SIAM J. COMPUT.
Vol. 14, No. 1, February 1985

013

POLYNOMIAL TIME ALGORITHMS FOR THE MIN CUT PROBLEM ON DEGREE RESTRICTED TREES*

MOON-JUNG CHUNG†, FILLIA MAKEDON‡,
IVAN HAL SUDBOROUGH§ AND JONATHAN TURNER¶

Abstract. Polynomial algorithms are described that solve the MIN CUT LINEAR ARRANGEMENT problem on degree restricted trees. For example, the *cutwidth* or *folding number* of an arbitrary degree d tree can be found in $O(n(\log n)^{d-2})$ steps. This has applications to integrated circuit layout, in particular the layout of Weinberger arrays [41]. This also yields an algorithm for determining the black/white pebble demand of degree three trees. We also show that for degree three trees, cutwidth is identical to search number and give a forbidden subgraph characterization of degree three trees having cutwidth k.

Key words. MIN CUT LINEAR ARRANGEMENT problem, cutwidth, search number, black/white pebble demand, integrated circuit layout, VLSI

1. Introduction. Let $G=(V, E)$ be a finite undirected graph. A (one-dimensional) *layout* of G is a one-to-one function σ mapping the set of vertices V onto $\{1, 2, \cdots, |V|\}$. We consider the following layout problem:

MIN CUT LINEAR ARRANGEMENT PROBLEM (MIN CUT)
Instance: A finite undirected graph $G=(V, E)$ and a positive integer k.
Question: Does there exist a layout σ such that, for all i $(1 \leqq i < |V|)$, there are at most k edges in the set $cut_\sigma(i)=\{\{x, y\} \in E | \sigma(x) \leqq i \wedge \sigma(y) > i\}$?

The *cutwidth* of G with respect to a layout σ, denoted $\gamma_\sigma(G)$ is defined as $\max\{|cut_\sigma(i)|: 1 \leqq i < |V|\}$. The *cutwidth* of G, denoted $\gamma(G)$, is defined as the minimum over all layouts σ of $\gamma_\sigma(G)$. A simple example is shown in Fig. 1.

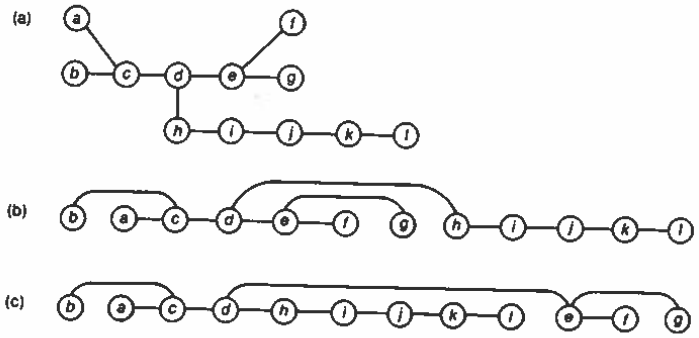

FIG 1. (a) *A tree T.* (b) *A layout minimizing the sum of the edge lengths.* (c) *A layout minimizing the cutwidth.*

* Received by the editors October 31, 1983.

† Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York. The research of this author was supported in part by the National Science Foundation under grants MCS 79-08919 and 81-09280.

‡ Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois 60616. The reasearch of this author was supported in part by the National Science Foundation under grants MCS 79-08919 and 81-09280.

§ Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60201. The research of this author was supported in part by the National Science Foundation under grants MCS 79-08919 and 81-09280.

¶ Bell Laboratories, Murray Hill, New Jersey 07974. Current address, Computer Science Department, Box 1045, Washington University, St. Louis, Missouri 63130.

MIN CUT is one of several one-dimensional layout problems for undirected graphs. Other layout problems are the BANDWIDTH MINIMIZATION problem and the OPTIMAL LINEAR ARRANGEMENT problem [2], [9], [10], [13], [15], [22], [30], [38]. The MIN CUT problem for general graphs is known to be NP-complete [11], [39]. A recent result shows that it is NP-complete even when restricted to graphs with maximum vertex degree three [23]. F. R. K. Chung [3] and others call the cutwidth of a graph its "folding number."

The complexity of the MIN CUT problem when restricted to trees has been an open problem of some recognized importance. Lengauer [19] described a polynomial time approximation algorithm for this problem. Lengauer's algorithm obtains a layout σ for any tree T such that $\gamma_\sigma(T) < 2\gamma(T)$. In addition, Lengauer gave a linear time algorithm to obtain an optimum layout of a complete k-ary tree, where k is any positive integer. Some of the known applications for the MIN CUT LINEAR ARRANGEMENT problem when restricted to trees are discussed below.

1.1. Black/white pebble demand for binary trees. Let $G=(V, E)$ be a directed acyclic graph. The black/white demand of G is the minimum number of pebbles required to play the black/white pebble game on G. The rules of the black/white pebble game are:

- A white pebble may be placed on any vertex at any time.
- A white pebble may be removed from a vertex only if all the predecessors of that vertex are pebbled.
- A black pebble may be placed on a vertex only if the predecessors of that vertex are pebbled.
- A black pebble may be removed from a vertex at any time.

The object of the pebble game is to place a pebble on a distinguished vertex called the *sink*, using as few pebbles as possible.

Let $T=(V, E)$ be a directed binary tree with a sink vertex of degree one. The number of black and white pebbles needed to pebble the sink of T is equal to the cutwidth of the underlying undirected tree. (We wish to thank Nick Pippinger for pointing this out to us [34], [35], [36].) This can be seen by the following observations:

1. Let T be such a tree and let S be a black/white pebble game strategy for T using k pebbles. (We can assume that S does not involve recomputation [21].) A *defining move* of the sequence of steps in S is a move that either adds a black pebble to a vertex or deletes a white pebble from a vertex. Define a layout σ of T by $\sigma(x)=i$ if and only if the ith defining move of S involves vertex x. It follows that $\gamma_\sigma(T) \leqq k$. This is because at the ith defining move if the edge (y, z) is in $cut_\sigma(i)$ then there is a pebble on y. Since every vertex has out-degree at most one, it follows that each of these pebbles is on a unique vertex. Hence the size of $cut_\sigma(i)$ is bounded by k for all i.

2. Let T be a directed binary treee whose sink has degree one. Given a cutwidth k layout σ of T we can construct a pebbling strategy S for the black/white pebble game on T that uses at most k pebbles. During step i of this strategy the goal is to add a black pebble or to remove a white pebble from vertex $\sigma^{-1}(i)$. To accomplish this, white pebbles are added to all unpebbled predecessors of $\sigma^{-1}(i)$, a black pebble is placed on $\sigma^{-1}(i)$ or a white pebble is removed, and then black pebbles are removed from all vertices $\sigma^{-1}(j)$, $j \leqq i$ which are not predecessors of some vertex $\sigma^{-1}(h)$, $h > i$. By induction on i it can be shown that the number of pebbles on the vertices of T at each step i is not greater than the sum of the number of edges passing over $\sigma^{-1}(i)$ in the layout and the number of predecessors of $\sigma^{-1}(i)$. Using the fact that every vertex in T has at most two predecessors and one successor and the fact that any vertex with

two predecessors has a successor, this sum can be shown to be bounded by k. As this is true at every step, the black/white pebble demand of T is bounded by its cutwidth. (Note: it follows by a similar argument that if T is a directed tree with in degree $\leqq d$ in which the sink has degree one, then the black/white pebble demand of T is not greater than $\gamma(T)+\lfloor(d-1)/2\rfloor$. Thus it follows that the black/white pebble demand of such a tree is between $\gamma(T)$ and $\gamma(T)+\lfloor(d-1)/2\rfloor$. So, the algorithm we present for determining the cutwidth of a tree also gives approximate information about the black/white pebble demand for arbitrary degree bounded trees.)

