Another algorithm for reducing bandwidth and profile of a sparse matrix

by W. F. SMYTH
International Labor Office
Geneva, Switzerland

and

ILONA ARANY
Computing Center of the Ministry of Labor
Budapest, Hungary

ABSTRACT

The paper describes a new bandwidth reduction method for sparse matrices which promises to be both fast and effective in comparison with known methods. The algorithm operates on the undirected graph corresponding to the incidence matrix induced by the original sparse matrix, and separates into three distinct phases: (1) determination of a spanning tree of maximum length, (2) modification of the spanning tree into a free level structure of small width, (3) level-by-level numbering of the level structure. The final numbering produced corresponds to a renumbering of the rows and columns of a sparse matrix so as to concentrate non-zero elements of the matrix in a band about the main diagonal.

INTRODUCTION

As electronic computers make possible computations on ever larger data sets, it has come to be realized [1,p17,2] that most large matrices are, in the nature of things, sparse; more precisely, that a matrix of (large) order n will generally contain only Kn non-zero elements, where K tends to decrease as n increases (K is often as little as 2 or 3 and only rarely greater than 20). There has been, accordingly, during the last ten years, a good deal of research into techniques for efficient computer handling of large sparse matrices. These techniques may be separated into two classes:

- T_1: techniques which deal with the given sparse matrix, more or less directly, in a sparse form (some typical approaches are surveyed in [2] and [3]);
- T_2: techniques which transform the given sparse matrix into a band form (which may then be processed further by efficient and well-known band matrix algorithms).

Initially "direct" techniques T_1 attracted more interest, and the utility of "band" techniques T_2 was occasionally questioned.[4] Research into T_2 techniques was spurred however by the work of Cuthill-McKee,[5] who described an effective bandwidth reduction algorithm with execution time linear in Kn. Instead of dealing directly with the given matrix, Cuthill-McKee (hereafter called CM) dealt with the numbered graph whose connections correspond to the given matrix's zero/non-zero structure; they renumbered the vertices of this graph, and this renumbering therefore defined the interchanges of rows and columns required to transform the original sparse matrix into a band matrix.

The CM method was later modified in various ways, especially by Rose,[6] who also contributed an important analysis of the application of Gaussian elimination to band matrices, showing in particular the importance of the profile.

The algorithm presented here, called the SA algorithm, is of class T_2. It results from efforts to improve on previous work,[7] and has been directly stimulated by the approach suggested by Smyth-Benzi (hereafter called SB)[8] as well as by the related algorithm published independently by Gibbs-Poole-Stockmeyer (hereafter called GPS).[9] Like the CM and GPS algorithms, SA renumbers the vertices of a given numbered graph with the objective of minimizing the maximum difference between numbers assigned to connected vertices. Also in common with these algorithms, SA reduces profile as well as bandwidth[6] and may be applied to non-symmetric as well as to symmetric matrices.

TERMINOLOGY

We use the term *graph*, and the symbol G or G(V, E), to denote a finite connected * undirected graph without

* The algorithm may easily be modified to deal separately with the disjoint components of a single graph.

loops or multiple edges defined on a *vertex* set V of cardinality $n=|V|>1$ and an *edge* set E of cardinality $m=|E|$. For any distinct pair of vertices u, $v \epsilon V$, we define the usual *distance* function $\rho(u,v)$ to be the number of edges on the shortest *path* from u to v. For a single vertex u, we adopt the convention $\rho(u,u)=0$; for unconnected vertices u,v, we set $\rho(u,v)=\infty$; if $\rho(u,v)=1$, then u and v are said to be *adjacent*. The *diameter* of G is defined by

$$\text{diam}(G)=\max_{u,v\epsilon V}[\rho(u,v)].$$

Since we assume G is connected, $\text{diam}(G) \leq n-1$.

Apart from such basic terms, we need for our purposes here to define four main concepts: level structure, free level structure, numbering, and bandwidth.

A *level structure* LS of a graph G(V,E) is an assignment of the vertices of V into sets, called *levels* $L_1, L_2, \ldots, L_\lambda$, such that

(1) L_1 contains at least one vertex;
(2) for each $k=2,3,\ldots,\lambda$, L_k contains every vertex not in a previous level which is adjacent to some vertex of L_{k-1}.

It follows from this definition that if G is connected, LS contains all n vertices of V; that the levels are disjoint; that $1\leq\lambda\leq\text{diam}(G)+1$; and that LS is determined uniquely by the choice of vertices in L_1. We may therefore unambiguously denote LS by $LS(L_1)$ or, when $L_1=\{u\}$, by LS(u). We observe in fact that level L_k of $LS(L_1)$ consists of exactly those vertices which occur in the k^{th} level of every *spanning forest* (SF) grown from the vertex set L_1 (for a description of this process see Reference 5). $LS(L_1)$ therefore corresponds (one-many) to the $SF(L_1)$, and LS(u) to the *spanning trees* ST(u). Whenever edges are not important, then, we may refer to LS and SF/ST interchangeably.*

A *free level structure* FLS of a graph G(V,E) is an arrangement of all n vertices of V into λ levels $L_1, L_2, \ldots, L_\lambda$, such that

(1) no level is empty;
(2) if $u\epsilon L_k$ then all vertices adjacent to u are in either L_{k-1}, L_k, or L_{k+1}.

