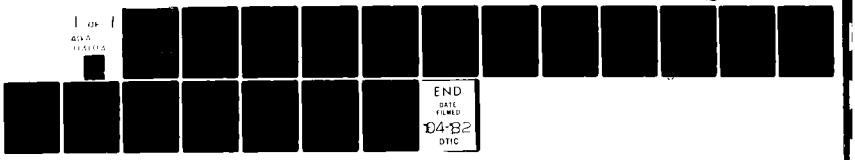
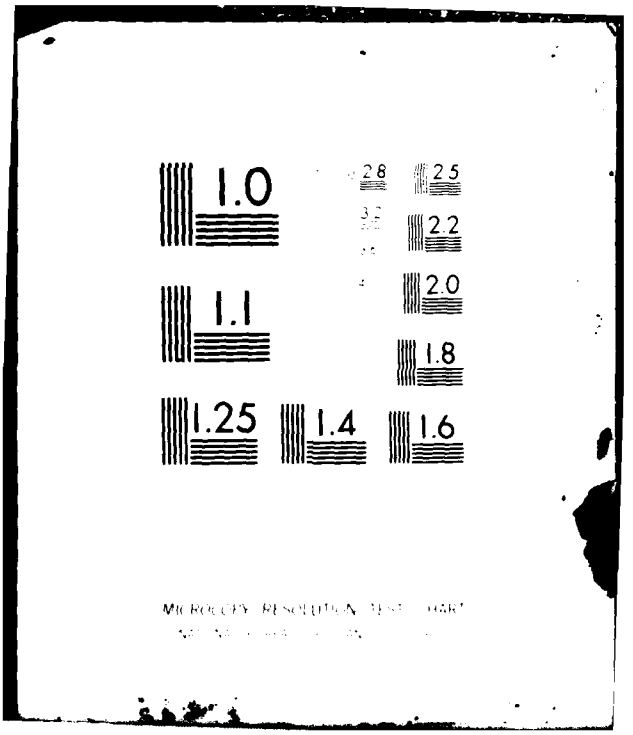
AD-A113 104 BOEING COMPUTER SERVICES CO TUKWILA WA F/G 12/1
IMPLEMENTATION OF THE GIBBS-POOLE-STOCKMEYER ALGORITHM AND THE --ETC(U)
JAN 82 J G LEWIS F49620-81-C-0072
UNCLASSIFIED AFOSR-TR-82-0264 NL



END
DATE FILMED
04-82
DTIC


1.0
1.1
1.25
1.4
1.6
2.0
1.8
2.2
2.5

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR- 82-0264	2. GOVT ACCESSION NO.: AD-A113104
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): IMPLEMENTATION OF THE GIBBS-POOLE-STOCKMEYER ALGORITHM AND THE GIBBS-KING ALGORITHM	5. TYPE OF REPORT & PERIOD COVERED: TECHNICAL
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John Gregg Lewis	8. CONTRACT OR GRANT NUMBER(s): F49620-81-C-0072
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Boeing Computer Services Co. 565 Andover Park West Tukwila WA 98188	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F; 2304/A3
11. CONTROLLING OFFICE NAME AND ADDRESS: Mathematical & Information Sciences Directorate Air Force Office of Scientific Research Bolling AFB DC 20332	12. REPORT DATE: JAN 82 13. NUMBER OF PAGES: 17
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Bandwidth reduction, profile reduction, wavefront reduction, sparse matrix, banded matrix, matrix reordering.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Implementations of two effective matrix reordering algorithms were published in this paper as Algorithms 508, which implements the Gibbs-Poole-Stockmeyer algorithm for reducing the bandwidth of a matrix, and 509, which implements the Gibbs-King algorithm for reducing the profile of a matrix. Reduction of matrix profile is more often required than bandwidth reduction, but Algorithm 509 is substantially slower than Algorithm 508. Consequently, the Gibbs-Poole-Stockmeyer algorithm has been recommended and used in contexts (CONTINUED)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED: where the better profile reduction provided by the Gibbs-King algorithm would be more appropriate. In addition, Algorithms 508 and 509 both contain unnecessary restrictions on problem size and provide little error checking. The authors describe a new FORTRAN implementation of both reordering algorithms which is portable, faster, more reliable and uses less storage than the original implementations. The new implementation of the Gibbs-King algorithm is much faster than Algorithm 509, generally slightly faster than Algorithm 508 and nearly as fast as the new implementation of the Gibbs-King-Stockmeyer algorithm.

