A NEW BANDWIDTH REDUCTION ALGORITHM FOR TREES

Chandra GowriSankaran, Dawson College, Montreal.
Zevi Miller, Miami University, Oxford, Ohio.
Jaroslav Opatrny, Concordia University, Montreal.

Keywords: Bandwidth of graphs, Bandwidth reduction algorithm, Level structure algorithm.

Abstract:

The most successful bandwidth reduction algorithms for graphs are level structure algorithms. This paper studies a new bandwidth reduction algorithm for trees, algorithm LST, which defines recursively a level structure for trees. Some theoretical properties and the time complexity of algorithm LST are discussed. Furthermore, we present the results of empirical studies, in particular comparing LST with the most successful bandwidth reduction algorithm to date, the GPS algorithm. It is shown that in almost all examples of trees studied, algorithm LST produced level structures of smaller widths than did GPS algorithm.

1. INTRODUCTION

We refer to [1] for basic terminology in graph theory.

Let G(V,E) be a graph with vertex set V and edge set E. Assume $|V| = n$. A one-to-one mapping f from V to the set $\{1,2,...,n\}$ is called a **labelling** of G. The **bandwidth of a labelling** f of G, denoted $B_f(G)$, is defined as

$$B_f(G) = \max \{|f(u)-f(v)| : (u,v) \text{ is in } E\}$$

and the **bandwidth** of G, denoted **B(G)**, is defined as

$$B(G) = \min \{B_f(G): f \text{ is a labelling of } G\}$$

A labelling f of G is called a **bandwidth labelling** if

$$B_f(G) = B(G)$$

Thus to find the **bandwidth of a graph** G, is to find a labelling f of G such that $B_f(G)$ is minimum.

A survey of the developments in the bandwidth problem for graphs and matrices up until 1981 is found in [2]. In 1976, Papadimitriou [14] proved that the bandwidth problem is NP-complete. Garey, Graham, Johnson and Knuth [6] further sharpened this result by proving that the bandwidth problem remains NP-complete even for trees with maximum degree 3.

In view of the NP-completeness of the bandwidth problem one does not expect to find an exact polynomial time solution to the general bandwidth problem. On the other hand, by

reducing as much as possible the width b of the band around the main diagonal which contains all the nonzero entries in an adjacency matrix of a graph, the time efficiency of any computation on this graph can be improved. Therefore, an alternative approach to the bandwidth problem is to try to find a time-efficient algorithm that gives a "good" approximation to the bandwidth of a graph. Such an algorithm should relabel the graph with a view to reduce as much as possible the bandwidth of the labelling. We will call such an algorithm a **bandwidth reduction algorithm**. These algorithms are of interest in practical applications. A brief survey of bandwidth reduction algorithms is found in [2].

One of the most successful bandwidth reduction algorithms was introduced in 1969 by Cuthill and McKee [5]. This will be referred to as algorithm **CUM**. This algorithm introduced the notion of a "level structure labelling" (defined in section 2) of a graph. The algorithm described by Gibbs, Poole and Stockmeyer in 1976 [8], is closely related to and an improvement over the algorithm CUM. It will be referred to as **GPS** algorithm.

Both CUM and GPS algorithms have been empirically proved to be far more successful in reducing bandwidths than any of their predecessors [5], [9]. This conclusion is theoretically justified by Turner [15]. Turner studies the performance of heuristic bandwidth minimization algorithms on random graphs. He shows that for an appropriate probability distribution on the space of n-vertex graphs with bandwidth less than or equal to k, suitably modified level structure algorithms produce a good bandwidth approximation for almost all graphs.

Although very successful when applied to gridded rectangles and cylinders, the GPS algorithm was observed to perform very poorly when applied to certain families of trees (see example 4.1). This seems to indicate that a new algorithm is necessary for approximations of bandwidths of trees in general. In view of Turner's result, it seems probable that an algorithm based on a suitably modified level structure of GPS will result in improved bandwidth reduction

for trees. In this paper, we present two versions of a new algorithm to compute a **level structure of a tree**, called algorithm **LST1**, We demonstrate that for trees, this algorithm finds, in most cases, level structures of smaller widths than those produced by the GPS algorithm. The new algorithm should, therefore, yield better approximations to tree bandwidths.

2. NEW LEVEL STRUCTURE ALGORITHMS FOR TREES

We need the following terminology in order to discuss bandwidth reduction algorithms.