Previous work on the black/white pebble demand of trees has appeared in [16], [17], [21], [26]. Our MIN CUT algorithm gives an $O(n \ln n)$ algorithm for determining the black/white pebble demand of binary trees.

1.2. VLSI layout. A central problem in VLSI is area efficient embeddings of various graphs in the plane. There are methodologies for automated component placement that suggest placing circuit elements in rows or along a single line [5], [6], [7], [20], [27], [33], [40], [43]. For example, Dolev and Trickey [5] have such a strategy in mind when they consider the MIN CUT LINEAR ARRANGEMENT problem for trees with the additional restriction that the edges are not allowed to cross. (They give an $O(n \log n)$ algorithm for this planar layout version of MIN CUT on trees.) Foster and Kung [7] consider the construction of VLSI circuits for regular languages with programmable building blocks. The circuits form degree three trees and, when automatic construction is desired from a given regular expression, one assigns the active elements to positions along the bottom row of a "programmable recognizer array" (PRA); the connections using at most $\log n$ tracks above. To minimize the number of tracks (and hence the area) for such circuits one positions the basic cells along the bottom row of the PRA in such a way that cutwidth is minimized.

Another important application is the layout of logic circuits using Weinberger gate arrays [1], [41], [43]. This technique is used in several experimental silicon compilers.

We describe an algorithm which solves the MIN CUT problem for trees. The algorithm obtains the optimum cutwidth or folding number for an arbitrary tree. In addition, for any fixed $d \geqq 3$, the algorithm takes at most $O(n(\log n)^{d-2})$ steps to determine the cutwidth of a degree d tree with n vertices. We observe that the degree of the polynomial time bound grows with the degree of the tree. (Recently, Yannakakis [42] has also found an $O(n \log n)$ algorithm for all trees.)

We also give an algorithm that not only determines the cutwidth but produces an optimal layout as well.

It should, perhaps, be noted that a layout to minimize cutwidth is not, in general, the same as a layout to minimize the sum of all the edge lengths (the latter being the goal of the OPTIMAL LINEAR ARRANGEMENT problem). For example, in Fig. 1, the first layout is among the best for OPTIMAL LINEAR ARRANGEMENT, but is not optimal for cutwidth, and the second layout is among the best for cutwidth, but is not a good layout for OPTIMAL LINEAR ARRANGEMENT. There are several results concerning the OPTIMAL LINEAR ARRANGEMENT problem on trees in the literature [2], [12], [15], [38]. The best result currently is due to F. R. K. Chung [2] and gives an $O(n^{1.58})$ algorithm. Optimal layouts for cutwidth are quite obviously, in general, not good layouts for bandwidth. It is known that the BANDWIDTH MINIMIZATION problem, even for degree three trees, is NP-complete [9].

In § 2 we give a general characterization of cutwidth k trees. For the special case of degree three trees we give a specific sequence of tree families with the property: a

tree has cutwidth at most k if and only if it does not contain a homeomorphic image of a tree in the $(k+1)$st family. In fact, the same result was obtained by Parsons [31], [32] for the notion of search number. Thus, we obtain the somewhat surprising result that, for the class of trees with degree three, a tree has cutwidth k if and only if it has search number k. (The search number of a graph is defined in [25]. In fact, a more recent result shows that search number and cutwidth are the same for all degree three graphs [22]. This uses the fact that "recontamination" does not reduce search number [14].) In § 3 we give an $O(n(\log n)^{d-1})$ algorithm for determining the cutwidth of any degree d tree. In § 4 we give a related algorithm for determining the cutwidth of a degree three tree in time $O(n \log n)$. This algorithm is then used to speed up the degree d algorithm to $O(n(\log n)^{d-2})$. The decision algorithms described in §§ 3 and 4 yield information that can be used to produce an optimal layout of the tree. An algorithm to construct the layout is described in § 5. We conclude with a list of open problems.

2. Characterization of trees with cutwidth k. Let $T=(V, E)$ be a tree and let $\{u, x_1, \cdots, x_r\} \subseteq V$. Define $T(u, x_1, \cdots, x_r)$ as the largest subtree of T that contains u but does not contain any of $x_1, \cdots, x_r$. This definition is illustrated in Fig. 2. Let σ be a layout of T. The vertex which is mapped to 1 by σ is referred to as the *leftmost vertex* in the layout. The vertex which is mapped to $|V|$ by σ is referred to as the *rightmost vertex.*

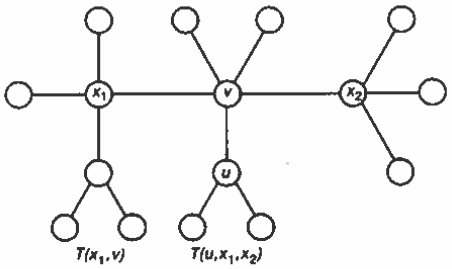

FIG. 2. *Notation for undirected tree.*

THEOREM 2.1 (general characterization theorem). *Let T be an undirected tree. $\gamma(T) \leqq k \Leftrightarrow$ every vertex u of degree at least two has neighbors x_1, x_2 such that $\gamma(T(u, x_1, x_2)) \leqq k-1$.*

It follows from Theorem 2.1 that the tree in Fig. 3 has cutwidth four since vertex u does not have neighbors x_1, x_2 such that $\gamma(T(u, x_1, x_2)) \leqq 2$.

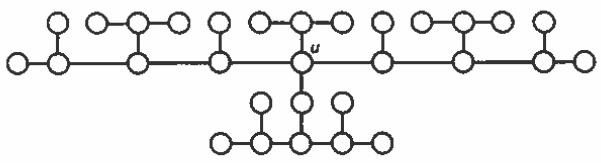

FIG. 3. *Application of Theorem* 2.1.

Proof. ($\Rightarrow$) Assume to the contrary that $\gamma(T) \leqq k$ and there exists a vertex u of degree $\geqq 2$ such that for every pair of vertices x_1, x_2 adjacent to u, $\gamma(T(u, x_1, x_2)) \geqq k$. Now let σ be a layout of T such that $\gamma_\sigma(T) \leqq k$ and let P be the path connecting the leftmost and rightmost vertices of T under σ. If u is an internal vertex of P let x_1, x_2 be the neighbors of u on P. As shown in Fig. 4(a), P passes entirely over $T(u, x_1, x_2)$

in the layout but since $\gamma(T(u, x_1, x_2)) \geqq k$ it follows that the cutwidth of the layout exceeds k, contradicting the assumption. If u is an endpoint of P then let x be the neighbor of u on P. In this case P passes over $T(u, x)$ and again we obtain a contradiction since $\gamma(T(u, x)) \geqq k$. Finally if u is not on P, let x be the neighbor of u on the path from u to P. Once again, P passes over $T(u, x)$ yielding a contradiction.

($\Leftarrow$) Consider two cases. First suppose that every vertex u has a neighbor x such that $\gamma(T(u, x)) \leqq k-1$. Starting from any vertex y_1, construct a path $y_1, \cdots, y_r$, where $\gamma(T(y_i, y_{i+1})) \leqq k-1$ for $1 \leqq i < r$ and $\gamma(T(y_r, y_{r-1})) \leqq k-1$. This construction is shown in Fig. 4(b). It is clear that $\gamma(T) \leqq k$.