IMPLEMENTATION OF THE GIBBS-POOLE-STOCKMEYER ALGORITHM AND THE GIBBS-KING ALGORITHM

John Gregg Lewis

Boeing Computer Services Co.

Implementations of two effective matrix reordering algorithms were published in this journal as Algorithms 508 [1], which implements the Gibbs-Poole-Stockmeyer algorithm for reducing the bandwidth of a matrix, and 509 [9], which implements the Gibbs-King algorithm for reducing the profile of a matrix. Reduction of matrix profile is more often required than bandwidth reduction, but Algorithm 509 is substantially slower than Algorithm 508. Consequently, the Gibbs-Poole-Stockmeyer algorithm has been recommended and used in contexts where the better profile reduction provided by the Gibbs-King algorithm would be more appropriate. In addition, Algorithms 508 and 509 both contain unnecessary restrictions on problem size and provide little error checking. We describe a new FORTRAN implementation of both reordering algorithms which is portable, faster, more reliable and uses less storage than the original implementations. The new implementation of the Gibbs-King algorithm is much faster than Algorithm 509, generally slightly faster than Algorithm 508 and nearly as fast as the new implementation of the Gibbs-Poole-Stockmeyer algorithm.

Key Words and Phrases: bandwidth reduction, profile reduction, wavefront reduction, sparse matrix, banded matrix, matrix reordering
CR categories: 5.14, 5.32
The Algorithm: The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices. ACM Trans. Math. Softw.

Author's address: Boeing Computer Services Co., Mail Stop 9C-01, 565 Andover Park West, Tukwila, Wa. 98188. This work was supported in part by the Air Force Office of Scientific Research under contract F49620-81-C-0072.

1. Introduction

Many algorithms and programs which solve sparse linear equations use Gaussian elimination in combination with a reordering of the coefficient matrix to preserve sparsity. One common approach for problems where the coefficient matrix is symmetric or structurally symmetric is to use symmetric row and column interchanges to reduce the bandwidth, the profile, or the wavefront of the matrix. This approach is known to be less than optimal for large problems derived from regular finite element discretizations. There is much recent literature on algorithms which use more sophisticated data structures than those of a "banded" matrix to achieve lower costs for the numerical factorization and solution phases of very large problems (see [2] for an excellent survey). Determining the reordering and setting up the data structures for these algorithms is substantially greater in cost than for the banded reorderings provided by the Gibbs-Poole-Stockmeyer algorithm (GPS) and the Gibbs-King algorithm (GK). For moderately large problems the banded or variable banded linear equation solvers often have smaller total cost than their more sophisticated counterparts. Problems for which banded orderings are appropriate occur frequently in finite element analyses of structures (see [4], [8], [12] for examples).

Banded orderings have obvious virtues on vector computers like the CDC CYBER 205 or the CRAY-1, where band Gaussian elimination can take much more advantage of the vector operations than can more general sparse elimination schemes. The simplicity of banded data structures favors band algorithms, even for large problems, on sequential machines like the CDC CYBER 175 computer where arithmetic is fast and memory accesses are slow. Very large problems can be

solved with sophisticated sparse solvers which use banded algorithms to solve subproblems [7]. These are reasons enough to return our attention to bandwidth reduction algorithms.