Definitions. Let $G(V,E)$ be a **connected graph**. A **Level Structure** of G, denoted by **L(G)** or L, is a partition of the vertex set $V(G)$ into sets N_1, N_2, ..., N_k, called **levels**, which satisfy the following condition:

For $1 \le i \le k$, all vertices adjacent to vertices in N_i are in $N_{i-1} \cup N_i \cup N_{i+1}$, where $N_0 = N_{k+1} = \emptyset$.

In particular, if v is any vertex in V, the **level structure rooted at** v, denoted by $L_v(G)$ or L_v, is the level structure for which

$$N_i = \{ u \text{ in } V(G): d(u,v) = i-1 \}.$$

The **width of level structure** L, denoted by $w(L)$, is defined as

$$\max \{ |N_i| : 1 \le i \le k \}$$

The **depth of level structure** L is k, the number of levels in L.

Let $L(G)$ be a level structure on G. Let f be a labelling of G which labels arbitrarily the vertices in V level by level, in the sense that for $0<i<k$, if u is in N_i and w is in N_{i+1}, then $f(u)<f(w)$. Such a labelling will be called a **level structure labelling** and a bandwidth reduction algorithm which labels G with a level structure labelling will be called a **level structure algorithm.** Clearly, if f is a level structure labelling of a graph G with level structure L, then

$$B(G) \le B_f(G) \le 2w(L) - 1.$$

Moreover, if L is a rooted level structure $L_v(G)$, then

$$B_f(G) \ge w(L_v).$$

Let $d(x,y)$ denote the distance between vertices x and y in G. Then the **diameter** D of G is given by

$$D = \max \{ d(x,y) \mid x, y \text{ are in } G \}$$

Any (x,y)-path such that $d(x,y) = D$ is called a **diameter path.**

Let T be a tree and $P = (p_1, p_2, \ldots p_k)$ a diameter path in T. **A diameter level structure** of T is the level structure

$$N = \{ L_1, L_2, \ldots, L_k \}$$

where for each i, $1 \leq i \leq k$, $L_i = \{ p_i \}$.

T-P is the graph obtained by deleting from T all the vertices in P and all the edges incident on these vertices.

Any connected component of T-P will be called an **off-diameter subtree.**

Let C be a subtree and x a vertex not in C. Then **x is adjacent to C** if $d(x,C) = 1$.

The reverse level structure L^R: Let $L = \{N_1, N_2, \ldots. N_m\}$ be a level structure. Then the reverse of level structure L is $L^R = \{N^R_1, N^R_2, \ldots. N^R_m\}$ where for all r, $N^R_r = N_{m+1-r}$.

The CUM algorithm uses a rooted level structure while the GPS algorithm uses a more general kind of level structure. Since in this paper we are comparing our new algorithm LST1 to algorithm GPS, we will give a short outline of the latter algorithm.

The GPS algorithm consists of the following three phases:

Phase I. Finding a pseudo diameter

Find the **endpoints** of a **pseudo diameter**, defined to be two vertices which are nearly **maximal** distance apart.

It can be easily proved that in case of a tree, this algorithm will, in fact, compute a diameter of the tree.

Phase II. Minimizing the level width

In phase II the two rooted level structures L_v and L_u, where v and u are the endpoints of a pseudo-diameter, are combined into a new level structure whose width is usually less than that of either of the original ones.

Phase III. Numbering

This phase uses the level structure L generated by the Phase II and assigns consecutive positive integers to the vertices

of G level by level, using a modified version of the CUM numbering algorithm.

Algorithm LST1 also computes the pseudo-diameter. It can be easily proved that, in case of a tree, this algorithm will in fact compute a diameter of the tree. However, phase II of GPS cannot work very well with trees or one-connected graphs. Let x be a vertex on a diameter D of a tree G and let x be in level N_i. Let C be a connected component of G-D, such that d(x,C) = 1. Let

$$r = \max \{ d(x,y) : y \text{ in } C \}.$$

Then the phase II of GPS algorithm will place all vertices in C either in $N_{i-1} \cup N_{i-2} \cup \ldots \cup N_{i-r}$,

or in $N_{i+1} \cup N_{i+2} \cup \ldots \cup N_{i+r}$, instead of spreading them in $N_{i-r} \cup \ldots\ldots \cup N_{i+r}$, and thus allowing a more uniform distribution of vertices in the level structure.

Algorithm LST1 remedies this situation by introducing a new, modified level structure for trees which exploits the special properties of trees and thus computes a level structure of smaller width than that produced by GPS. LST1 computes this level structure by a recursive procedure.

