the edges
       $E := E \cup (\text{adj}(u) \times \text{adj}(u)) \setminus ((\text{adj}(u) \times \{u\}) \cup (\{u\} \times \text{adj}(u)))$ 
    update  $V$  and  $E$ 
  for  $w$  such that  $(u, w) \in E$  for some  $u \in U$  and  $w$  not outmatched
    update the degree  $|\text{adj}(w)|$ 

```

FIG. 2. Multiple minimum degree defined on $G(V, E)$.

need not contain *maximally indistinguishable* sets of vertices. (Indeed, the graph $G(V, E)$ is the input to all ordering codes known to the author. It is only during the elimination process that indistinguishability is detected and profitably used.)

We will now provide some evidence that it is beneficial to determine the sets of indistinguishable vertices not just *during* the elimination process but also *prior* to the elimination process as a preprocessing step. Matrices that arise from finite element and finite difference discretizations have some degree of compression (meaning that the size of $G(V, E)$ is smaller than $G(V, E)$), if there is more than one solution component at a grid point. For example, a plane elasticity problem has both x - and y -displacements at each grid point; a Navier-Stokes system would have pressure, momentum(s), and energy components at each grid point.

We use a very simple algorithm to construct the compressed graph. Note that if u and v are indistinguishable, then $\text{adj}(u) = \text{adj}(v)$ and so

$$\text{chk}(u) \equiv \sum_{(u,w) \in E} w = \text{chk}(v) \equiv \sum_{(v,w) \in E} w.$$

We compute this *checksum* quantity for each vertex u and sort the vertices by their checksums. This takes $|E| + |V| \log |V|$ time. We then examine each set of vertices that have the same checksum. If $\text{chk}(u) = \text{chk}(v)$ and $|\text{adj}(u)| = |\text{adj}(v)|$, then we compare $\text{adj}(u)$ and $\text{adj}(v)$.

It is quite possible that two vertices u and v could satisfy $\text{chk}(u) = \text{chk}(v)$ and $|\text{adj}(u)| = |\text{adj}(v)|$, yet not be indistinguishable; in this case the effort to compare their adjacency sets would be wasted. Here is a pathological example. Let $G(V, E)$ be the graph with edge set

$$E = \{(u, u)\}_{u \in V} \cup \{(u, |V| - u + 1)\}_{u \in V}$$

for $|V|$ even. The matrix for $G(V, E)$ has a nonzero diagonal and a nonzero anti-diagonal. For each vertex u , $|\text{adj}(u)| = 2$ and $\text{chk}(u) = |V| + 1$. There are only $|V|/2$ pairs of indistinguishable vertices, yet $|V|^2/2$ comparisons could be made to detect them. Note that a random permutation of the vertices reduces the chance that $\text{chk}(u) = \text{chk}(v)$ and that u is not indistinguishable to v to a small probability (approximately $1/3|V|$), so this extreme case is mostly a function of the numbering of the vertices and not the edge set. We have found that the percentage of adjacency set comparisons made between vertices that are *not* indistinguishable is usually negligible.

To illustrate compression of the adjacency graph, we examine a set of test matrices. Most are taken from the Harwell-Boeing test collection [2]. Matrices with prefix "BCSSTK" are stiffness matrices from structural analysis, as is CT20STIF. BCSPWR10 is a power network

TABLE 1
Graph compression statistics.