The Gibbs-Poole-Stockmeyer algorithm [1], [11] has been shown to be an effective tool for reducing the bandwidth and profile of finite element matrices arising in structural engineering problems [4], [10], [16]. The Gibbs-King variation [9] of this algorithm provides better reduction of the profile and wavefront. However, this latter algorithm, as implemented as Algorithm 509, is often much slower in execution than its predecessor, Algorithm 508, which implemented the GPS algorithm; speed of execution is an important virtue of Algorithm 508. In this paper we present a new implementation, GPSKCA [14], of both algorithms. The new implementation executes more quickly than either Algorithm 508 or 509. The speedup for the GK algorithm is dramatic, and reduces its execution time to nearly that of the quick GPS algorithm. In addition, the new implementation provides better bounds checking, improved diagnostics for erroneous inputs, and more efficient use of space.

A working knowledge of the original Gibbs-Poole-Stockmeyer paper [11] is assumed throughout this paper. Definitions of the graph-theoretic terms in this paper will be found therein.

2. Outline of the algorithms

We assume that the coefficient matrix is symmetric or at least has its nonzero entries occurring in symmetric locations. Further, we assume that diagonal pivoting (symmetric row and column interchanges) does not produce numerical instability. These assumptions make it possible to carry out the

reordering algorithms in the context of the graph of the matrix: each equation corresponds to a single node in the graph; there is an edge or arc connecting nodes i and j only if the i,j-th element of the matrix is nonzero.

The two reordering algorithms consist of three phases, and they differ only in the last phase. The phases are:

1. find a pseudo-diameter of the graph (a heuristic for finding two nodes far apart in the graph)
2. combine the level structures rooted at either end of the pseudo-diameter into a more general level structure.
3. number the nodes according to their locations in the level structure, using a generalization of
   a. the reverse Cuthill-McKee algorithm (GPS), or
   b. King's algorithm (GK).

Discussions of the details of these phases are given in [9] and in [11].

3. Appropriate data structures

The key operations performed by these algorithms involve creating or operating upon level structures. A level structure for a graph is an ordered list of equivalence classes of nodes such that, for every node v, all of v's immediate neighbors are in the same equivalence class, the immediately preceding class or the immediately succeeding class. Rooted level stuctures are level structures whose first level consists of a single node u, and where all succeeding levels consist of nodes with equal minimum distance to u. There are two obvious and compact representations for such structures in FORTRAN:

1. an n-vector containing in each entry the level number at which the corresponding node is found

2. a list of the nodes in each level, with as many lists as there are levels, represented by an n-vector of node numbers together with a shorter vector giving the index of the initial node in the corresponding level.

These two representations provide equivalent information with differing costs. The first form is efficient for deciding to which level a particular node belongs. The second form provides easy access to the set of nodes forming a given level. Each representation requires $O(n)$ operations to extract the information for which the other representation is efficient.

In the GPS algorithm the second representation of a level structure is the more appropriate form for phase 1. The first representation is more efficient for phase 2. The third phase can be performed most easily if both representations are available. Only the first representation is used in the original implementation. Our new implementation uses whichever representations are more appropriate for each phase. Between phases the representations are transformed from one form to another. The overhead of the transformations is recovered by using more appropriate data structures in other parts of the algorithm, particularily in phase 3. The use of the second representation also reduces the storage requirements in phase 1 for symmetrically reducible matrices, which are encountered with surprising frequency in structural engineering contexts.

An opportunity for improving the choice of data structures is particularly evident in phase 3b, Gibbs' variant of the King algorithm. In this numbering scheme there are three sets S2, S3 and Q, which are described as "queues" of integers. Neither S2 nor S3 is a queue. Both are priority queues [13]: while additions occur only at the end of the queue, deletions are made on the

basis of a merit function which depends only partly on location within the queue. Our implementation of the Gibbs-King algorithm represents S2 and S3 by simple linked lists, which reduces the complexity of the deletion operations which occur on both priority queues. These priority queues are represented awkwardly by one-dimensional (unlinked) vectors in Algorithm 509.