We note that in this case also $1\leq\lambda\leq\text{diam}(G)+1$, but that FLS(u) is no longer uniquely determined (Figure 1). For either LS or FLS we speak of the *width of level k*, $w(L_k)=|L_k|$, and the *width of the structure*, w(LS) or $w(FLS)=\max_{1\leq k\leq\lambda} w(L_k)$. λ is called the *length* of the structure.

Following GPS, we now define a *numbering* $\alpha=\alpha(V)$ of G(V,E) to be a one-one map of V onto the first n natural numbers $\{1,2,\ldots,n\}$. For a given numbering

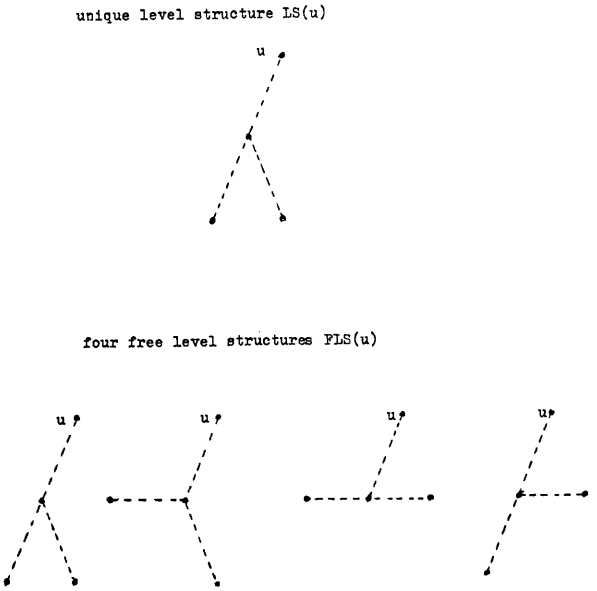

Figure 1

α, we may define $\delta_\alpha(G)$, *the bandwidth of the graph G relative to the numbering* α, to be

$$\delta_\alpha=\delta_\alpha(G)=\max_{(u,v)\epsilon E}|\alpha(u)-\alpha(v)|.$$

The *bandwidth of G* is then

$$\delta=\delta(G)=\min_{\text{all}\alpha}[\delta_\alpha(G)].$$

GENERAL DESCRIPTION OF THE SA ALGORITHM

Our goal is therefore, given G(V,E), to find a numbering $\alpha=\alpha(V)$ such that $\delta_\alpha=\delta(G)$. Our (heuristic) algorithm approaches this problem in three distinct phases:

I *Find LS*
Using the SB method,[8] a level structure LS $=LS(u)$ is found such that $\lambda=\text{diam}(G)+1$.

II *Find FLS*
Using a subset M ($|M|=\min\{|L_\lambda|, |\bar{n}/\bar{\lambda}|\}$) of the vertices of level L_λ in LS(u), a new level structure, denoted $LS'=LS'(M)$, is grown. LS′ is also of length λ. LS and LS′ are systematically compared, and an FLS is determined such that $w=w(FLS)$ is small (if possible, $w=|\bar{n}/\bar{\lambda}|$).

III *Number FLS*
FLS is numbered on a level-by-level basis; that is, first, the integers $\{1,2,\ldots,w_1\}$ are assigned to the vertices of the first level of FLS (w_k denotes the width of the k^{th} level of FLS);

* We dwell on this point to avoid confusion. The definition of LS given here is more restrictive than the original definition given in Reference 7, but is compatible with the SF/ST usage of Reference 8. The GPS definition of LS Reference 9 is compatible with Reference 7, and what GPS call the "level structure rooted at u" is identical to our LS(u).

second, the integers $\{w_1+1, w_1+2, \ldots, w_1+w_2\}$ are assigned to the vertices of the second level; and so on, until all vertices have been numbered. The algorithm makes use of knowledge of the edges joining vertices of successive levels in an effort to minimize bandwidth: it searches for a level-by-level numbering α such that the corresponding bandwidth satisfies

$$\delta_\alpha \leq w+\Delta,$$

where Δ successively takes the values $0,1, \ldots, w-1$. The first numbering found which satisfies this condition is the required numbering α.