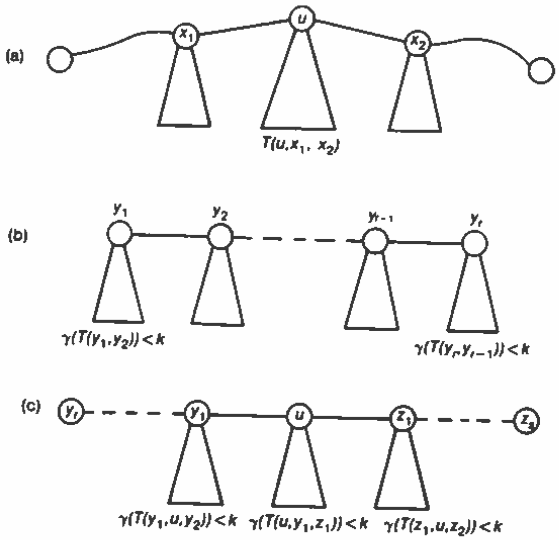

FIG. 4. *Illustrations for Theorem* 2.1.

Next suppose that there is some vertex u such that for all neighbors x of u, $\gamma(T(u, x)) \geqq k$. By the hypothesis, u has neighbors y_1, z_1 such that $\gamma(T(u, y_1, z_1)) \leqq k-1$. Now if y_1 is not a leaf then it has a neighbor y_2 such that $\gamma(T(y_1, u, y_2)) \leqq k-1$. Similarly if y_2 is not a leaf then it has a neighbor y_3 such that $\gamma(T(y_2, y_1, y_3)) \leqq k-1$. Continuing in this fashion one can construct a path $u, y_1, \cdots, y_r$ such that for $1 \leqq i \leqq r-2$, $\gamma(T(y_{i+1}, y_i, y_{i+2})) \leqq k-1$, and y_r is a leaf. One can construct a similar path $u, z_1, \cdots, z_s$. This construction is illustrated in Fig. 4(c). Again it is clear that $\gamma(T) \leqq k$. □

Let $T_d(k)$ denote the set of smallest trees with degree d and cutwidth k.

COROLLARY 2.1. $T_3(1)$ *is the singleton set containing the tree with two vertices. For* $k > 1$ *each tree in* $T_3(k)$ *can be formed by identifying a leaf in three (not necessarily distinct) trees from* $T_3(k-1)$.

This construction is illustrated in Fig. 5 for $T_3(2)$, $T_3(3)$, and $T_3(4)$.

Proof. The proof is by induction. The basis, $k = 1$ is immediate. Assume then that $k > 1$ and let T be any tree in $T_3(k)$. By Theorem 2.1, T must contain a vertex u with neighbors x_1, x_2, x_2 such that $\gamma(T(u, x_1, x_2)) \geqq k-1$, $\gamma(T(u, x_1, x_3)) \geqq k-1$ and $\gamma(T(u, x_2, x_3)) \geqq k-1$. Since T is a smallest degree three tree with cutwidth k, it follows that $T(u, x_1, x_2)$, $T(u, x_1, x_3)$ and $T(u, x_2, x_3)$ must be smallest degree three trees with cutwidth $k-1$, that is they must be in $T_3(k-1)$. □

Let $n_d(k)$ be the number of vertices in a smallest degree d tree with cutwidth k.

COROLLARY 2.2. $n_3(k) = 3^{k-1} + 1$.

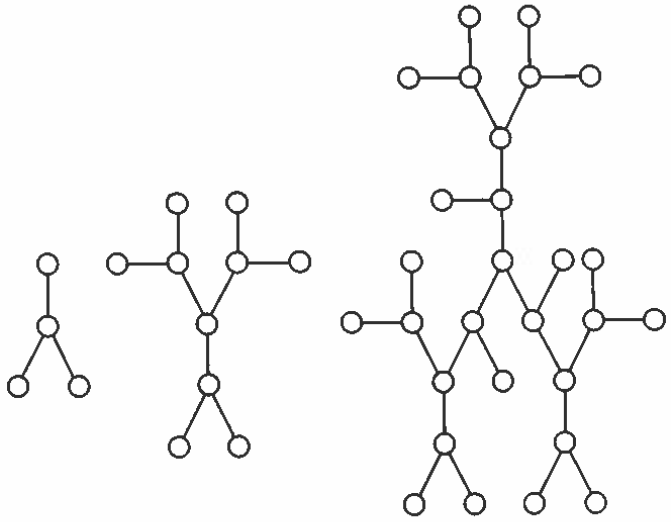

FIG. 5. $T_3(2)$, $T_3(3)$, *and* $T_3(4)$.

Proof. By induction. The basis $k=1$ is immediate from Corollary 2.1. Assume then that $n_3(k-1)=3^{k-2}+1$. By Corollary 2.1,

$$n_3(k)=3n_3(k-1)-2=3(3^{k-2}+1)-2=3^{k-1}+1. \qquad \square$$

The next corollary relates the cutwidth of a tree to its search number. To understand the notion of search number, let $G=(V,E)$ be a graph and think of the vertices of G as rooms, and the edges as interconnecting corridors. Now, assume that there is an escaped convict lurking somewhere within G. Your job is to organize a search party to capture the fugitive and since you have limited resources you want to do it with the fewest possible number of searchers. The *search number* of G is the minimum number of searchers required to guarantee that the fugitive is captured. The following result follows immediately from Theorem 2.1 and a characterization of trees with search number k given by Parsons [31]. (We are indebted to S. L. Hakimi for pointing out Parson's result.)

COROLLARY 2.3. *A degree three tree has cutwidth* $k \Leftrightarrow$ *it has search number* k.

The next theorem provides a forbidden subgraph characterization of degree three trees with cutwidth k. First we require some definitions. Let $f(T)$ be the set of trees obtained from T by replacing a single edge $\{u, v\}$ with the tree shown in Fig. 6, where x and y are new vertices and neither u nor v is adjacent to a leaf of T. If S is a set of trees, $f(S)$ is the union of the sets $f(T)$ for all T in S. Let $L_3(1)$ be the singleton set containing the tree on two vertices. $M_3(k)$ is the union of $L_3(k)$ and $f(L_3(k))$. $L_3(k+1)$ is the set of trees that are obtained by identifying a leaf in three (not necessarily distinct) trees from $M_3(k)$. For $k \leqq 5$, $L_3(k)=T_3(k)$.

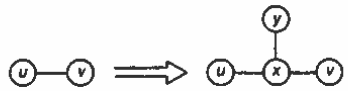

FIG. 6. *Definition of* $f(T)$.

THEOREM 2.2. *Let* T *be a degree three tree.* $\gamma(T)=k \Leftrightarrow T$ *contains a homeomorphic image of a tree in* $L_3(k)$ *and does not contain a homeomorphic image of any tree in* $L_3(k+1)$.

Proof. By induction. The basis, $k=1$, is obvious. By Theorem 2.1, T has some vertex u with neighbors x_1, x_2, x_3 such that $\gamma(T(u, x_1, x_2)) \geqq k-1$, $\gamma(T(u, x_1, x_3)) \geqq k-1$ and $\gamma(T(u, x_2, x_3)) \geqq k-1$. By the induction hypothesis each of these subtrees contains a homeomorphic image of a tree in $L_3(k-1)$. Consider the subtree consisting of these three images together with the paths that join them to u. This is a homeomorphic image of a tree in $L_3(k)$. Since each tree in $L_3(k+1)$ has cutwidth $k+1$, T cannot contain a homeomorphic image of any tree in $L_3(k+1)$. □

Theorem 2.2 can be generalized to give a somewhat more complex forbidden subgraph characterization of degree d trees.

The next theorem relates the number of vertices in a degree bounded tree to its cutwidth. This will be used in the complexity analysis of our cutwidth minimization algorithm.