Matrix	$ V $	$ V $	$ E $	$ E $	$ V / V $	$ E / E $	Indistinguishable vertices	
							Compared	Found
BCSPWR10	5300	5291	16542	16486	1.00	1.00	393	9
BCSSTK15	3948	3948	113868	113868	1.00	1.00	19	0
BCSSTK16	4884	1778	285494	36502	2.75	7.82	3234	3106
BCSSTK17	10974	5219	417676	81062	2.10	5.15	5938	5755
BCSSTK18	11948	10926	137142	122177	1.09	1.12	1239	1022
BCSSTK23	3134	2930	42044	35256	1.07	1.19	339	204
BCSSTK24	3562	892	156348	12756	3.99	12.26	2708	2670
BCSSTK25	15439	13183	236802	161964	1.17	1.46	2410	2256
BCSSTK29	13992	10202	605496	313846	1.37	1.93	4009	3790
BCSSTK30	28924	9289	2014568	222884	3.11	9.04	19658	19635
BCSSTK31	35588	17403	1145828	288806	2.04	3.97	18298	18183
BCSSTK32	44609	14821	1970092	226974	3.01	8.68	30054	29788
BCSSTK33	8738	4344	583166	164284	2.01	3.55	4476	4394
BCSSTK35	30237	6611	1419926	65934	4.57	21.53	24155	23626
BCSSTK36	23052	4351	1120088	37166	5.30	30.14	18779	18701
BCSSTK37	25503	7093	1115474	88924	3.60	12.54	18441	18410
BCSSTK39	46772	10140	2042522	81762	4.61	24.98	4476	4394
CRYSTK02	13965	4765	954618	108882	2.93	8.76	9540	9200
CRYSTK03	24696	8339	1726482	191998	2.96	8.99	36872	36632
CT20STIF	52329	17723	2646134	356666	2.95	7.42	35949	34606

matrix, and CRYSTK02 and CRYSTK03 come from a finite element analysis of a semiconductor device. The graph compression statistics are found in Table 1.

For any given matrix, the compression is not uniform, meaning that any two indistinguishable sets need not contain the same number of vertices, as we see in Table 2. Generally, there can be three displacements and three rotations associated with any mesh point in a structural analysis problem, and some of these components may not be present due to constraints on the structure. At present we cannot explain the presence of indistinguishable sets of size greater than six for matrices BCSSTK30, BCSSTK32, BCSSTK35, BCSSTK36, and CT20STIF. Possibly it is due to the static condensation performed by the MSC NASTRAN program. The number of adjacency set comparisons and number of indistinguishable vertices detected are found in the last two columns of Table 1. Only for four matrices does the percentage of successful comparisons drop below 90%.

3. Empirical results. We now describe some experiments where the uncompressed graph $G(V, E)$ and compressed graph $G(\mathbf{V}, \mathbf{E})$ were ordered using the minimum degree algorithm. The software we used in these experiments is found in BCSLIB-EXT [1], a FORTRAN library for sparse matrix computations. The subroutines that perform the minimum degree ordering are derived from Liu's multiple external minimum degree code GENMMD [4]. We made some very minor modifications in several subroutines where it was assumed that an adjacent vertex was truly a single vertex. We also added a subroutine to compress $G(V, E)$ into $G(\mathbf{V}, \mathbf{E})$, which is called during an initialization phase before the matrix is ordered. The minimum degree ordering of $G(V, E)$ is performed in $|E|$ storage with a small number of work vectors with length $|V|$. This can be reduced to $|E|$ storage with work vectors of size $|V|$ by using the compressed graph $G(\mathbf{V}, \mathbf{E})$, but we have not taken advantage of this sometimes large decrease in working storage in our present codes.

In Table 3 we present the ordering times for our collection of test matrices. We have measured the initialization time, the ordering time, and the total time to order both the uncom-

TABLE 2
Breakdown of sizes of indistinguishable sets.