The justification of algorithms such as this is partly theoretical, partly experimental, and numerous variations in strategy are possible. In practical terms, we are trying to find a "reasonably good" numbering without needing to investigate all of the n! different possible numberings; our strategy therefore is always influenced by estimates, often very rough in nature, of the additional benefit to be expected from the additional effort expended. We will find an example of this kind of strategic thinking in Phase II of SA: we do not carry out an exhaustive search to find the FLS of truly minimum width, even though such a search might not on the average be too laborious, simply because (1) it seems that an exhaustive search would not be likely to decrease w; (2) the result might be merely to decrease w by 1, but not decrease δ_α, and in addition make numbering more lengthy and difficult. On the other hand, in Phase I, we propose using the SB algorithm [8] instead of the GPS pseudo-diameter algorithm,[9] because the former guarantees finding a longest spanning tree at (apparently) no additional cost. On the theoretical side, the basic justification for level-by-level numbering is the result of Arany-Szóda [10] that corresponding to every numbering α of G there exists at least one FLS whose width $w=\delta_\alpha$, and which may be numbered on a level-by-level basis to yield the numbering α. Since the case $\delta_\alpha=\delta$ is included in this result, it follows that level-by-level numbering of an FLS is an acceptable approach to bandwidth reduction, in that it does not exclude any minimum case. The result however does not point to any particular FLS-growing or FLS-numbering algorithms.

PHASE I OF SA: FIND LS

As noted above, the SB diameter algorithm is proposed here, because it guarantees finding $\lambda=\text{diam}(G)+1$ and apparently is comparable to the GPS pseudo-diameter algorithm in execution time. The SB algorithm is described fully in Reference 8 and is not included here*.

* We do however provide a correction to the algorithm SPAN (h,i) given in the Appendix of Reference 8. In step 7 replace the ending "." with "goto11.". Step 8 should read "[Is the vertex connected to jorig in level g+1?] *If h=1* and level (h, elist(j))=g+1, goto10; otherwise goto11.". Interchange steps 9 and 10.

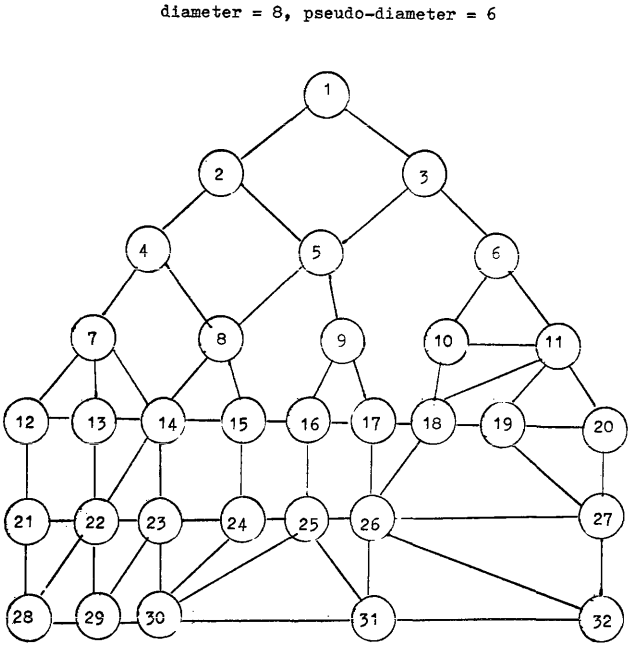

Figure 2

GPS state in Reference 9 that their pseudo-diameter algorithm found the true diameter in all test cases. Figure 2 illustrates a graph of diameter 8 whose GPS pseudo-diameter is 6. The GPS algorithm would require the growth of 6 spanning trees to achieve this result, the SB algorithm 3 spanning trees (starting vertex ①). Removal of edges (4,8) and (5,9) from Figure 2 and insertion of edge (4,5) would permit GPS to find a pseudo-diameter 7 at a cost of growing 4 spanning trees. For a symmetrized version of the Curtis matrix[4] (Figure 3), SB finds diameter 7 at a cost of 6 spanning trees (starting vertex ①), and GPS finds pseudo-diameter 7 from each of six starting vertices of minimum degree at an average cost of 5.5 spanning trees. See Table I.

A more detailed analysis of the application of the GPS pseudo-diameter algorithm to the Curtis matrix

TABLE I—Comparison of SB and GPS

	(pseudo-) diameter		number of LS grown	
example	SB	GPS	SB	GPS
Figure 2	8	6	3	6
Figure 2 (mod.)	8	7	3	4
Curtis matrix	7	7	6	5.5