THEOREM 2.3. *Let T be a degree d tree with cutwidth k. T contains at least $(d/(d-2))^{k-1}+1$ vertices.*

Proof. By Theorem 2.1 T contains a vertex u with neighbors $x_1, \cdots, x_r$ $(r \leqq d)$ such that $\gamma(T(u, x_i, x_j)) \geqq k-1$ for $1 \leqq i < j \leqq r$. Now select x_i, x_j so that $|T(x_i, u)| \geqq |T(x_j, u)| \geqq |T(x_h, u)|$ for $1 \leqq h \leqq r$, $i \neq h \neq j$. If $|T| = n$ and $|T(u, x_i, x_j)| = m$ then $(m-1) \leqq ((r-2)/r)(n-1)$ or $n \geqq (d/(d-2))(m-1)+1$, since $r/(r-2) \geqq d/(d-2)$. Since this holds for all trees T and since $m \geqq n_d(k-1)$ it follows that

$$n_d(k) \geqq \frac{d}{d-2}(n_d(k-1)-1)+1 \geqq \left(\frac{d}{d-2}\right)^{k-1}(n_d(1)-1)+1 = \left(\frac{d}{d-2}\right)^{k-1}+1. \quad \square$$

COROLLARY 2.4. *Let T be a degree d tree with n vertices. $\gamma(T) < (d/2)\ln n + 1$.*

Proof. Let $k = \gamma(T)$. By the theorem

$$n > (d/(d-2))^{k-1}, \qquad k-1 < \frac{\ln n}{\ln (d/(d-2))} < \frac{d}{2}\ln n. \quad \square$$

3. A cutwidth minimization algorithm for trees. This section describes a general algorithm for the cutwidth minimization problem on trees. The time bound for the algorithm is

$$O\left(\binom{k+d-1}{d-1} n \log n\right),$$

where n is the number of vertices in the tree, k is its cutwidth and d is the maximum vertex degree. For trees with fixed maximum degree this quantity is $O(n(\ln n)^{d-1})$.

In the remainder of this section we assume that all trees are directed and rooted. This is strictly for notational convenience. The cutwidth of a directed tree is the same as the cutwidth of the underlying undirected tree.

If T is a tree, $T[u]$ denotes the induced subtree with root u. $T[u, x_1, \cdots, x_r] = T[u] - \cup_{i=1}^{r} T[x_i]$. These definitions are illustrated in Fig. 7.

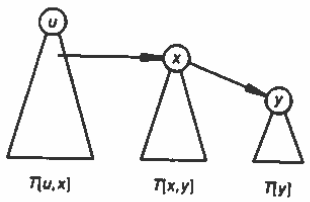

FIG. 7. *Notation for directed trees.*

Let T be a tree with root u and children $v_1, \cdots, v_d$ of u. Define $\delta(T) = \min_{1 \leq i \leq d} \max\{\gamma(T[v_i]), \gamma(T[u, v_i])\}$ when $d \geq 1$ and, $\delta(T) = 0$ when $d = 0$. Note that $\delta(T) \leq \gamma(T) \leq \delta(T) + 1$, since T is formed by joining $T[v_i]$ and $T[u, v_i]$ together with an edge, for every i.

Let x be a vertex in a tree T and let k be a positive integer. We say that x is *k-critical* if $k = \delta(T[x])$ and for all children y of x, $\gamma(T[x, y]) \geq k$.

The next theorem is essentially a restatement of Theorem 2.1. We will find this form more convenient in what follows.

THEOREM 3.1. *Let T be a tree with root u and let $k = \delta(T)$. $\gamma(T) = k \Leftrightarrow T$ has no k-critical vertex or T has exactly one k-critical vertex x and x has children y, z such that $\gamma(T[u, y, z]) < k$.*

Proof. ($\Rightarrow$) Let T' be the underlying undirected tree. If T has two k-critical vertices v, x then one can show that at least one of them, say x, does not have neighbors y, z such that $\gamma(T'(x, y, z)) < k$, contradicting Theorem 2.1. Similarly, if T has a k-critical vertex x with no children y, z such that $\gamma(T[u, y, z]) < k$, then x has no neighbors y, z such that $\gamma(T'(x, y, z)) < k$, contradicting Theorem 2.1. (Note $T'(x, y, z)$ is the underlying tree for $T[u, y, z]$.)

($\Leftarrow$) If T has no k-critical vertex then every vertex x in T' has neighbors y, z such that $\gamma(T'(x, y, z)) < k$, and by Theorem 2.1 $\gamma(T) \leq k$. Since $\delta(T) = k$, $\gamma(T) = k$. If T has exactly one k-critical vertex x that satisfies the condition stated, then Theorem 2.1 applies and again $\gamma(T) = k$. □

Using Theorem 3.1 we can compute the cutwidth of small trees by hand. We will illustrate this procedure with an example before giving the formal presentation of the algorithm. Given a tree T, we work from the bottom up assigning labels to each of the vertices in the tree. These labels consist of a decreasing sequence of integers; the largest integer in the sequence is the cutwidth of the subtree whose root is the associated vertex. Consider the tree T_1 shown in Fig. 8(a). The label next to vertex b means that the cutwidth of the subtree containing just vertex b is 0. Given the labels on b and c we want to use Theorem 3.1 to determine the cutwidth of T_1. The first step is to determine $\delta(T_1)$. In this case, one can see that $\delta(T_1) = 1$, hence the cutwidth of T_1 is either 1 or 2. The next step is to determine if T_1 contains a 1-critical vertex. In fact, a is 1-critical, so the next step is to determine if a satisfies the condition given in the theorem. In this case the answer is yes, since $\gamma(T_1[a, b, c]) = 0 < 1$. Thus, according to the theorem, $\gamma(T_1) = 1$, which is clearly true. The label next to vertex a in the figure is [1, 0]. The meaning of this label is that (1) the cutwidth of $T_1[a]$ is 1 and (2) $T_1[a]$ contains a 1-critical vertex with children b, c such that the cutwidth of $T_1[a, b, c]$ is 0.

Now consider the tree T_2 shown in Fig. 8(b). Using the result for T_1, one can show that $\delta(T_2) = 1$. Thus, we want to determine if T_2 contains a 1-critical vertex. In fact d is 1-critical, so the next step is to determine if d has children x, y such that

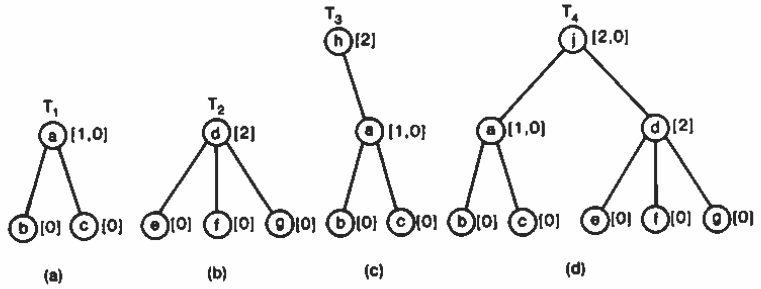

FIG. 8. *Example to illustrate cutwidth computation.*

$\gamma(T_2[d, x, y]) < 1$. Since it does not, we conclude from Theorem 3.1 that $\gamma(T_2) = 2$. The label next to vertex d means that the cutwidth of $T_2[d]$ is 2 and $T_2[d]$ contains no 2-critical vertex. A similar argument can be used to determine the cutwidth of T_3 in Fig. 8(c).

Now consider T_4 in Fig. 8(d). Using the results of the previous examples we can show that $\delta(T_4) = 2$. We can also see that j is a 2-critical vertex. Since j has children a, d such that $\gamma(T_4[j, a, d]) = 0 < 2$ it follows from Theorem 3.1 that the cutwidth of $T_4 = 2$. Furthermore, the label for vertex j is $[2, 0]$.