Matrix	Number of indistinguishable sets of size						
	1	2	3	4	5	6	Other (number, size)
BCSPWR10	5282	9	0	0	0	0	
BCSSTK15	3948	0	0	0	0	0	
BCSSTK16	239	91	1404	6	1	37	
BCSSTK17	2329	1301	1163	1	0	425	
BCSSTK18	10149	583	161	24	0	9	
BCSSTK23	2737	182	11	0	0	0	
BCSSTK24	186	158	76	0	0	472	
BCSSTK25	10937	2236	10	0	0	0	
BCSSTK29	8626	713	299	141	59	364	
BCSSTK30	3852	927	560	939	2192	287	(63,7), (80,8), (31,9), (269,10), (62,11), (14,12), (10,15), (1,16), (2,17)
BCSSTK31	6824	5266	4346	9	590	368	
BCSSTK32	4852	3091	2274	69	749	3770	(16,7)
BCSSTK33	1801	692	1851	0	0	0	
BCSSTK35	981	627	653	4	55	4290	(1,12)
BCSSTK36	188	209	194	4	690	3064	(2,7)
BCSSTK37	1358	1469	1096	0	1101	2069	
BCSSTK39	982	0	0	0	9158	0	
CT20STIF	5441	3451	4332	42	156	4225	(8,7), (25,8), (26,9), (1,10), (16,12)
CRYSTK02	139	52	4574	0	0	0	
CRYSTK03	126	69	8144	0	0	0	

pressed and compressed graphs. The initialization for $G(V, E)$ takes $O(|V|)$ operations where a small number of work vectors with size $|V|$ are set. The initialization for the compressed graph $G(\mathbf{V}, \mathbf{E})$ does include this $O(|V|)$ work, and the graph compression is also done at this step. The compression is dominated by an $O(|E|)$ term to compute the checksums and an $O(|V| \log |V|)$ term to sort the vertices by their checksum. As one can see from the columns labeled “init,” the initialization time for $G(V, E)$ is trivial while that for $G(\mathbf{V}, \mathbf{E})$ can be substantial.

The columns labeled “order” contain the time spent in the ordering software but do not include the initialization nor any postprocessing time. The ratio of these pure ordering times is given in the last column of Table 3. Note the fairly good correlation with the ratio $|V|/|\mathbf{V}|$ found in Table 1.

The “overall speedup” measures the improvement by using the compressed graph in place of the uncompressed graph. This assumes that only one run of the ordering software is made, for the total time to order $G(\mathbf{V}, \mathbf{E})$ includes the time to construct the compressed graph. The speedup in the column labeled “order” assumes the cost to construct $G(\mathbf{V}, \mathbf{E})$ is negligible or has been amortized.

In practice, the time spent to order a large matrix is small when compared with the time to factor the matrix. It may be worthwhile to make several runs of the ordering software using permuted adjacency structures as input. Each run would produce a permutation vector and the best could be chosen.² A small percentage decrease in factorization time can easily offset several ordering runs. The compressed graph need be constructed only once, no matter how many runs are made using the permuted adjacency structure, and so the cost to compress the

²“Several” should be small, for if n runs have been made, the probability that the next run will give a better permutation than the previous n is exactly $1/(n + 1)$. We generally limit the number of runs to five.

TABLE 3
Ordering time statistics.