FLS corresponding to Curtis matrix

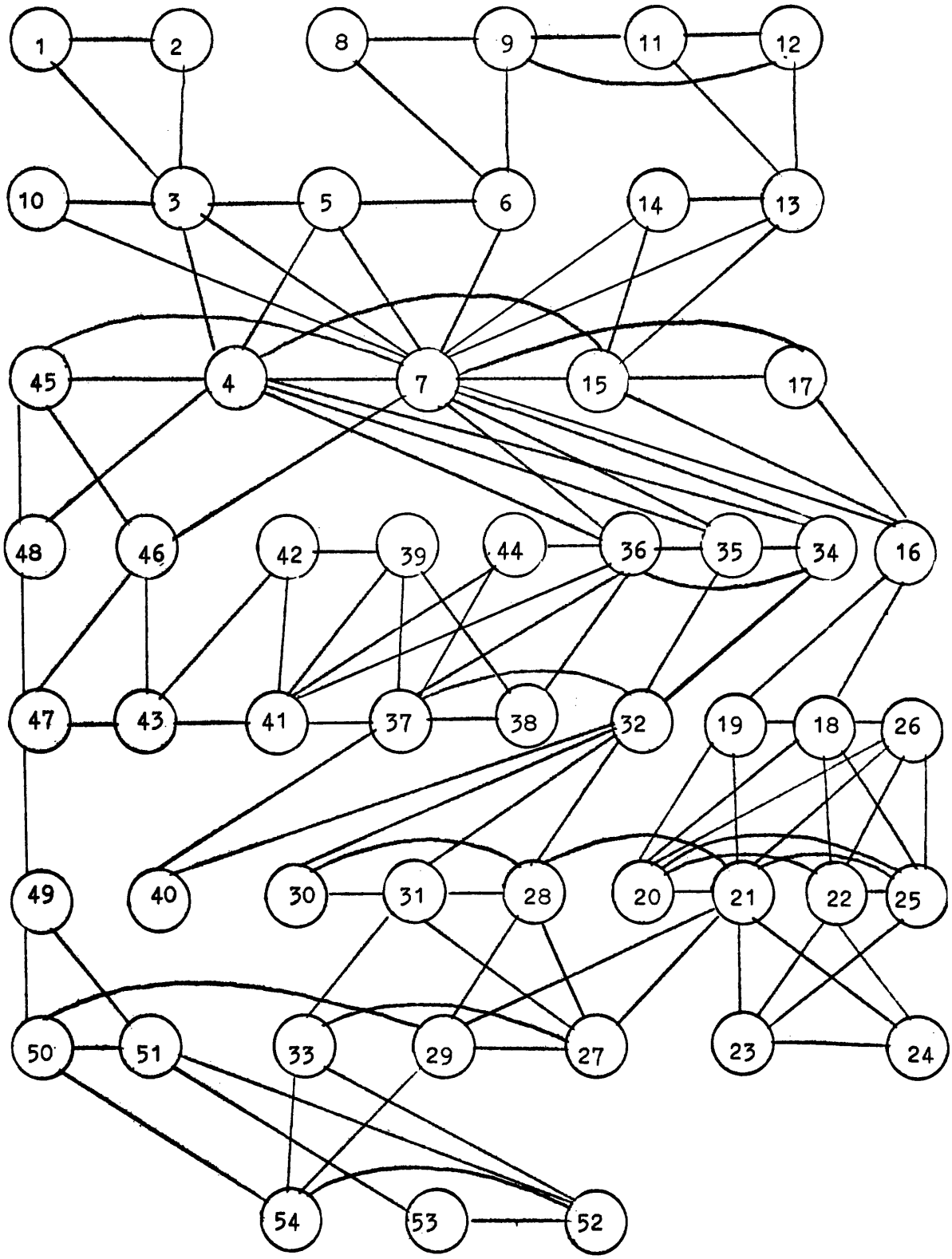

Figure 3

shows that 48 of the 54 vertices would, used as starting root vertices, yield maximum pseudo-diameter 7; of these 48 vertices, the 9 extreme vertices yield $\lambda=8$ immediately, 22 yield $\lambda=7$ but contain an extreme vertex in L_7, and 15 yield $\lambda=6$ with an extreme vertex in L_6 (there are two other special cases). These statistics appear to be related to the success of the GPS algorithm, and it would be interesting to know if for much larger graphs also (n=500 to 5000) such a high percentage of optimal starting root vertices occurs.

GPS initially searches V for a vertex of lowest degree as starting root vertex, and from the corresponding L_λ selects additional root vertices in increasing order of degree. SB includes no such searches.

PHASE II OF SA: FIND FLS

The algorithm described below is an attempt to deal efficiently with difficult cases which may arise when G is not "well" connected (Figure 4); that is, when a number of vertices swing into different levels in the level structure LS′ grown from vertices of L_λ in LS(u). It seems that the growing of this "reverse" level structure (step (1) of the following algorithm) will normally accomplish what GPS accomplish in phase II of their algorithm; the other steps in the algorithm given below make use of the same information proposed by GPS, but in a more flexible manner.