We now proceed with the formal presentation of the algorithm. We will use the usual lexicographic ordering on decreasing sequences of integers. $[a_1, \cdots, a_r] < [b_1, \cdots, b_s]$ if (1) for some i $(1 \leqq i \leqq \min\{r, s\})$, $a_i < b_i$ and for all j $(1 \leqq j < i)$, $a_j = b_j$ or (2) $r < s$ and for $1 \leqq j \leqq r$, $a_j = b_j$. We will also apply set operations to such sequences with the obvious interpretation.

Let T be a tree with root u. We define $\Gamma(T)$ recursively as follows:

- If $\gamma(T) = 0$ then $\Gamma(T) = [0]$.
- If $\gamma(T) = k$ and T contains no k-critical vertex then $\Gamma(T) = [k]$.
- If $\gamma(T) = k$ and T contains a k-critical vertex x then,

$$\Gamma(T) = [k] \cup \min_{y,z} \Gamma(T[u, y, z])$$

where y, z range over all children of x.

We can restate this definition in iterative form as follows. $\Gamma(T) = [a_1, \cdots, a_r]$ if $a_1 > \cdots > a_r \geqq 0$ and T contains vertices $x_1, \cdots, x_{r-1}$ where x_i has children y_i, z_i such that

1. For $1 \leqq i \leqq r$, $\gamma(T[u, y_1, z_1, \cdots, y_{i-1}, z_{i-1}]) = a_i$.
2. For $1 \leqq i < r$, x_i is an a_i-critical vertex in $T[u, y_1, z_1, \cdots, y_{i-1}, z_{i-1}]$.
3. $T[u, y_1, z_1, \cdots, y_{r-1}, z_{r-1}]$ contains no a_r-critical vertex.
4. There is no sequence $[b_1, \cdots, b_s] < [a_1, \cdots, a_r]$ that also satisfies conditions 1–3.

Let T be a tree with root u having children $v_1, \cdots, v_d$. In Fig. 9, an algorithm called Γ is described which computes $\Gamma(T)$ from $\Gamma(T[v_1]), \cdots, \Gamma(T[v_d])$. By applying *Gamma* $(\cdot)$ recursively we can attach a label $\lambda(x) = \Gamma(T[x])$, to each vertex x. A procedure for computing these labels is shown in Fig. 10. An example of a tree with the labels attached to the vertices is shown in Fig. 11. Once the labels have been computed finding an optimal layout is straightforward. The layout algorithm is described in § 5.

```
[1]  procedure Gamma(S_1, ..., S_d)
[2]      { Relabel if necessary so that S_1 ≥ ... ≥ S_d }
[3]      if d = 0 then return [0]
[4]      if d = 1 then begin
[5]          if min S_1 ≠ 0 then return S_1
[6]          y ← min {x > 0 | x ∈ S_1}
[7]          return [y] ∪ {x > y | x ∈ S_1}
[8]          end
[9]      { d ≥ 2 }
[10]     S_2 ← Gamma(S_2, ..., S_d)
[11]     S_1 ← Gamma(S_2, ..., S_d)
[12]     k ← max {S_1 ∪ S_2}
[13]     { k = δ(T) }
[14]     if k = max S_2 then return {k+1}
[15]     if k ≠ max S_1 or k = min S_1 then begin
[16]         if k = max S_1 then return {k} ∪ S_2
[17]         else return {k}
[18]         end
[19]     { T[v_1] contains a k-critical vertex }
[20]     if k = max S_1 then return {k+1}
[21]     H ← Gamma(S_1 − {k}, S_2, ..., S_d)
[22]     if k = max H then return {k+1}
[23]     else return {k} ∪ H
[24]     end
```

FIG. 9. *Recursive algorithm for computing* $\Gamma(T)$ *from* $\Gamma(T[u, v_1]), \cdots, \Gamma(T[u, v_d])$.

```
[1] procedure Label(T,x)
[2]     Let v_1, ..., v_d be the children of x.
[3]     Let S_i ← Label(T[v_i],v_i) for 1 ≤ i ≤ d.
[4]     λ(x) ← Gamma(S_1, ..., S_d)
[5]     return λ(x)
[6] end
```

FIG. 10. *Vertex labeling procedure.*

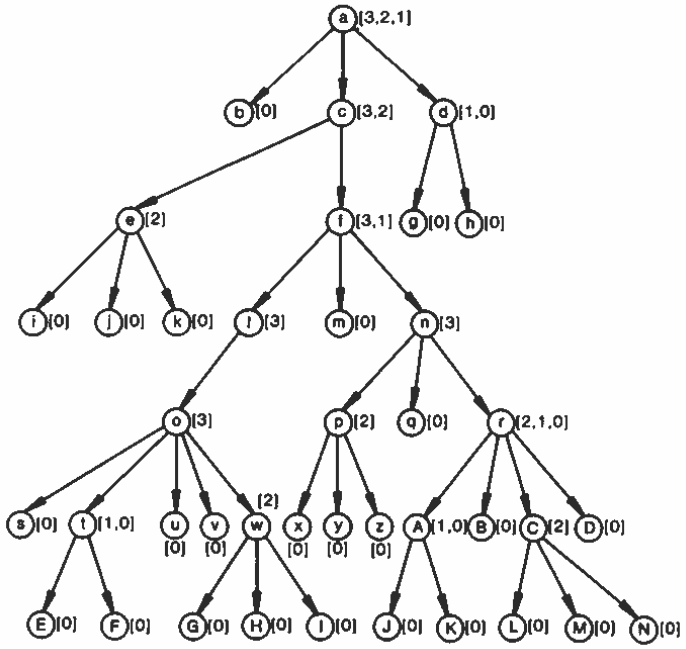

FIG. 11. *Example of vertex labeling produced by Gamma* $(\cdot)$.

The following theorem establishes the correctness of the algorithm.

THEOREM 3.2. *Let T be a tree with root u having children $v_1, \cdots, v_d$ and let $S_i = \Gamma(T[v_i])$, for all i $(1 \leqq i \leqq d)$. Gamma $(S_1, \cdots, S_d) = \Gamma(T)$.*

The proof of Theorem 3.2 requires a technical result given in Theorem 3.3. Let T_1 and T_2 be trees. The notation $T_1 : T_2$ denotes the tree obtained by making T_2 a subtree of the root of T_1. This operation is illustrated in Fig. 12.

THEOREM 3.3. *Let R, S, S' be trees with roots u, v, v' and let $T = R \cdot S$, $T' = R \cdot S'$. $\Gamma(S) \leqq \Gamma(S') \Rightarrow \Gamma(T) \leqq \Gamma(T')$.*

The situation described in Theorem 3.3 is shown in Fig. 13. The proof is given in the appendix.

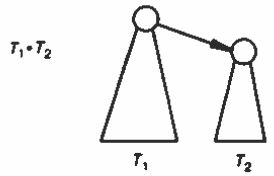

FIG. 12. *Definition of $T \cdot T_2$.*

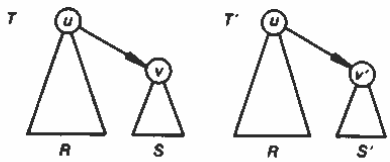

FIG. 13. *Illustration for Theorem* 3.3.

The following corollaries to Theorem 3.3 are used in the proof of Theorem 3.2. Let $T_1 \cdot T_2 \cdot \cdots \cdot T_r$ denote $(\cdots((T_1 \cdot T_2) \cdot T_3) \cdots) \cdot T_r)$.