We first describe below the main steps in algorithm LST1.

Algorithm LST1.

Step 1. Given a tree T, compute a diameter path P of T. Create the diameter level structure of P.

Step 2. For every connected component C of T-P do

(A)I. If the size of C is less than 3, create a diameter level structure of "small subtree" as follows:

(a) assign levels 1 and 2 to the vertices of C if there are 2 vertices.

(b) if C has only 1 vertex, assign level 1 to it.

II. If size of C is 3 or more, apply the algorithm recursively and find a level structure of C.

(B) Merge the level structure of C into the diameter level structure of P.

A more detailed specification of LST1 using a Pascal-

like pseudo-code for algorithm LST1 is given below. We specify first the main procedures and functions of the algorithm.

Procedure **DIAMETER** (T, P, m):

For a given tree T, this procedure computes the diameter path P = { $u_1, u_2, \ldots u_m$ } and its length m. The algorithm used is a modification of the standard algorithm to find a pseudo-diameter of a general connected graph from phase I of the GPS algorithm.

```
procedure MERGE (L,L',u,j);
{Here L = {N1,N2,....Nm}  and L' = {N'1,N'2,....N'm'}  are
two level structures. The vertex u is in level k in L'.  The
two structures are merged in such a way that the level of u
in L is j. The merged structure is stored as L.}
begin
  k := index of the level of u in L';
  for every level class N'i in L' do
    begin
      {find the index of the level in L for vertices in N'i}
      t := j+i-k;
      for every vertex x in N'i do
          assign x to level t in L;
        {N'i gets merged into Nt}
    end;
update widths of level classes in L;
update width(L);
end; {procedure MERGE }
```

Function **WIDTHMRG** (L,L',u,j)

{This function computes the width of the level structure produced by MERGE (L,L',u,j) without actually computing the merged level structure. Algorithm LST1 uses this width as a criterion to choose the best possible merger out of a number of feasible alternatives.}

{All the parameters are as in procedure MERGE. Arrays widthl[1..m] and widthl'[1..m'] store the widths of the corresponding level classes of L and L' resp. We may assume that these widths are previously computed and saved whenever a level structure is generated newly or augmented by a merge operation. width(L) and width(L') are also assumed to be available }

```
begin
  maxw := 0;
  k := index of the level of u in L';
  for i:= 1 to m' do
    begin
      {find t where N'i gets merged into Nt}
      t := i+j-k;
      w := widthl[t] + widthl'[i];
{find the width of the level class t after the implied
merger. At the same time, find the maximum of the resulting
```

```
class widths}
      if (w > maxw ) then maxw := w;
    end;
{ find maximum of all level widths in resulting structure L}
 widthmrg := max (maxw, width(L));
end; {procedure widthmrg }
```

A pseudo-code for algorithm LST1 now follows.

```
Algorithm LEVEL_STRUCTURE (T,L,f)
{Computes recursively the level structure L of tree T. f is a
boolean variable which is true for T and false for a subtree}
          begin
{PART I :Compute a diameter path P = {u1,u2,...um} of T}
          DIAMETER (T, P, m);
            for i:= 1 to m do
               begin
{initialize the diameter level structure of P}
                Ni := {ui};
                 mark ui;
               end;

{PART II :Compute level structures of off-diameter subtrees.
If the size of the subtree is 3 or more this is done by a
recursive call to LEVEL_STRUCTURE. Merge these structures
into the diameter level structure of P}
          begin
            for i:= 2 to m-1 do
               for every unmarked neighbor x of ui do
                 begin
                    delete the edge (ui, x);
                    generate the component C of T-P
                    containing x;
                    If |V(C)| >= 3
                      then LEVEL_STRUCTURE (C,L',false)
                      else create level structure L'
                           of "small subtree" C;
                      for j := 1 to 3 do
{Compute the widths of the three merges of  L and L'}
                           sj := WIDTHMRG (L,L',x, i+j-2);
                      for j := 4 to 6 do
{Compute the widths of the three merges of   L  and L'^R}
                           sj := WIDTHMRG (L,L'^R,x,i+5-j);
                      s := min {sj : 1<=j<=6};
                      k := min {j : sj = s};
{determine the best merger rule: the one with minimum width
for merged structure; in case of a tie, choose the rule with
smallest index.}      if k <= 3 then
                              MERGE (L,L',x,i+k-2)
                         else MERGE (L,L'^R,x,i+5-k);
                 end;
          end;
```