Given LS(u) of length $\lambda=\mathrm{diam}(G)+1$, phase II proceeds as follows:

(1) [Grow "reverse" LS′.] For $R=\lfloor \bar{n}/\bar{\lambda} \rfloor$,* choose a set of vertices $M<L_\lambda$ such that $|M|=\min [R, w(L_\lambda)]$ and grow LS′(M) with levels L_k' of width $w'(k)$, $k=1,2, \ldots, \lambda$ ($L_1'=M$).

(2) [Calculate initial value of T.] $T\leftarrow\Sigma_{1\leq k\leq\lambda} \min [R-w'(k),0]$; if T=0, goto (16). [Two criteria are used to determine whether the movement of a vertex of FLS(M) from one level to another yields a "better" FLS: the value T given here and the value $T'=\Sigma[R-w'(k)]^2$. The FLS is "better" if the vertex movement increases T (by unity) or decreases T′. T is also an "absolute" criterion, since any FLS for which T=0 is accepted; T′ is not calculated explicitly because it is not "absolute", so that only ΔT′ is required (see step (12)). Both T and T′ may be used since T′ decreases iff T does not increase.]

(3) [Calculate level pairs.] As in Reference 9, associate with each vertex $v \in V$ a *level pair* (g_v,h_v), where g_v is the level of v in LS′(M), where $h_v=\lambda-k_v+1$, and k_v is the level of v in LS(u).

(4) [Separate "movable" vertices into r connected components.] As in Reference 9, separate all vertices such that $g_v=h_v$ into r disjoint connected components $C_1,C_2, \ldots, C_r$ of cardinality $c_1,c_2, \ldots, c_r$, respectively, arranged so that $c_1\leq c_2\leq \ldots \leq c_r$.

(5) If r=0, goto (16).

(6) [Organize storage for vertices in component C_r.] Associate with each vertex $x_{rj}\in C_r$, $j=1,2, \ldots, c_r$, an *identifier* [v(j),g(j),h(j)], where v(j) is a pointer to the level pair of the vertex $v=x_{rj}$, and [g(j),h(j)] is the level pair of x_{rj} (that is, $g(j)=g_{v(j)}$, $h(j)=h_{v(j)}$); order the vertices so that $g(1)\geq g(2)\geq \ldots \geq g(c_r)$.

(7) [Initialize vertex movement parameters for C_r.] $j\leftarrow 1$; $\Delta T\leftarrow 0$, $\Delta T'\leftarrow 0$; for each $k=1,2, \ldots, \lambda$, $w(k)\leftarrow w'(k)$.
[For each C_r, we need to provide temporary storage for the level widths w′(k): this temporary storage into w(k) is used to keep track of moves made since the last acceptance of a vertex movement (last increase of T or decrease of T′).]

(8) [Move vertex x_{rj} from level g(j) to g(j)−1.] Perform MOVE-VERTEX(j,γ) as follows:
(8.1) $\gamma\leftarrow g(j)\leftarrow g(j)-1$;
(8.2) $w(\gamma)\leftarrow w(\gamma)+1$, $w(\gamma+1)\leftarrow w(\gamma+1)-1$;
(8.3) let j′ successively take the values $j+1, \ldots, c_r, 1, \ldots, j$; for the first value j′ such that $g(j')>h(j')$, set $j\leftarrow j'$;
(8.4) if there is no such j′, set $j\leftarrow 0$.
[Given j, MOVE-VERTEX(j,γ) returns the new level γ of the moved vertex x_{rj}, together with the next admissible value of j. The vertex movements take place in a "cascade" from higher-level vertices to lower-level vertices, in accordance with the observation that under these circumstances a vertex x_{rj} may always be moved to the preceding level, provided that $g(j)>h(j)$ (initially, of course, $g(j)>h(j)$ for every $j=1,2, \ldots, c_r$). Note that not all possible configurations of C_r are necessarily covered by the movements included here.]

(9) [Calculate ΔT.] $\Delta T\leftarrow\Delta T+$ (if $w(\gamma+1)>R-1$, 1; otherwise, 0) − (if $w(\gamma)>R$, 1; otherwise 0).
[The full expression for ΔT is
$$T=-\min\{R-[w(\gamma)-1],0\}+\min\{R-w(\gamma),0\}-\min\{R-[w(\gamma+1)+1],0\}+\min\{R-w(\gamma+1),0\}.]$$

(10) [For increased T, store present arrangement of vertices into levels.] If $\Delta T=1$, $T\leftarrow T+1$, $\Delta T\leftarrow 0$, and perform STORE-PATTERN; otherwise, goto (12).
[STORE-PATTERN does the following: for $j=1,2, \ldots, c_r$, $g_{v(j)}\leftarrow g(j)$; for $k=1,2, \ldots, \lambda$, $w'(k)\leftarrow w(k)$; $\Delta T'\leftarrow 0$. Note that ΔT′ is reset after every increase of T, but that acceptance based on ΔT′ does not reset ΔT (step (13)).]