COROLLARY 3.1. *Let $R, S_1, \cdots, S_r, S'_1, \cdots, S'_r$ be trees. If $\Gamma(S_i) \leqq \Gamma(S'_i)$ for $1 \leqq i \leqq r$ then $\Gamma(R \cdot S_1 \cdot \cdots \cdot S_r) \leqq \Gamma(R \cdot S'_1 \cdot \cdots \cdot S'_r)$.*

Proof. By successive applications of Theorem 3.3,

$$\Gamma(R \cdot S_1 \cdot \cdots \cdot S_r) \leqq \Gamma(R \cdot S'_1 \cdot S_2 \cdot \cdots \cdot S_r) \leqq \cdots \leqq \Gamma(R \cdot S'_1 \cdot \cdots \cdot S'_r). \qquad \square$$

Let R, S be trees and let u be a vertex in R. Let $R \cdot S|_u$ denote the tree formed by making the root of S a child of u as shown in Fig. 14.

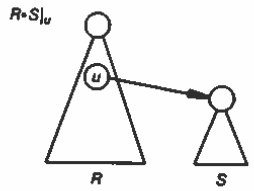

FIG. 14. $R \cdot S$.

COROLLARY 3.2. *Let R, S, S' be trees and let u be a vertex in R and let $T = R \cdot S|_u$, $T' = R \cdot S'|_u$. $\Gamma(S) \leqq \Gamma(S') \Rightarrow \Gamma(T) \leqq \Gamma(T')$.*

Proof. Let v be the root of R. The proof is by induction on the length of the path from v to u. (Basis) Assume $v = u$. In this case the statement reduces to Theorem 3.3. (Induction) If $v \neq u$ then let w be the child of v that is on the path from v to u. By induction we can assume that $\Gamma(T[w]) \leqq \Gamma(T'[w])$. We can then apply Theorem 3.3 with $R[v, w]$, $T[w]$ and $T'[w]$ and the result follows. $\square$

COROLLARY 3.3. *Let T be a tree with root u having children $v_1, \cdots, v_d$, $d \geqq 1$, such that $\Gamma(T[v_1]) \geqq \cdots \geqq \Gamma(T[v_d])$. $\delta(T) = \max\{\gamma(T[v_1]), \gamma(T[u, v_1])\}$.*

Proof. The result is trivially true for $d = 1$. Assume then that $1 < i \leqq d$. Since $T[v_1]$ is a subtree of $T[u, v_i]$, $\gamma(T[v_1]) \leqq \gamma(T[u, v_i])$. We can now apply Theorem 3.3 (with $R = T[u, v_1, v_i]$, $S = T[v_i]$ and $S' = T[v_1]$), yielding $\gamma(T[u, v_1]) \leqq \gamma(T[u, v_i])$. Hence $\max\{\gamma(T[v_1]), \gamma(T[u, v_1)\} \leqq \max\{\gamma(T[v_i]), \gamma(T[u, v_i])\}$. $\square$

COROLLARY 3.4. *Let T be a tree with $\delta(T) = k$ and let u be a vertex with child x such that for all children y of u, $\Gamma(T[x]) \geqq \Gamma(T[y])$. Then u is k-critical $\Leftrightarrow \gamma(T[u, x]) \geqq k$.*

Proof. The forward implication is immediate. For the converse, let y be any child of u, let $S = T[y]$, $S' = T[x]$, $R = T[u, x, y]$ and note that $R \cdot S = T[u, x]$ and $R \cdot S' = T[u, y]$. By Corollary 3.2, $\gamma(T[u, x]) \leqq \gamma(T[u, y])$. $\square$

COROLLARY 3.5. *Let T be a tree with root u, let $\gamma(T) = k$ and let x be a k-critical vertex. If y, z are children of x such that for all children w of x $(w \neq y)$, $\Gamma(T[y]) \geqq \Gamma(T[z]) \geqq \Gamma(T[w])$ then $\gamma(T[u, y, z]) < k$ and $\Gamma(T) = [k] \cup \Gamma(T[u, y, z])$.*

Proof. By definition of $\Gamma(T)$, x has children v, w such that $\Gamma(T) = [k] \cup \Gamma(T[u, v, w])$. By Corollaries 3.1 and 3.2, $\Gamma(T[u, y, z]) \leqq \Gamma(T[u, v, w])$. Hence $\Gamma(T) = [k] \cup \Gamma(T[u, y, z])$. $\square$

Proof of Theorem 3.2. Let T be a tree with root u having children $v_1, \cdots, v_d$, let $S_i = \Gamma(T[v_i])$, for all i $(1 \leqq i \leqq d)$, and let $S_1 \geqq \cdots \geqq S_d$. We want to show that *Gamma* $(S_1, \cdots, S_d) = \Gamma(T)$. We first prove the correctness for the special cases $(d \leqq 1)$ and $(d \geqq 2 \wedge \delta(T) = 1)$. When $d \leqq 1$, *Gamma* $(\cdot)$ is given by one of lines [3], [5] or [7].

line [3]. $d = 0$. This means that T consists of a single vertex and by definition $\Gamma(T) = [0]$.

line [5]. $d = 1 \wedge \min S_1 \neq 0$. Let $S_1 = \Gamma(T[v_1]) = [a_1, \cdots, a_r]$ and let x_i, y_i, z_i $(1 \leqq i < r)$ be the vertices referred to in the definition of $\Gamma(\cdot)$. Let $H =$

$T[u, y_1, z_1, \cdots, y_{r-1}, z_{r-1}]$ and note that $\delta(H) = a_r$. Since $a_r > 0$, H has no a_r-critical vertex, and by Theorem 3.1 $\gamma(H) = a_r$. Now, let $J = T[u, y_1, z_1, \cdots, y_{r-2}, z_{r-2}]$ and note that $\delta(J) = a_{r-1}$. Since $a_{r-1} > 0$, the only possible a_{r-1}-critical vertex in J is x_{r-1}, and since $\gamma(H) = a_r < a_{r-1}$, $\gamma(J) = a_{r-1}$ by Theorem 3.1. Continuing in this fashion yields $\Gamma(T) = \Gamma(T[v_1])$.

line [7]. $d = 1 \wedge \min S_1 = 0$. Let $S_1 = \Gamma(T[v_1]) = [a_1, \cdots, a_r]$ and let x_i, y_i, z_i $(1 \leqq i < r)$ be the vertices referred to in the definition of $\Gamma(\cdot)$. Let $w = \min\{x > 0 | x \notin S_1\}$ and let $a_i = w - 1$. Since $a_r = 0$, $T[v_1, y_1, z_1, \cdots, y_{r-1}, z_{r-1}]$ consists of a single vertex. Obviously $\gamma(T[u, y_1, z_1, \cdots, y_{r-1}, z_{r-1}]) = 1$. For all j $(i \leqq j \leqq r)$, $\gamma(T[u, y_1, z_1, \cdots, y_{j-1}, z_{j-1}]) = a_{j-1} + 1$, by repeated applications of Theorem 3.1. Since $w > 0$ and $w \notin S_1$, $T[u, y_1, z_1, \cdots, y_{i-1}, z_{i-1}]$ has no w-critical vertex. For $1 \leqq j < i$, x_j is an a_j-critical vertex in $T[u, y_1, z_1, \cdots, y_{j-1}, z_{j-1}]$. Thus $\Gamma(T) = [a_1, \cdots, a_{i-1}, w]$ as claimed.

When $(d \geqq 2 \wedge \delta(T) = 1)$ T is a subtree of the tree shown in Fig. 15. If $d = 2$ then the values of $\bar{S}_1$ and $\bar{S}_2$ computed in lines [10] and [11] are correct, since we have established the correctness of *Gamma* $(\cdot)$ for $d < 2$. In particular $\bar{S}_1 = [1]$, $\bar{S}_2 = [0]$ and hence the value of k computed in line [12] is 1. If $\Gamma(T[v_1]) = [1]$ then v_1 has one child and *Gamma* will return $[1, 0]$ at line [16] which is correct. If $\Gamma(T[v_1]) = [1, 0]$ then v_1 has two children and *Gamma* will return [2] at line [20] which is correct. Thus *Gamma* is correct if $d = 2$ and $\delta(T) = 1$. Consequently, when $d = 3$ and $\delta(T) = 1$, the values of $\bar{S}_1$ and $\bar{S}_2$ computed in lines [10] and [11] are correct. In particular $\bar{S}_1 = [1, 0]$ and $\bar{S}_2 = [1]$. In this situation, *Gamma* returns [2] at line [14] which is correct.