The initial version of our algorithm, Algorithm LST, treats

"small subtrees", that is subtrees of size 1 or 2, in a different manner than the rest of the subtrees. In part II of algorithm LEVEL_STRUCTURE, LST finds a level structure of a subtree only if its size is 3 or more and processes all small subtrees at the very end in part III. Thus LST differs from LST1, which, irrespective of the size of an off-diameter subtree, finds its level structure and merges it into the "parent" diameter structure when it is first seen in part II. Accordingly, Step 2 in LST1 is replaced with Step 2 and Step 3 in the description of algorithm LST that follows. Problems encountered in LST in merging a level structure of a subtree C into a parent diameter structure of T before processing all "small subtrees" of C are explained in [10].

Algorithm LST

Step 1. Given a tree T, compute a diameter path P of T. Create the diameter level structure of P.

Step 2. For every connected component C of T-P, of size 3 or more do:
(a) Apply steps 1 and 2 of the algorithm recursively and find a level structure of C.
(b) Merge the level structure of C into the diameter level structure of P to obtain a level structure of T.

Step 3. Finally, merge all the components of size less than 3 into the level structure in such a way that the differences between the widths of consecutive levels are as small as possible.

3. TIME COMPLEXITY OF LST1

Let T be a tree on n vertices and let d be the maximum degree in T. We assume that T is sparse, meaning that the adjacency matrix of T is sparse. We assume that T is stored in one of the typical compact forms used for storing sparse graphs, **the adjacency table**, which is an n*d array organized as follows. If the vertex k has degree r, then the kth row of the adjacency table lists the r vertices adjacent to vertex k, followed by d-r zeroes.

The following lemmas are needed in deriving an upper bound on the time complexity of LST1. Detailed proofs are given in [10].

Lemma 3.1: Let T be a tree on n vertices with maximum degree d. Then a diameter level structure L of T is computed in time $O(d*n)$.

Lemma 3.2: Let T, n and d be as in lemma 3.1. Let k be the maximum depth of recursion attained by algorithm LST1 when applied to T. Then the time required to compute all the diameter level structures is $O(k*d*n)$.

Lemma 3.3: Let D be a diameter path in a tree T. Let T' be a connected component of T-D. Let D' be a diameter path in T'. Then $S(D) \geq S(D')+2$.

We can now state the following theorem.

Theorem 3.1: Let T be a tree on n vertices. Then in an application of algorithm LST1 to T, the depth of recursion cannot exceed $\sqrt{n}$.

Proof: Let $T_1 = T$ and let D_1 be the diameter of T_1 computed when algorithm LST1 is applied to T_1. Let k be the maximum depth of recursion. Then there is a sequence of subtrees $T=T_1, T_2, \ldots, T_k$, and a sequence of corresponding diameter paths $D=D_1, D_2, \ldots, D_k$, such that for all i, $2 \leq i \leq k$,

(1) T_i is a connected component of $T_{i-1}-D_{i-1}$,

(2) Algorithm LST1 computes D_i as a diameter path in T_i.

Since D_k has at least 1 vertex, by lemma 3.3 D_{k-1} has at least 3 vertices. Applying lemma 3.3 to all the preceding diameters in the sequence, we have

$$1 + 3 + \ldots + (2k-1) \leq S(D_k) + S(D_{k-1}) + \ldots + S(D_1) \leq n$$

Therefore, $k^2 \leq n$, and $k \leq \sqrt{n}$.

For each k, one can construct a tree on k^2 vertices for which an application of algorithm LST1 can result in recursive calls of depth k [10].

Combining Lemma 3.1 and Theorem 3.1 gives the following result.

Theorem 3.2: Let T be a tree on n vertices and let $\Delta(T)=d$. Then the time required for algorithm LST1 to compute all the diameter level structures in T is $O(d*n^{3/2})$.

The merge operation : When a level structure L' is merged into a level structure L, the following two steps are executed.

Step (1): The vertices in L' are assigned new level numbers.

Step (2): The widths of levels in L are modified with the additions of the widths of the merged classes of L'.

Since the depth of recursion cannot exceed $\sqrt{n}$, a vertex may be part of a merge operation at most $\sqrt{n}$ times and therefore may get relabelled at most $\sqrt{n}$ times. Therefore step (1) in all the merge operations can be executed in total time $O(n*\sqrt{n})$. Relabelling of vertices can be avoided by opting for more book-keeping and labelling the vertices only once, which in turn will reduce this time to O(n).