Matrix	$G(V, E)$ CPU			$G(V, E)$ CPU			Speedups	
	Init	Order	Total	Init	Order	Total	Overall	Order
BCSPWR10	0.01	0.21	0.22	0.06	0.25	0.31	0.71	0.84
BCSSTK15	0.00	1.04	1.04	0.09	0.91	1.00	1.04	1.14
BCSSTK16	0.01	0.50	0.51	0.17	0.18	0.35	1.46	2.78
BCSSTK17	0.01	1.32	1.33	0.30	0.48	0.78	1.71	2.75
BCSSTK18	0.03	1.75	1.78	0.18	1.76	1.94	0.92	1.00
BCSSTK23	0.00	0.97	0.97	0.04	0.90	0.94	1.03	1.08
BCSSTK24	0.01	0.23	0.24	0.09	0.05	0.14	1.71	4.60
BCSSTK25	0.03	2.92	2.95	0.27	2.42	2.69	1.10	1.21
BCSSTK29	0.02	2.15	2.17	0.46	1.80	2.06	1.05	1.19
BCSSTK30	0.06	3.50	3.56	1.15	1.15	2.30	1.55	3.04
BCSSTK31	0.07	5.60	5.67	0.92	3.38	4.30	1.32	1.66
BCSSTK32	0.08	5.22	5.30	1.35	2.06	3.41	1.55	2.53
BCSSTK33	0.01	1.66	1.67	0.36	0.70	1.06	1.58	2.37
BCSSTK35	0.05	3.21	3.26	0.90	0.77	1.67	1.95	4.17
BCSSTK36	0.04	1.93	1.97	0.68	0.42	1.10	1.79	4.60
BCSSTK37	0.05	2.28	2.33	0.73	0.67	1.40	1.66	3.40
BCSSTK39	0.09	6.29	6.38	1.38	1.09	2.47	2.58	5.77
CT20STIF	0.11	9.45	9.56	1.89	3.39	5.28	1.81	2.79
CRYSTK02	0.02	1.61	1.63	0.52	0.54	1.06	1.53	2.98
CRYSTK03	0.05	3.12	3.17	0.98	0.90	1.88	1.69	3.47

graph can be amortized. The true speedup achieved by multiple runs using the compressed graph will lie between the “overall” and “order” speedups.

It is widely believed that the minimum degree algorithm spends the majority of its time updating the degrees of vertices. Most codes differentiate between two types of vertices: a “2-adjacent” vertex w such that $\text{adj}(w) = \text{adj}(u) \cup \text{adj}(v)$ where u and v have been eliminated, and all others. The 2-adjacent vertices are used to detect indistinguishability and outmatching, and the cost to update the degree of a 2-adjacent vertex is usually less than the cost to update a vertex of the other type.

Table 4 contains the number of degree updates performed during the orderings. The number of 2-adjacent updates is virtually equal for both compressed and uncompressed graphs, for the number of 2-adjacent vertices during the factorization measures in some sense a property of the factor graph. The factor graphs produced by the two orderings are not identical (except for BCSSTK15, which has no compression), but they are very similar.

The far-right column holds the ratio of total updates performed when ordering the uncompressed and compressed graphs. The correlation between the ratio of updates and the “order speedup” of Table 3 is less strong than the correlation of the ratio $|V|/|V|$ and “order speedup.” This can partially be explained by noting that the cost of a degree update is a function of the sizes of the adjacency sets. The number of entries in $\text{adj}(u)$ will generally be greater than that of $\text{adj}(u)$. The greater number of updates for $G(V, E)$ and their greater cost better reflect the differences between ordering times than just their number alone.

4. Summary and extensions. We have shown that compressing a graph $G(V, E)$ into a graph $G(V, E)$ defined on sets of indistinguishable vertices has the potential to reduce the overall ordering time of the minimum degree algorithm whenever the original and compressed graphs differ in size. Our collection of test matrices contains a power network matrix, 17 matrices from structural analysis and 2 matrices from semiconductor applications. The geometric mean of the speedups is 1.43 when the cost to compress the graph is included in the time to

TABLE 4
Comparison of number of degree updates.

Matrix	$G(V, E)$			$G(V, E)$			Ratio
	2-adj	Other	Total	2-adj	Other	Total	
BCSPWR10	2586	8489	11075	2587	8483	11020	1.00
BCSSTK15	2133	18803	20936	2133	18803	20936	1.00
BCSSTK16	1664	5672	7336	1660	1853	3513	2.09
BCSSTK17	4506	21170	25676	4479	9161	13640	1.88
BCSSTK18	7741	39989	47730	7787	39081	46868	1.02
BCSSTK23	1422	22535	23957	1488	21419	22907	1.05
BCSSTK24	691	3875	4566	741	1010	1751	2.61
BCSSTK25	9923	69312	79235	9947	58823	68770	1.15
BCSSTK29	6760	43302	50062	5685	31917	37602	1.33
BCSSTK30	9829	39271	49100	9245	15609	24854	1.98
BCSSTK31	19428	81280	100708	19248	47978	67226	1.50
BCSSTK32	17093	82144	99237	17110	32017	49127	2.20
BCSSTK33	3825	18219	22044	3754	7960	11714	1.88
BCSSTK35	6960	55330	62290	7186	13007	20193	3.08
BCSSTK36	6016	29861	35877	6165	5665	11830	3.03
BCSSTK37	8880	36104	44984	9077	10275	19352	2.32
BCSSTK39	10125	62220	72345	10166	12646	22812	3.17
CT20STIF	16118	143753	159871	16335	56541	72876	2.19
CRYSTK02	4761	8299	13060	4622	3302	7924	1.65
CRYSTK03	7803	14042	21845	8028	3094	11122	1.96

order $G(V, E)$. When the compression cost is neglected, the geometric mean of the speedups is 2.30.