The interaction between the data structures and the Gibbs-King merit function provides the most dramatic improvement over the original implementation. The merit rule for the numbering is: number the first node in S2 with the fewest connections to nodes in S3, then delete from S3 all nodes connected to the newly numbered node. This latter operation makes the connection counts from S2 to S3 dynamic. It is in repeatedly computing connection counts that King's numbering is more complex than the Cuthill-McKee numbering. In Algorithm 509 these connection counts are recomputed each time a new node is to be numbered, with membership in S3 being determined by a sequential search through a list. We compute the connection counts in the new implementation only at the beginning of an outer iteration (once for each level). On each inner iteration (numbering of a node), we update the counts for only those elements in S2 which connect to elements being deleted from S3. This is performed efficiently by using the connectivity information for S3 together with an encoding of both membership in S2 and connectivity counts in the n-vector holding the first representation of the level structure.

4. Storage Requirements

The changes in data structures produce faster algorithms without problem restrictions, but also necessitate different calling sequences for the

algorithms. Two major changes in the representation of the problem have been made, both of which usually reduce the overall storage requirements of the algorithm. The first change is to the workspace needed for the reordering. The variety of internal data structures are all created in a contiguous workspace provided by the user. Consequently, all working space for the reordering is given as a single vector, whose length is specified by the user; the new code checks that it does not access elements beyond this length. A vector of length $6n+3$ will suffice for any problem, but a vector of length $4n$ suffices for most problems. The actual space used for any particular problem is reported back to the user. (The space required ranges from $2n$ at one extreme, a diagonal matrix, to $6n+3$ at the other extreme, a tridiagonal matrix.) By contrast, Algorithms 508 and 509 require $5n + 802$ words of working storage for any problem, and will be unable to correctly reorder certain sparse symmetric matrices.

A more dramatic reduction in space requirements has been made by changing the representation of the graph or the connectivity structure of the matrix to a more compact form. The matrix is represented by a connectivity or adjacency table in both implementations. This table gives a list of the nodes to which each node is connected. Equivalently, we require for each row of the matrix a list of the columns in which nonzero entries occur off the diagonal. Algorithms 508 and 509 require that this table be a rectangular array, with n rows and at least as many columns as the maximum degree of any node (the maximum number of off-diagonal nonzeroes in any row). An additional n-vector gives the actual degree for each node. This table is given to the new implementation as a list of lists, where the connections for each node are given in consecutive locations. This requires a vector of length nz, the total number of off-diagonal

nonzeroes. (The full symmetric structure is needed in both cases for efficiency.) We also require one additional vector, of length n, giving in each entry the index in the connectivity table of the first node in the sublist for the corresponding row. A second vector giving the degree of each node (the length of the corresponding connection list) is still needed. This form can never require more than n words more storage than the original and will require much less storage whenever the number of nonzeroes in rows varies significantly. It also permits the use of compressed subscript storage schemes [3] and is quite similar to the structure used in [6].

The storage savings provided by the compact representation of the connection table dominate the differences in space requirements between the new and old implementations, as reported in the following section. The reasons for preferring this format are evident. However, for compatibility with Algorithms 508 and 509, a version of the new implementation which uses the old format for the connection table is available from the author (not through the ACM distribution service). This alternative provides the time savings and the added reliability of the new implementation, but little of the storage savings.

5. Empirical results

The original and the new implementations of the two reordering algorithms were compared on 30 test matrices collected by Everstine [4]. These test problems are an outgrowth of the 19 problems used in the original papers [9] and [11]. Everstine has reported the comparative timing and effectiveness of Algorithm 508 and other banded reordering strategies, but his paper excluded Algorithm 509 because of its relatively poor execution speed. The following

results show that excluding the Gibbs-King algorithm is unwarranted; an efficient implementation of the Gibbs-King Algorithm compares very well in terms of both speed and effectiveness.

We give three sets of results in this section. First we give relative timings for the new implementation of the GPS algorithm and the GK algorithm, and for Algorithms 508 and 509. The second set of results is the relative effectiveness of the reorderings, in the same form as reported in [4]. This makes it possible to include the Gibbs-King Algorithm in the comparisons in [4]. Finally we give the storage requirements for the new and the old implementations of the algorithms.