(11) If T=0, goto (16).

(12) [Calculate ΔT′.] $\Delta T'\leftarrow\Delta T'+[w(\gamma)-w(\gamma+1)-1]$.

(13) If $\Delta T'<0$, perform STORE-PATTERN.

* SB propose rather $\lfloor \bar{n}/\bar{\lambda} \rfloor$.

(14) [More vertex movements possible?] If $j>0$, goto (8).

(15) [More components of "movable" vertices?] $r \leftarrow r-1$, goto (5).

(16) Exit.
[The FLS(M) which best satisfies the T and T' criteria has now been determined. Each vertex v of FLS(M) is placed into the level specified by g_v, and the width of each level L_k' is given by $w'(k)$.]

Figure 4 illustrates the result of the application of the algorithm to an LS with many movable vertices. In this case only three vertices would need to be moved before FLS became acceptable ($T=0$). In the example of Figure 2, an FLS of width 5 would be accepted ($T=-3$) after movement of two vertices. For the two dimensional grid discussed in detail by GPS, phase II yields an optimal FLS=LS(M) after step (1), and no vertex movements are required. For the Curtis matrix, on the other hand, consideration of the connected components in decreasing order of size actually inhibits bandwidth reduction: for starting vertex u=① and a corresponding LS(M) of width 11 ($T=-9$), the algorithm requires 56 vertex movements to yield an FLS of width 10 ($T=-7$). Although this result leads to an optimum numbering ($\delta=10$), much unnecessary work is done. Clearly, in order to evaluate the efficiency and utility of phase II of the SA algorithm, considerable computational experience, especially with large matrices, is desirable.

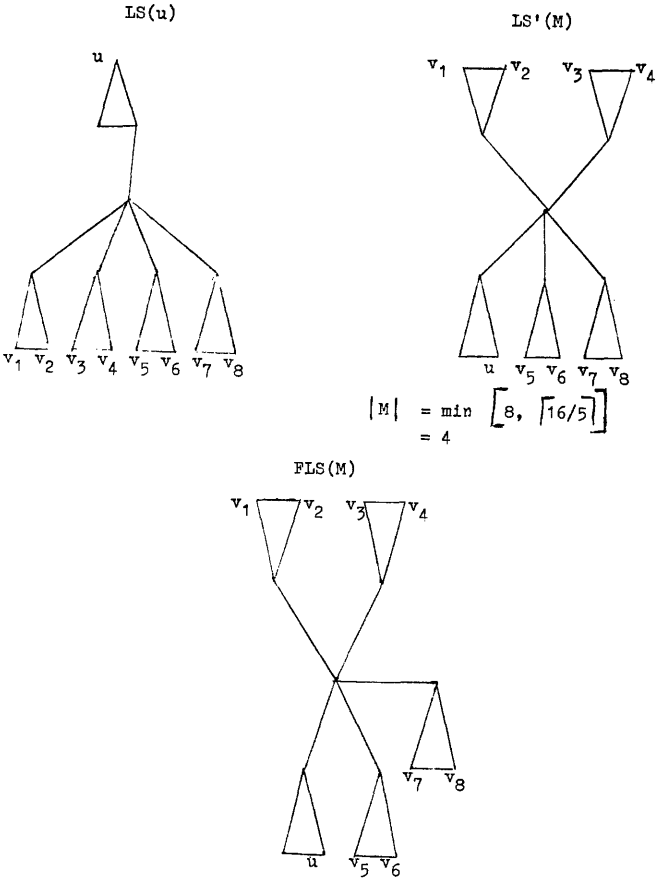

Figure 4

PHASE III OF SA: NUMBER FLS

We propose here a somewhat more sophisticated numbering algorithm than has previously been employed,[5,7,9] but which retains the property of having execution time approximately linear in n. Indeed, as we shall see, the numbering algorithm has the interesting property of being rather more efficient in the more difficult cases.

Suppose we are given an FLS(M) of length λ and width w', with levels L_k' of width $w'(k)$, $k=1,2,\ldots,\lambda$. Suppose further that corresponding to each $v \in V$, we may identify g_v, the level of v, and A(v), the set of vertices adjacent to v; and that corresponding to each level L_k' of FLS, we may identify the vertices $x_{kj} \in L_k'$, $j=1,2,\ldots,w'(k)$. Phase III assigns numbers to the vertices on a level-by-level basis, with the objective of arriving at a numbering α of G which corresponds to a suitably small bandwidth δ_α. In the large, phase III proceeds as follows:

(1) $\Delta \leftarrow 0$.

(2) $\delta_\alpha \leftarrow w' + \Delta$.

(3) For each $k=1,2,\ldots,\lambda$, try to assign numbers to the vertices $x_{kj} \in L_k'$ in such a manner that the *number* n(x) of each vertex x satisfies

$$|n(x)-n(y)| \leq \delta_\alpha,$$

for every $y \in A_x$.