Next, we note that every level structure L' is merged only once in some level structure L of bigger length. The number of levels modified in step (2) above is equal to the length of L'. Since the lengths of all level structures generated during the application of the algorithm LST1 cannot exceed n, the time for modifying all the level widths during all the merge operations will be O(n). Thus the execution of the two steps above together will cost time $O(n^{3/2})$.

Combining the time taken for computing and merging all the level structures, we can state the following theorem.

> **Theorem 3.3:** Let T be a tree on n vertices and let $\Delta(T)=d$. Then the algorithm LST1 computes a level structure of T in time $O(d*n^{3/2})$.

4. PERFORMANCE OF LST1

We next show that algorithm LST1 can be combined with a labelling algorithm to obtain the bandwidth of a nontrivial family of trees for which the performance of the GPS algorithm is particularly poor.

Example 4.1. Bandwidth labelling of a nontrivial family of trees :Let T_1 and T_2 be as shown. For $n \geq 3$, define tree T_n recursively as follows:

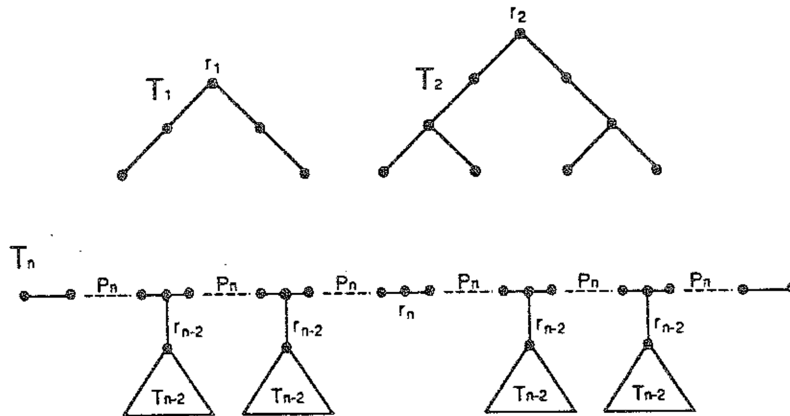

Figure 4.1

where for $n \geq 3$, P_n is a path of length $2*(\text{depth}(T_{n-2}))+4$.

Theorem 4.1: For every $n \geq 1$, algorithm LST1 constructs a level structure of T_n of width $\lceil (n+1)/2 \rceil$.

Proof: We prove the theorem by induction on n. Clearly, for n=1 and n=2 algorithm LST1 constructs a level structure T_1 and T_2 of width 1 and 2 respectively.

Assume that for $n \leq k$ algorithm LST1 creates a level structure of T_n of width $\lceil (n+1)/2 \rceil$. Consider now application of LST1 to T_k and let N be the level structure generated. Clearly, the path consisting of six P_k paths is the diameter of T_k and algorithm LST1 will place one vertex of the path in each level of N. Further, recursive application of algorithm LST1 will produce level structures N^1, N^2, N^3, N^4 for the four copies of T_{k-2}. By induction hypothesis,

$$\text{width}(N^i) \leq \lceil (k-2+1)/2 \rceil = \lceil (k-1)/2 \rceil.$$

Since the length of P_k is greater than $2*\text{depth}(T_{k-2})$, by merging N^1, N^2, N^3, N^4 into N, we obtain a level structure N such that

$$\begin{aligned} \text{Width (N)} &= 1 + \max \{\text{width } (N^i) \mid 1 \leq i \leq 4\} \\ &\leq 1 + \lceil (k-1)/2 \rceil = \lceil (k+1)/2 \rceil. \end{aligned}$$

One can further construct a labelling f such that

$$B_f(T_k) \leq \text{width (N)} = \lceil (k+1)/2 \rceil.$$

Since it can be proved that $B(T_k) \geq \lceil (k+1)/2 \rceil$, it follows that algorithm LST1 constructs optimum solutions for the family of trees T_n. Now for tree T_{n-2}, there exists i such

that there are $2^{n/2}$ vertices at distance i from r_{n-2}. Thus if algorithm GPS is applied to T_n, it will construct a level structure of width greater than or equal to $2^{n/2}+1$. Thus while LST1 creates a level structure of T_n with width O(n), the level structure created by GPS has width $O(2^{n/2})$ which clearly demonstrates the advantage of algorithm LST1 over algorithm GPS for such a family of trees.