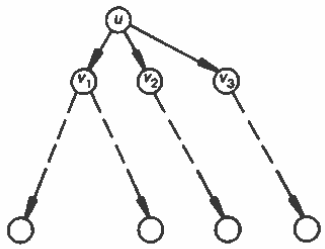

FIG. 15. *Special case of Theorem* 3.2—$d \geqq 2 \wedge \delta(T) = 1$.

The proof now proceeds by induction on d and $\delta(T)$. Assume that *Gamma* is correct for all trees T' in which either the root has fewer than d children or the root has d children but $\delta(T) < k$. Thus, the values of $\bar{S}_1$ and $\bar{S}_2$ computed in lines [10] and [11] are equal to $\Gamma(T[u, v_1])$ and $\Gamma(T[u, v_1, v_2])$ respectively, and the value of k computed in line [12] is equal to $\delta(T)$ by Corollary 3.3. Now considering each of the return statements in lines [14], [16], [17], [20].

line [14]. $d \geqq 2 \wedge k = d(T) \wedge k = \gamma(T[u, v_1, v_2])$. Since $k = \gamma(T[u, v_1, v_2])$, $k = \gamma(T[u, v_1])$ and by Corollary 3.4 u is k-critical. By Theorem 3.1, $\gamma(T) = k + 1$ and since T contains no $(k+1)$-critical vertex, $\Gamma(T) = [k+1]$.

line [16]. $d \geqq 2 \wedge k = \delta(T) \wedge k > \gamma(T[u, v_1, v_2]) \wedge T[v_1]$ does not contain a k-critical vertex $\wedge$ u is k-critical. By Theorem 3.1, $\gamma(T) = k$, and by Corollary 3.5, $\Gamma(T) = [k] \cup \Gamma(T[u, v_1, v_2])$.

line [17]. $d \geqq 2 \wedge k = \delta(T) \wedge k > \gamma(T[u, v_1, v_2]) \wedge T$ does not contain a k-critical vertex. By Theorem 3.1, $\gamma(T) = k$, and since there is no k-critical vertex, $\Gamma(T) = [k]$.

line [20]. $d \geqq 2 \wedge k = \delta(T) \wedge k > \gamma(T[u, v_1, v_2]) \wedge T[v_1]$ contains a k-critical vertex $\wedge$ u is k-critical. By Theorem 3.1, $\gamma(T) = k + 1$, and since T contains no $(k+1)$-critical vertex, $\Gamma(T) = [k+1]$.

At line [21], we have $d \geqq 2 \wedge \Gamma(T[u, v_1]) < k \wedge T[v_1]$ contains a k-critical vertex. Let x be the k-critical vertex in $T[v_1]$ and let y, z be children of x such that $\Gamma(T[v_1, y, z]) = S_1 - [k]$. Since $\delta(T[u, y, z]) < k$, the value of H computed in line 21 is $\Gamma(T[u, y, z])$ by the induction hypothesis. Considering the return statements in lines [22] and [23].

line [22]. $\Gamma(T[u, y, z]) = k$. By Theorem 3.1, $\gamma(T) = [k+1]$, and since T contains no $(k+1)$-critical vertex, $\Gamma(T) = [k+1]$.

line [23]. $\Gamma(T[u, y, z]) < k$. By Theorem 3.1, $\gamma(T) = [k]$ and by Corollary 3.5, $\Gamma(T) = [k] \cup \Gamma(T[u, y, z])$.

Thus, we have shown that the procedure *Gamma* returns the correct value, $\Gamma(T[u])$. □

The time required to execute *Label* $(\cdot)$ is proportional to n times the time required to execute *Gamma* $(\cdot)$. Excluding the recursive calls to *Gamma* at lines [10], [11] and [21], the time required to execute *Gamma* is $O(d \log n)$. (This follows from Corollary 2.4.) Note that the recursive call at line [11] can be ignored since the computation performed there is actually a subset of the computation made at line [10]. Consequently we can express the number of recursive calls required to compute *Gamma* $(S_1, \cdots, S_d)$ using the recurrence $M(k, d) \leqq M(k, d-1) + M(k-1, d)$ where $k = \max S_1 \cup \cdots \cup S_d$. This follows from the observation that in line [10] the number of subtrees is reduced by one and in line [21] the $\max S_1 \cup \cdots \cup S_d$ is reduced by at least one. Of course this is the defining recurrence for the binomial coefficients. Using the boundary conditions $M(k, 1) = M(0, d) = 1$ yields

$$M(k, d) \leqq \binom{k+d-1}{d-1}.$$

Using Corollary 2.4 we can show that for fixed d, $M(k, d) \leqq O((\log n)^{d-1})$. This means that the time complexity of *Gamma* is $O((\log n)^d)$. If we select a degree one vertex as the root of T then $d \leqq D-1$ where D is the maximum vertex degree for the entire tree. Hence the time required to execute *Label* is $O(n(\log n)^{D-1})$.

4. A cutwidth minimization algorithm for degree three trees. Megiddo et al. [25] describe an $O(n \log n)$ algorithm for determining the search number of a tree. Megiddo [24] shows how to reduce this time bound to $O(n)$. As a direct consequence of Corollary 2.3 these algorithms can be used to determine the cutwidth of a degree three tree, although of course they will not directly yield the optimal layout. In this section we describe an $O(n \log n)$ algorithm for degree three trees that is based on the general algorithm presented in § 3. This algorithm is simpler than the search number algorithm given in [25] and can be used to obtain an optimal layout. Furthermore, we can use it to reduce the complexity of the degree d algorithm by a factor of $\log n$.

Let T be a tree with root u having children v_1, v_2. Figure 16 gives a procedure *Gamma2* for computing $\Gamma(T)$ from $\Gamma(T[v_1])$ and $\Gamma(T[v_2])$. Notice that there is no degree restriction on $T[v_1]$ or $T[v_2]$. The correctness of the algorithm is established by the following theorem.

THEOREM 4.1. *Let T be a tree with root u having children v_1, v_2 and let $S_1 = \Gamma(T[v_1])$, $S_2 = \Gamma(T[v_2])$. Gamma2$(S_1, S_2) = \Gamma(T)$.*

Proof. The correctness of the assertion at line [13] of Fig. 16 follows from Theorem 3.2. Let $k = \max H_1$ and note that $k \leqq \gamma(T) \leqq k+1$. Consider three cases.