The timing results are given in Tables 1 and 2. The times in Table 1 were obtained on a CDC Cyber 175 computer, using the FTN 4.6 FORTRAN compiler with optimization level 2. The times in Table 2 were observed on an IBM 3032, using FORTRAN H Extended with optimization level 2, using full length integers. Both tables clearly show the increased speed of the new implementation and also demonstrate that an efficient implementation of the GK algorithm can be nearly as fast as an efficient implementation of the GPS algorithm.

Table 3 contains the reordering results from the GK algorithm needed to complete Everstine's comparison of banding reordering algorithms. These results show that, on half of the test problems, the GK algorithm performs better than all of the algorithms considered by Everstine. Everstine concluded that Algorithm 508 was the best algorithm because of its effectiveness and speed; the Gibbs-King algorithm always performs at least as well as the Gibbs-Poole-Stockmeyer algorithm on these test problems, and is faster in its new implementation than the version of Algorithm 508 used by Everstine.

The new implementations are more efficient in storage requirements. In Table 4 we give the minimum storage requirements for all of the vector arguments to the reordering algorithms, including the representation of the adjacency structure of the matrix. For the new implementation, this is based on the minimum workspace requirement reported back by the code, which is less than what the user would need without a priori knowledge. The actual runs used a working vector of length 5n; the last column of Table 4 gives the minimum length (in units of n), which is used to compute the first column of this table. The requirements for the old implementations are those of the vector and matrix inputs, plus 801 words for the required named COMMON areas. We also give the workspace requirements for a machine like the IBM 370 series, where short integers suffice to store numbers as large as n. We assume in Table 4 that the longer integer representation requires twice as much storage as the shorter representation. The results are given in "long integer words". Thus, the actual requirements in bytes for an IBM 370 or DEC VAX 11 computer would be four times as large.

The space requirements do not include the space occupied by the reordering codes, which is clearly a system dependent quantity. As a single benchmark, the code produced by the FTN compiler for the CDC CYBER 175 required 2163 words for the new algorithms and 1434 words for the old algorithms. Even though the new, faster implementation requires more storage for the code itself, the total space required will usually be less.

6. Concluding Remarks

The use of more appropriate data structures leads to reordering algorithms which are more efficient in time and space. The new implementation is more robust than are the original implementations because the change in data structures, and the concomitant use of a single workspace array, makes it possible to remove several hidden restrictions on the matrices which can be reordered. The new implementation also has significant internal error checking for improper input. Both implementations, for example, require that the structure of the matrix be symmetric -- the i,j-th element is nonzero if and only if the j,i-th element is nonzero. Neither implementation checks this explicitly because of cost, but in most cases the new implementation will detect an inconsistency in its data structures and halt the reordering.

There are possibilities for further enhancement of the speed of these algorithms. Several of these were considered in the development of these codes, but rejected. The new implementation uses insertion sorts for the numerous lists which are sorted; an asymptotically faster method like quicksort [15] might provide some speedup. However, the lists to be sorted are usually short and the total time spent in sorting is small. The use of quicksort would gain little in time while requiring additional space for code. Further, the instability [13] of the sorting algorithm would change the strategies for breaking ties, thus changing the final reordering. For these reasons, the new implementation uses insertion sorts and produces the same reorderings as Algorithms 508 and 509.

A more significant change in the algorithm is suggested in [5] and [6]. The initial phase of both algorithms, finding a pseudo-diameter or a pair of

pseudo-peripheral nodes, is critical to the success of the reordering, but it can also be expensive. George and Liu examine several modifications to the heuristic used in [9], [10], and [11]. They conclude that a quicker heuristic is also more effective, at least in the context of the orderings they consider in [6]. We experimented with their recommended "S2" heuristic, but took advantage of the context of algorithms 508 and 509 to begin the pseudo-diameter calculation with a node of minimum degree rather than an arbitrary node. This selective choice of starting node improves the performance of both pseudo-peripheral node finders, especially the original GPS heuristic. We concluded that the faster heuristic also carries the risk of significantly reducing the effectiveness of the algorithm. The implementation published in [14] uses the same heuristic for computing the pseudo-diameter and produces the same reorderings as the original implementations of algorithms 508 and 509.