As is the case with any heuristic algorithm, one can find examples in which the performance of algorithm LST1 is very poor. In some instances, a decision to choose a merger rule over another, with a view to keep to the minimum the width of the resulting level structure, may prove to result later in increased width. The following theorem makes a general statement in this direction.

Theorem 4.2: For every positive integer $k \geq 3$, there exists a tree G_k such that $B(G_k) = 3$ and the width of the level structure of G_k produced by algorithm LST1 is k+1.

Proof: For each $k \geq 3$ one can construct a tree G_k as shown in Figure 3.10. G_k is a caterpillar. The diameter path in G_k is (u,v). The centre of the diameter is w. P(t) denotes a path of length t. The legs from w towards u are paths P(2), ..., $P(2^{k-2})$, $P(2^{k-1})$, $P(2^k)$. The first one of these, P(2), is at distance 2 from w. For $t \geq 1$ the part of the diameter between legs $P(2^t)$ and $P(2^{t+1})$ is a path $P(2^t)$ at the centre of which is a leg P(1). The part of the

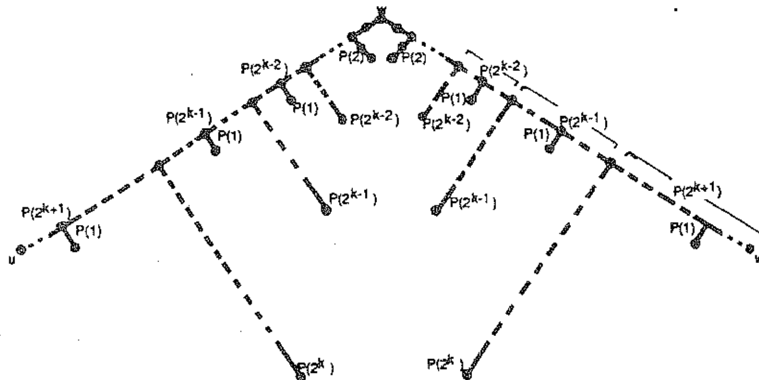

Figure 4.2

diameter between the leg $P(2^k)$ and vertex u is a path $P(2^{k}+1)$ on which there is a leg P(1) at distance $3*(2^{k-1})$ from u. The graph is symmetric about w and can be described similarly between w and v. Clearly $B(G_k) = 3$.

It is not difficult to see that algorithm LST1, after choosing u as the starting vertex of the main diameter, will merge all the legs of length greater than 1 between u and w by turning them towards w thus creating a level structure of width k+1.

We note that algorithm GPS will fare equally poorly in this particular case.

The performance of algorithm LST1 can be improved in some instances by applying a "level smoothing operation" after all the vertices are assigned levels. By comparing widths of three consecutive level classes, vertices can be reassigned to neighboring level classes to further redistribute them as uniformly as possible. This could reduce in some cases the width of the level structure generated.

5. EMPIRICAL EVALUATION

The performance of any heuristic algorithm must be evaluated empirically. In this section, by performance of an algorithm we mean its ability to minimize the width of a level structure produced. The empirical study of time efficiency of the algorithm is not considered here. Besides algorithm LST1, we also evaluated the performance of algorithm LST.

The results obtained by applying the algorithm to some examples (data) can be studied in two ways.

(1) If the theoretical value of the bandwidth of a tree is known, as is the case with binary trees, stars, caterpillars and some other graphs, one can compare the results obtained with the optimum results.

(2) In general, in the case where the exact bandwidth of a

tree is not known, one can evaluate the performance of the algorithm by comparing the results with those obtained by some well known algorithms. Therefore we evaluate the performance of algorithms LST and LST1 by using the same test data for algorithms LST, LST1 and GPS and comparing the results obtained. Although the ACM algorithm 582 is the most time and space efficient version of algorithm GPS, we opted for the simpler version, ACM algorithm 508, which has the same mathematical properties. This was run on the VAX 8500 system of Concordia University. Algorithm LST and LST1 were run on an IBM-PC. The details of the implementation of LST and LST1 in Pascal are found in [10].

Two groups of problems were chosen in the comparative analysis of the three algorithms.

GROUP 1. Here we have a group of 16 problems, numbered 1 to 16, most of whom are chosen for some special feature. The small size of many of the problems was very helpful in analyzing the nature of the tree, the operation of the algorithm and the intermediate level structures generated.. Some of the problems in this group are listed below.

No. 3. A tree on 30 vertices with maximum degree 5.

No. 5. A rooted complete binary tree.