A compressed graph has other applications. We have software for performing profile orderings of sparse matrices based on compressed graphs. The savings in CPU time are larger than we have observed for the minimum degree algorithm, for the time for a profile ordering has a strong $O(|E|)$ component, which is reduced to $O(|E|)$ for a compressed graph. Compressed graphs can also be used when improving separators [5].

We conjecture that they can be profitably used in spectral nested dissection [7]. The Laplacian matrix Q of $G(V, E)$ has the entries

$$q_{u,v} = \begin{cases} -|\text{adj}(u)| + 1 & \text{if } u = v, \\ 1 & \text{if } u \neq v \text{ and } (u, v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

It would be straightforward to consider the smaller Laplacian matrix defined on $G(V, E)$. We conjecture that the generalization of the Laplacian matrix for $G(V, E)$ given by

$$q_{u,v} = \begin{cases} -|u|(|\text{adj}(u)| - |u|) & \text{if } u = v, \\ |u||v| & \text{if } u \neq v \text{ and } (u, v) \in E, \\ 0 & \text{otherwise} \end{cases}$$

shares the same properties of Q that enable one to detect connected components from the eigenspace associated with the second eigenvalue.

It may be beneficial for extremely large matrices to relax the definition of indistinguishability. For example, if

$$|\text{adj}(u) \cap \text{adj}(v)| \geq \tau |\text{adj}(u) \cup \text{adj}(v)|,$$

then vertices u and v are τ -indistinguishable, where $0 \leq \tau \leq 1$. A set of vertices $\mathbf{u}$ is τ -indistinguishable if its members are pairwise τ -indistinguishable. Using this technique one can construct a sequence of graphs that vary in size and approximate the usual $\tau = 1$ compressed graph. This could have application in two areas: constrained minimum degree [6], where a single run of minimum degree is performed to find a separator that is then improved in some manner, and spectral nested dissection, where a sequence of approximations to the second eigenvector is computed.

Acknowledgments. We would like to thank Stan Eisenstat and Joseph Liu for helpful correspondence during this research and Roger Grimes and Dan Pierce for their careful reading of this manuscript. Stan Eisenstat provided a pathological example matrix to show the worst case complexity of the number of adjacency set comparisons. Fritz Scholz and Antonio Possolo continue to answer my questions on probability and statistics. Comments by the referees have helped to improve the presentation of this paper.

REFERENCES

- [1] BCSLIB-EXT MATHEMATICAL LIBRARY, Document 20462-0520-R2, Boeing Computer Services, Seattle, WA, 1991.
- [2] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.
- [3] J. A. GEORGE AND J. W. H. LIU, *The evolution of the minimum degree ordering algorithm*, SIAM Rev., 31 (1989), pp. 1–19.
- [4] J. W. H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.
- [5] ———, *A graph partitioning algorithm by node separators*, ACM Trans. Math. Software, 15 (1989), pp. 198–219.
- [6] ———, *The minimum degree ordering with constraints*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1136–1145.
- [7] A. POTHEN, H. D. SIMON, AND K-P. LIOU, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.