ACKNOWLEDGMENT

The author wishes to thank W. G. Poole, Jr. for making available copies of the original implementation of the algorithms and for his helpful criticism of this paper. D. S. Dodson and A. M. Erisman also helped to improve the presentation, and G. C. Everstine provided his collection of test matrices.

REFERENCES

[1] Crane, H. L. Jr., Gibbs, N. E., Poole, W. G. Jr., and Stockmeyer, P. K. Algorithm 508. Matrix Bandwidth and Profile Reduction, ACM Trans. Math. Software 2, 4, (Dec. 1976), 375-377

[2] Duff, I. S. A Survey of Sparse Matrix Research, Proc. IEEE 65, 4, April 1977, 500-535

[3] Eisenstat, S. C., Schultz, M. A., and Sherman, A. W. Efficient Implementation of Sparse Symmetric Gaussian Elimination, Proceedings of the AICA International Symposium on Computer Methods for PDE's, 1975, 33-39

[4] Everstine, G. C. A Comparison of Three Resequencing Algorithms for the Reduction of Matrix Profile and Wavefront, International Journal for Numerical Methods in Engineering 14, 1979, 837-853

[5] George, A. Solution of Linear Systems of Equations: Direct Methods for Finite Element Problems, in Sparse Matrix Techniques, Copenhagen 1976, Dold. A., and Eckman, B., ed., Springer-Verlag, Berlin, 1977

[6] George, A. and Liu, J. W. H. An Implementation of a Pseudoperipheral Node Finder, ACM Trans. Math. Software 5, 3, (Sept. 1979), 284-295

[7] George, A. and Liu, J. W. H. Algorithms for Matrix Partitioning and the Numerical Solution of Finite Element Systems, SIAM J. Numer. Anal. 15, 2, April 1978, 297-327

[8] George, A., Liu, J., and Ng, E. User Guide for Sparspak: Waterloo Sparse Linear Equations Package, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, 1979

[9] Gibbs, N. E. Algorithm 509. A Hybrid Profile Reduction Algorithm, ACM Trans. Math. Software 2, 4, (Dec. 1976), 378-387.

[10] Gibbs, N. E., Poole, W. G. Jr., and Stockmeyer, P. K. A Comparison of several Bandwidth and Profile Reduction Algorithms, ACM Trans. Math. Software, 2, 4 (Dec. 1976), 322-330.

[11] Gibbs, N. E., Poole, W. G. Jr., and Stockmeyer, P. K. An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix, SIAM J. Numer. Anal. 13, 2 (April 1976), 236-250.

[12] Jennings, A. Matrix Computations for Engineers and Scientists, John Wiley and Sons, London, 1977

[13] Knuth, D. E. The Art of Computer Programming, Volume 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

[14] Lewis, J. G. Algorithm ___. The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices, this journal

[15] Sedgewick, R. Implementing Quicksort Programs, Comm. ACM 21, 10, October 1978, 847-856

[16] Thierer, A. A Comparison of Ordering Schemes for Profile Minimization of Sparse Symmetric Matrices, Center for Numerical Analysis, University of Texas report CNA-146, Austin, Texas, 1978