No. 6. A rooted binary tree.

No. 7. A tree where off-diameter subtrees are connected to the main diameter by "small subtrees".

No. 8. Similar to No. 7. However there is only one off-diameter subtree here.

No. 9. A tree on 36 vertices. Possible 6 deep recursion.

No.10. The tree in example 4.1 (the optimum result example)

No.11. The complete binary rooted tree on 63 vertices.

No.12. A star.

No.13. A tree with just one off diameter subtree and only a unique choice of diameter.

No.14. A caterpillar.

No.15. A spider.

No.16. A large tree with overlapping spider subtree.

The level widths obtained for these problems are tabulated in Table 1 a.

GROUP 2. The second group consisted of 15 problems. The problems in this group are trees generated randomly by procedure gentree. Given the number of vertices n, the maximum degree d and the probability p of an edge in the tree, this procedure generates a tree as follows. Starting with vertex 1, at every vertex, d edges are created, each with probability p. The endpoint of a new edge is assigned the next vertex number. This process continues until n vertices are created. Various values of n, d and p were arbitrarily chosen to generate these 15 trees. Some problems of large size were included in this group. Problem No. 31 and 32 are complete binary trees which are the exception to the rule of random generation. All the three algorithms were applied to these 15 problems. Procedure gentree, which generates each tree randomly starting with vertex 1 as the root, also computes in each case, as byproducts, the rooted level structure L_1 and level widths of L_1. The results are tabulated in Table 1b.

6. CONCLUSIONS.

From our theoretical and empirical results the following observations and conclusions can be made.

1. From the two tables it is clear that LST1 almost always produces level structures of smaller level width than does GPS, and the difference is quite considerable in most cases, ranging between 13% to 66% in all cases but 2 in Table 1a. and between 8% to 100% in all cases but one in Table 1b. The widths produced by LST are comparable to those by LST1 though in most cases slightly higher. These tables show that algorithm LST1 has a slight edge over LST.

2. If the tree has one dominant (in size) off-diameter subtree as in problem NO. 13, GPS seems to produce really poor results. The level structure width can be as much as double that of LST1. Similar results should be expected if there are an odd number of components of T-D rather than just

one. GPS has the same disadvantage when the nodes connecting these components to the diameter are sufficiently far apart.

Problem No.	$\|V(T)\|$	$\Delta(T)$	W(LST1)	W(LST)	W(GPS)	B(GPS)
1	10	4	2	2	3	3
2	20	3	2	3	3	3
3	30	5	3	4	4	5
4	34	4	7	6	8	8
5	31	3	5	4	5	5
6	25	3	4	4	5	5
7	50	5	7	7	6	7
8	22	4	3	3	5	5
9	36	4	4	5	4	5
10	103	3	3	3	5	5
11	63	3	8	8	9	9
12	9	8	3	4	4	4
13	26	4	3	3	6	6
14	35	6	4	4	5	6
15	15	6	3	3	3	3
16	108	7	7	4	7	7

Table 1 a.

No.	$\|V(T)\|$	$\Delta(T)$	p	W(T)	W(LST1)	W(LST)	W(GPS)	B(GPS)
21	25	4	0.60	9	4	4	4	4
22	25	4	0.70	12	5	4	5	6
23	45	4	0.60	20	6	6	7	7
24	30	5	0.50	17	5	6	6	7
25	50	3	0.65	11	5	5	7	7
26	50	7	0.35	25	8	9	10	10
27	40	3	0.80	15	4	5	5	6
28	40	5	0.50	23	7	7	7	8
29	50	5	0.45	23	5	6	10	10
30	40	6	0.40	18	5	5	7	8
31	127	3	1.00	64	15	13	17	17
32	255	3	1.00	128	24	24	33	33
33	200	4	0.45	59	14	14	18	18
34	300	5	0.40	54	26	29	29	35
35	200	4	0.40	36	12	11	13	17

Table 1 b.

p: probability of an edge $\Delta(T)$: maximum degree in T
W(T): Level width of L_1 $|V(T)|$: No. of vertices in T
W(LST1): Level width by LST1 W(LST): Level width by LST
W(GPS): Level width by GPS B(GPS): bandwidth by GPS

3. If the tree is fairly regular and the density evenly spread, GPS and LST come quite close in their output as in No.4 and No.11. On the other hand, in the second group of problems, a smaller value of p and larger value of d indicate that there may be variation in the degrees of vertices. In this case the GPS level width is as much as double that of LST/LST1.