(4) If for some k, it turns out that the vertices cannot be numbered to satisfy this condition, then $\Delta \leftarrow \Delta+1$, goto (2).

We observe that, in principle, this algorithm will terminate sooner or later; indeed, as CM remark,[5] for $\delta_\alpha = 2w'-1$, *any* level-by-level numbering of FLS will satisfy the condition given in step (3). We observe further that, in practice, the algorithm will normally terminate at some value δ_α close to w'; in fact, in the great majority of cases, for $\delta_\alpha = w'$.*

We give now a more detailed description of step (3). Since the same procedure is used for the numbering of each level, we confine ourselves to describing the numbering of the k^{th} level L_k'. Certain basic values need to be defined and calculated:

$W'(k) = \Sigma_{1 \leq k' \leq k} w'(k')$, $1 \leq k \leq \lambda$;
$= 0$, $k=0$.

$A_{\pm 1}(j) = \{v \mid x_{kj} \in L'_k \wedge v \in A(x_{kj}) \wedge v \in L_{k\pm 1}'\}$, the set of vertices of L'_{k+1} (L'_{k-1}) adjacent to the j^{th} vertex in L'_k.

* However an unpublished example due to Arany-Szóda has the following characteristics: $n=60$, $m=112$, $w'=9$, $\delta(G)=14$.

$a_{\pm 1}(j) = |A_{\pm 1}(j)|$.

$lown(j) = \min_{x \in A_{-1}(j)} n(x), a_{-1}(x) > 0;$
$= W'(k-1), a_{-1}(x) = 0;$
the lowest number assigned to any vertex in L_{k-1}' which is adjacent to the j^{th} vertex in L_k'.

The processing required for each level falls naturally into two stages, which we call INITIALIZE L_k' and NUMBER L_k':

INITIALIZE L_k'

(1) For each vertex x_{kj}, $j=1,2,\ldots, w'(k)$, calculate
(1.1) $xmin(j) \leftarrow \max[1+W'(k-1), W'(k)+a_{+1}(j)-\delta_\alpha]$, the least number assignable to x_{kj} which is compatible with bandwidth δ_α;
(1.2) $xmax(j) \leftarrow \min[W'(k), \delta_\alpha + lown(j)]$, the greatest number assignable to x_{kj} which is compatible with δ_α;
(1.3) $xrange(j) \leftarrow xmax(j) - xmin(j) + 1$; if $xrange(j) \leq 0$, goto INCREMENT Δ (step (4) in the general outline of phase III given above).

(2) $nrangeclass(j) \leftarrow 0$, $j=1,2,\ldots, w'(k)$.
[The number of vertices to which a number $W'(k-1)+j'$, $j'=1,2,\ldots, w'(k)$, may be assigned will be determined; nrangeclass(j) will then contain a pointer to the first of the numbers $W'(k-1)+j'$ which may be assigned to exactly j vertices.]

(3) For each number $j'=1,2,\ldots, w'(k)$:
(3.1) calculate
$nr(j')$, the number of values of j for which $xmin(j) \leq j' \leq xmax(j)$;
$nlist(j',1)$ to $nlist(j',nr(j'))$, a list containing the values of j for which $xmin(j) \leq j' \leq xmax(j)$;
(3.2) if $nr(j') = 0$, goto INCREMENT Δ;
(3.3) $np(j') \leftarrow nrangeclass(nr(j'))$, $nrangeclass(nr(j')) \leftarrow j'$.
[$np(j')$ is a pointer to the next number in $nrangeclass(nr(j'))$.]

NUMBER L_k'