Table 1

Relative Execution Times for the New and Old Implementations

Of the Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms

(CDC Cyber 175)

n	Time in Seconds for New GPS	Relative Execution Time* for Old GPS	New GK	Old GK
59	.006	1.3	1.4	2.4
66	.006	1.5	1.4	2.1
72	.006	1.3	1.4	2.3
87	.009	1.4	1.5	4.6
162	.017	1.5	1.5	4.2
193	.078	2.1	1.3	12.2
198	.036	1.7	1.2	2.5
209	.026	1.5	1.5	16.6
221	.030	1.5	1.4	4.2
234	.033	1.9	1.3	3.1
245	.031	1.5	1.4	14.1
307	.045	1.6	1.4	22.1
310	.044	1.6	1.4	3.7
346	.059	1.5	1.4	16.2
361	.035	1.7	1.5	7.3
419	.055	1.5	1.5	14.5
492	.059	1.8	1.4	6.2
503	.069	1.6	1.7	36.2
512	.163	2.1	1.2	3.9
592	.101	1.6	1.4	9.2
607	.123	1.7	1.3	16.5
758	.120	2.1	1.3	5.2
869	.202	1.6	1.3	7.6
878	.223	1.6	1.2	7.6
918	.188	1.6	1.3	9.6
992	.555	1.6	1.2	6.7
1005	.136	1.6	1.6	50.6
1007	.271	1.6	1.2	7.4
1242	.321	1.5	1.3	34.0
2680	.448	1.7	1.5	32.6

*Relative to time for the new implementation of the GPS algorithm.

Table 2

Relative Execution Times for the New and Old Implementations

Of the Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms

(IBM 3032)

n	Time in Seconds for New GPS	Relative Execution Time* for Old GPS	New GK	Old GK
59	.008	1.2	1.2	1.9
66	.009	1.4	1.2	1.9
72	.008	1.3	1.4	2.1
87	.012	1.3	1.4	2.8
162	.024	1.4	1.3	2.6
193	.111	1.6	1.3	5.1
198	.052	1.5	1.2	1.9
209	.037	1.3	1.5	7.1
221	.042	1.4	1.3	2.6
234	.044	1.6	1.2	2.3
245	.043	1.3	1.4	6.6
307	.064	1.3	1.3	9.1
310	.061	1.5	1.3	2.4
346	.083	1.3	1.4	6.7
361	.047	1.6	1.5	4.1
419	.078	1.4	1.4	6.3
492	.084	1.6	1.3	3.5
503	.095	1.5	1.6	13.9
512	.208	1.7	1.1	2.5
592	.139	1.5	1.3	4.5
607	.164	1.5	1.3	7.4
758	.164	1.9	1.3	3.2
869	.286	1.4	1.2	3.7
878	.312	1.3	1.2	3.7
918	.266	1.4	1.2	4.5
992	.773	1.3	1.2	3.1
1005	.192	1.5	1.5	19.3
1007	.385	1.3	1.2	3.6
1242	.458	1.3	1.3	13.0
2680	.601	1.7	1.4	13.4

*Relative to time for the new implementation of GPS algorithm.

Table 3

Bandwidth, Profile and Wavefront Computed by the Gibbs-King Algorithm

n	Bandwidth	Profile	RMS Wavefront	Max Wavefront	Average Wavefront
59	11	255	5.53+	8	5.32*
66	3	127	2.94+	3	2.92*
72	12	172	3.46+	4	3.39
87	20	595	8.24+	12	7.84*
162	17	1417	10.09	13	9.75
193	45	4416	24.86	36	23.88
198	11	1115	6.95+	9	6.63*
209	48	3823	20.40	32	19.29
221	20	2002	10.53	17	10.06
234	18	1115	6.29	12	5.76
245	48	3568	16.64	27	15.56
307	63	7825	27.36	35	26.49
310	28	2696	9.85	16	9.70
346	55	7335	23.29	34	22.20
361	14	4699	14.23+	15	14.02*
419	41	7654	19.96	30	19.27
492	37	5021	11.99	22	11.21
503	69	14539	32.22+	50	29.90*
512	29	4309	11.43+	23	9.42*
592	47	10333	19.70+	33	18.45*
607	78	14153	27.25+	38	24.32*
758	25	7417	12.01+	25	10.78*
869	62	14859	19.87	37	18.10*
878	40	18818	22.60+	25	22.43*
918	64	19580	23.39+	39	22.33*
992	35	33076	34.66+	36	34.34*
1005	135	39136	44.80	89	39.94*
1007	54	21422	22.60+	33	22.27*
1242	129	51710	46.26	84	42.63
2680	89	96746	38.03+	60	37.10*

+ would supplant best algorithm for reducing RMS wavefront in Everstine's Table 2 [4].