The relationship between the level structure width and the concept of even distribution of density can be expressed more clearly as follows. Let T be a tree satisfying the following two properties:
(1) the length of the diameter is "sufficiently" large
(2) there is a vertex x and a positive integer p such that $|S| = |\{ y \mid d(x,y) = p \}|$ is much larger than B(G)
then algorithm GPS cannot find a good level structure.

4. One can compare the results for the complete binary tree with the (theoretical) bandwidth of a binary tree. Chung [3] has proved that for a complete (rooted) binary tree T_k with k levels

$$B(T_k) = \lceil (2^{k-1} - 1)/(k-1) \rceil.$$

Problems No. 5, 11, 31, 32 are complete binary trees with 5, 6, 7, 8 levels (k) respectively. Chung's formula gives their bandwidths as 4, 6, 11, 19 respectively. Our results are not particularly close to the bandwidth, though they are closer than GPS. The GPS results seem to be getting farther from optimum value as the problem size gets larger.

5. One may ask whether a level structure of smaller width necessarily results in smaller bandwidth after labelling? The answer to this question is no. In problem No.12, algorithm LST tries to spread the vertices of a star in three levels of equal widths. However, an application of the GPS numbering algorithm to this will not result in a bandwidth labelling of the star. For the same problem the GPS algorithm produces a level structure of larger width, yet with GPS numbering this results in a bandwidth labelling of the star.

6. Only 6 out of 31 problems list a smaller width for LST than LST1. Two of these 6 problems are binary trees. This result seems to show that the additional overhead to redistribute small subtrees in the second pass is not worthwhile.

We may conclude that algorithm LST1 very consistently generates level structures of much smaller width than does algorithm GPS. This fact indicates that LST1 will be more successful in reducing bandwidths of trees than other known bandwidth reduction algorithms.

REFERENCES

1. Bondy J. A. and U. S. R. Murty. Graph Theory with Applications. The Macmillan Press Ltd. (1976).

2. Chinn P. Z., J. Chvátalová, A.K. Dewdney and N. E. Gibbs. "The bandwidth problem for graphs and matrices: a survey". Journal of Graph Theory 6 (1982). pp. 223-254.

3. Chung F. R. K. "Some problems and results on labelings of graphs". Bell Laboratories. Murray Hill, New Jersey.

4. Crane H. L. Jr., N. E. Gibbs, W. G. Poole Jr. and P. K. Stockmeyer. "Algorithm 508. Matrix bandwidth and profile reduction [FI]". ACM transactions on Mathematical Software 2, 4 (1976). pp. 375-377.

5. Cuthill E. and J. McKee. "Reducing the bandwidth of sparse symmetric matrices". Proceedings of the 24th National Conference of ACM. (1969). pp. 157-172.

6. Garey M. R., R. L. Graham, D. S. Johnson and D. E. Knuth. "Complexity results for bandwidth minimization". SIAM Journal of Applied Mathematics. 34 (1978). pp.477-495.

7. Garey M. R. and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., San Fransisco. (1979).

8. Gibbs N. E., W. G. Poole and P. K. Stockmeyer. "An algorithm for reducing the bandwidth and profile of a sparse matrix". SIAM Journal of Numerical Analysis. 13 (1976). pp. 235-251.

9. Gibbs N. E., W. G. Poole and P. K. Stockmeyer. "A comparison of several bandwidth and profile reduction algorithms". ACM Transactions on Mathematical Software. 2, 4 (1976). pp. 322-330.

10. GowriSankaran C. "A Bandwidth Reduction Algorithm for Trees". M. Comp.Sci. Thesis, Concordia University, Montreal. (1988).

11. Harary F. Problem 16 in Theory of Graphs and its applications. M. Fiedler, Ed. Czechoslovak Academy of Science. Prague. (1967).

12. Lewis J. G. "Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms". ACM Transactions on the Mathematical Software. 8, 2 (1982). pp.180-189.

13. Lewis J. G. "Algorithm 582. The Gibbs-Poole-Stockmeyer and Gibbs-King algorithms for reordering sparse matrices". ACM Transactions on the Mathematical Software. 8, 2 (1982). pp.190-194.

14. Papadimitriou C. H. "The NP-completeness of the bandwidth minimization problem". Computing. 16 (1976). pp. 263-270.

15. Turner J. "Probabilistic Analysis of bandwidth minimization algorithms". Proceedings of the 15th ACM Symposium on Theory of Computing. (1983). pp.467-486.