(4) $J \leftarrow 1$, $counter \leftarrow 0$.
(5) If $nrangeclass(J) = 0$, $J \leftarrow J+1$, goto (5).
[The numbers to be assigned are chosen in increasing order of nrangeclass.]
(6) $j' \leftarrow nrangeclass(J)$, $nrangeclass(J) \leftarrow np(j')$.
[j' is the number to be assigned.]
(7) [Determine j, the vertex to which j' is assigned.]
(7.1) $h \leftarrow 1$, $j \leftarrow nlist(j',h)$;
(7.2) $h \leftarrow h+1$, $j_1 \leftarrow nlist(j',h)$; if $j_1 = 0$, goto (7.6);
(7.3) if $range(j_1) < xrange(j)$, $xrange(j) \leftarrow xrange(j) - 1$, $j \leftarrow j_1$, goto (7.2);
(7.4) $xrange(j_1) \leftarrow xrange(j_1) - 1$; if $xrange(j_1) = 0$, goto INCREMENT Δ;
(7.5) goto (7.2);
(7.6) exit.
(8) [Assign j' to j.]
(8.1) $n(x_{kj}) \leftarrow W'(k-1)+j'$;
(8.2) $counter \leftarrow counter+1$; if $counter = w'(k)$, goto (11);
(8.3) $xrange(j) \leftarrow \infty$, $nr(j') \leftarrow 0$.
(9) [Delete j and j' from storage.] For every $J' = xmin(j), xmin(j)+, \ldots, xmax(j)$ such that $nr(J') = 0$, do the following:
(9.1) $rold \leftarrow nr(J')$, $nr(J') \leftarrow nr(J') - 1$;
(9.2) if $rold = 1$, goto INCREMENT Δ;
if $rold = J$, $J \leftarrow nr(J')$;
(9.3) $pold \leftarrow np(J')$, $np(J') \leftarrow nrangeclass(nr(J'))$, $nrangeclass(r(J')) \leftarrow J'$;
(9.4) if $nrangeclass(rold) = J'$, $nrangeclass(rold) \leftarrow pold$, goto (9.7); otherwise, $j_2 \leftarrow nrangeclass(rold)$;
(9.5) if $np(j_2) = J'$, $j_2 \leftarrow np(j_2)$, goto (9.5);
(9.6) $np(j_2) \leftarrow pold$;
(9.7) exit.
(10) Goto (5).
(11) Exit.

Table II displays results obtained using the above algorithm on a few examples; these results [δ(SA)] are compared with optimum numbering [δ(FLS)] and with results [δ(GPS)] obtained by the GPS numbering algorithm. δ(FLS) was attained neither by SA nor GPS, primarily because more than single-level lookahead was required.

It appears that, in phase III as in the other phases, SA produces results at least as satisfactory as those of GPS. The execution time of phase III is bounded above by a value proportional to $\Sigma w'(k)^2$, and depends essentially on the size of the variables xrange(j) and nr(j'): when these variables are small—that is, when the numbering is more difficult—execution time will correspondingly be small. Storage required for phase III is of the order of $w'(w'+6)$.

CONCLUSIONS

We have described a bandwidth reduction algorithm which appears to be competitive in effectiveness and efficiency with presently known algorithm. Systematic testing on large matrices encountered in practice is re-

TABLE II—Numbering FLS using SA

FLS	δ(SA)	δ(FLS)	δ(GPS)
Figure 2	6	5	6
Figure 3	11	10	11
FLS of width 10 determined by applying phase II to Curtis matrix $[L_1' = \{52,53,54\}]$	11	10	12
Figure 4	5	4	6

quired. Some of the more important questions remaining to be clarified, either by experiment or by analysis, are as follows:

(1) The execution time of SB (phase I of SA) is proportional to 2Hm. What can be said about the magnitude of H?
(2) In what (more efficient) way can vertex movements be evaluated during phase II, in order to yield an optimum FLS?
(3) Given two FLS of G of widths w_1 and $w_2 > w_1$, denote by δ_1 and δ_2 the least bandwidths obtainable by level-by-level numbering of the first and second FLS, respectively. Does it follow that $\delta_2 \geq \delta_1$?

REFERENCES

1. Willoughby, Ralph A., *Sparse Matrix Algorithms and Their Relation to Problem Classes and Computer Architecture*, IBM Research Publication RC 2833, March 1970, 38 pp.
2. Pooch, Udo W. and Al Nieder, "A Survey of Indexing Techniques for Sparse Matrices," *ACM Computing Surveys* 5-2, June 1973, pp. 109-133.
3. Tewarson, R. P., "Computations with Sparse Matrices," *SIAM Review*, 12-4, October 1970, pp. 527-543.
4. Curtis, A. R. and J. K. Reid, "The Solution of Large Sparse Unsymmetric Systems of Linear Equations," *Proc. IFIP Congress 71*, 1972, pp. 1240-1245.
5. Cuthill, E. and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. 24th Nat. Conf. ACM*, 1969, pp. 157-172.
6. Rose, D. J., *Symmetric Elimination on Sparse Positive Definite Systems and the Potential Flow Network Problem.* Ph.D. thesis, Harvard Univ., 1972.
7. Arany, Ilona, W. F. Smyth and Lajos Szóda, "An Improved Method for Reducing the Bandwidth of Sparse Symmetric Matrices," *Proc. IFIP Congress 71*, 1972, pp. 1246-1250.
8. Smyth, W. F. and W. M. L. Benzi, "An Algorithm for Finding the Diameter of a Graph," *Proc. IFIP Congress 74*, 1975, pp. 500-503.
9. Gibbs, Norman E., William G. Poole and Paul K. Stǫckmeyer, *An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix*, Technical Report No. 5, July 1974, 25 pp.
10. Arany, Ilona and Lajos Szóda, "Ritka Szimmetrikus Matrixok Sávszélesség Redukciója, "*Információ Elektronika* 4, 1973, pp. 273-282.