* would supplant best algorithm for reducing average wavefront or profile in Everstine's Table 3 [4].

Table 4

Workspace Requirements for Old and New Implementations

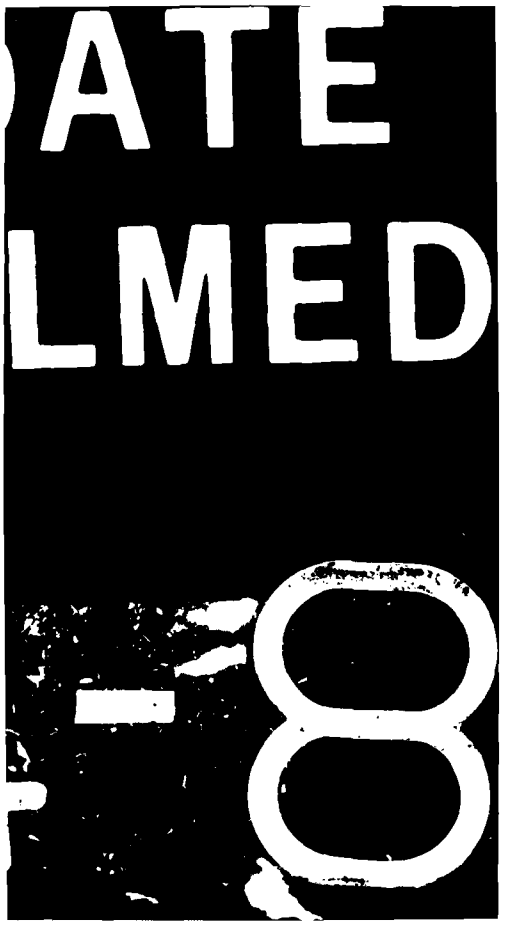
of the GPS and GK Algorithms

	Unpacked Integers		Packed Integers*		Workspace Factor
n	Old	New+	Old	New+	(*n)
59	1510	607	1363	332	3.76
66	1594	752	1429	408	4.55
72	1594	651	1450	361	3.96
87	2455	1021	1933	553	3.52
162	3232	2064	2584	1112	3.44
193	7750	4485	4952	2338	3.14
198	4366	2544	3277	1370	3.82
209	5609	2830	3937	1519	3.20
221	4780	2821	3565	1520	3.39
234	4546	2208	3493	1220	3.87
245	5457	2743	3987	1493	3.23
307	5407	4103	4179	2204	3.15
310	6072	4121	4522	2215	3.40
346	9452	5300	6338	2822	3.99
361	6217	4854	4773	2607	3.27
419	8763	5730	6249	3074	3.17
492	9166	5859	6706	3175	3.49
503	16395	8617	10359	4559	3.15
512	11554	6542	7970	3526	3.94
592	13234	8181	9090	4386	3.20
607	12942	8718	8997	4662	3.91
758	13688	10105	9898	5431	3.42
869	18182	11807	12534	6337	3.20
878	14850	11964	10899	6420	3.14
918	18244	12106	12736	6511	3.14
992	24610	21800	16178	11395	3.10
1005	33967	13754	20902	7379	3.11
1007	16914	13757	12383	7381	3.15
1242	23158	16735	16327	8988	3.08
2680	67802	38657	43682	20668	3.09

---

* Number of full length integer words required when integers are packed two to a word or half-length integer words are used.

\+ In the new implementation the storage requirements for the Gibbs-Poole-Stockmeyer algorithm and the Gibbs-King algorithm may differ slightly; they are identically the same in all of these examples.


ATE
LMED
8