Case 1. $H_1 \cap H_2 = \emptyset$. Let $h = \max\{\min H_1, \min H_2\}$. We claim that $\Gamma(T) = [i \geqq h | i \in H_1 \cup H_2]$. The proof is by induction on $|H_1| + |H_2|$. (Basis) Assume $H_1 = [k]$. Then $T[v_1]$ contains no k-critical vertex and since $H_1 \geqq H_2$ and $H_1 \cap H_2 = \emptyset$, $\max H_2 <$

```
[1]  procedure Gamma2(S1,S2)
[2]      {Relabel if necessary so that S1 ≥ S2.}
[3]      if min S1 ≠ 0 then H1←S1
[4]      else begin
[5]          j←min {i > 0 | i ∉ S1}
[6]          H1←[j] ∪ {i > j | i ∈ S1}
[7]          end
[8]      if min S2 ≠ 0 then H2←S2
[9]      else begin
[10]         j←min {i > 0 | i ∉ S2}
[11]         H2←[j] ∪ {i > j | i ∈ S2}
[12]         end
[13]     {H1←Γ(T[u,v2]),H2←Γ(T[u,v1])}
[14]     if H1 ∩ H2=∅ then begin
[15]         h←max {min H1,min H2}
[16]         return {i ≥ h | i ∈ H1 ∪ H2}
[17]         end
[18]     h←max H1 ∩ H2
[19]     if h=min H1=min H2 then
[20]         return {0} ∪ {i ≥ h | i ∈ H1 ∪ H2}
[21]     i←min{j > h | j ∉ H1 ∪ H2}
[22]     return {i} ∪ {j > i | j ∈ H1 ∪ H2}
[23]     end
```

FIG. 16. *Procedure for computing* $\Gamma(T)$ *from* $\Gamma(T[v_1])$ *and* $\Gamma(T[v_2])$.

k. Hence u is not k-critical either and $\Gamma(T)=[k]$ as claimed. (Induction) If $|H_1|>1$ then $T[v_1]$ contains a k-critical vertex x with children y, z such that $\Gamma(T[v_1, y, z])=H_1-[k]$. By the induction hypothesis $\Gamma(T[u, y, z])=[i\geqq h|i\in(H_1-[k])\cup H_2]$. Since the largest integer in this sequence is less than k it follows from Corollary 3.5 that $\Gamma(T)=[k]\cup\Gamma(T[u, y, z])$.

Case 2. $H_1\cap H_2=[\min H_1]=[\min H_2]$. Let $h=\min H_1$. We claim that $\Gamma(T)=[0]\cup[i\geqq h|i\in H_1\cup H_2]$. The proof is by induction on $|H_1|+|H_2|$. (Basis) Assume $H_1=H_2=[h]$. Then neither $T[v_1]$ nor $T[v_2]$ contains an h-critical vertex. However u is h-critical and hence by Theorem 3.2 $\Gamma(T)=[h, 0]$ as claimed. (Induction) If $|H_1|+|H_2|>2$ then $T[v_1]$ has a k-critical vertex x with children y, z such that $\Gamma(T[v_1, y, z])=H_1-[k]$. By the induction hypothesis

$$\Gamma(T[u, y, z])=[0]\cup[i\geqq h|i\in(H_1-[k])\cup H_2].$$

Since the largest integer in this sequence is less than k it follows from Corollary 3.5 that $\Gamma(T)=[k]\cup\Gamma(T[u, y, z])$.

Case 3. $H_1\cap H_2\neq\varnothing\wedge(H_1\cap H_2\neq[\min H_1]\vee(H_1\cap H_2\neq[\min H_2])$. Let $h=\max H_1\cap H_2$ and $i=\min[j>h|j\notin H_1\cup H_2]$. We claim that $\Gamma(T)=[i]\cup[j>i|j\in H_1\cup H_2]$. We consider two subcases.

Subcase 3a. $i>k$. In fact, in this case, $i=k+1$ since $k=\max H_1>\max H_2$. We claim that $\Gamma(T)=[k+1]$. The proof is by induction on $k-h$. (Basis) Assume $k-h=0$. Note that $T[v_1]$ contains a k-critical vertex. Since $k\in H_2$ it follows that u is also k-critical, and by Theorem 3.2, $\Gamma(T)=[k+1]$ as claimed. (Induction) Assume $k-h>0$. Note that H_1 contains a k-critical vertex x with children y, z such that $\Gamma(T[v_1, y, z])=H_1-[k]$. By the induction hypothesis $\Gamma(T[u, y, z])=[k]$ and thus by Theorem 3.2, $\Gamma(T)=[k+1]$.

Subcase 3b. $i<k$. The proof is by induction on $k-i$. (Basis) Assume that $k-i=1$. Then $T[v_1]$ contains a k-critical vertex with children y, z such that $\Gamma(T[v_1, y, z])=H_1-[k]$. By Subcase 3a, $\Gamma(T[u, y, z])=[i]$. Applying Theorem 3.2 yields $\Gamma(T)=[k, i]$. (Induction) Again $T[v_1]$ contains a k-critical vertex x with children y, z such that $\Gamma(T[v_1, y, z])=H_1-[k]$. By the induction hypothesis $\Gamma(T[u, y, z])=[i]\cup[j>i|j\in(H_1-[k])\cup H_2]$. By Theorem 3.2 then $\Gamma(T)=[k]\cup\Gamma(T[u, y, z])$ as claimed. □

For any degree three tree T we can determine the cutwidth by applying *Gamma*2 from the bottom up. Each call to *Gamma*2 requires time $O(|S_1|+|S_2|)\leqq O(k)$ where $k=\gamma(T)$. By Corollary 2.4 $k=O(\log n)$, hence the cutwidth of T can be determined in time $O(n\log n)$.

The procedure *Gamma*2 can also be used to speed up the degree d algorithm. By calling *Gamma*2 whenever the current vertex has two children, one step of recursion is eliminated, reducing the time required to execute *Gamma* by a factor of log n. This yields an $O(n\ (\log n)^{d-2})$ algorithm for determining the cutwidth of a degree d tree. In [24], Megiddo gives a linear time algorithm for determining the search number of a tree. The technique used there can also be used to give a linear time version of procedure *Gamma*2. This in turn, can be used to reduce the complexity of the general algorithm by another factor of log n.

It is also worth noting in passing that almost all random trees have a maximum vertex degree that is $O(\log n/\log\log n)$ [29]. Using this one can show that the general cutwidth minimization algorithm runs in time $O(n^{2+\varepsilon})$ on randon trees where ε is any positive constant.

5. A layout algorithm. The labeling algorithms described in the previous sections produce a label $\lambda(x)=\Gamma(T[x])$ for every vertex x in T. Using these labels one can produce an optimal layout of T. The basic idea is contained in the proof of Theorem 2.1. If $\gamma(T)=k$ and T has no k-critical vertex then every vertex x has a child y such that $\gamma(T[x, y])<k$. Consequently one can construct a path $v_1, \cdots, v_s$ where v_1 is the root of T, v_s is a leaf and $\gamma(T[v_i, v_{i+1}])<k$ for $1\leqq i<s$. This is illustrated in Fig. 17. Once we have found this path we apply the layout algorithm recursively to the subtrees $T[v_i, v_{i+1}]$. If T does have a k-critical vertex x then one can construct a similar path $v_1, \cdots, v_s$. In this case v_1 and v_s are both leaves and x is contained in the path.

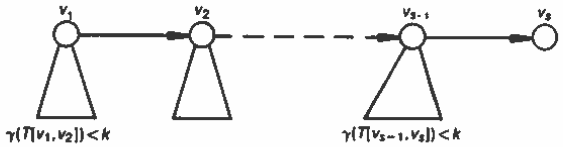

FIG. 17. *Motivation for layout algorithm.*

```
[1]  procedure Layout (T,λ,k,σ,pos)
[2]      r← the root of T
[3]      if r is the only vertex in T then begin
[4]          σ(r)←pos
[5]          pos←pos + 1
[6]          return
[7]          end
[8]      if r has a child x such that λ(x) > ⌊k⌋ then begin
[9]          y← the vertex that satisfies λ(y) > ⌊k⌋
[10]             and for all children z of y, λ(z) ≤ ⌊k⌋
[11]     else y←r
[12]     <v1, . . . , vs> ←chain (T,y,λ)
[13]     for each vertex x in T do
[14]         λ(x)←λ(x) − ⌊k⌋
[15]     Layout (T(v1,v2),λ,k−1,σ,pos)
[16]     for j = 2 to s − 1 do
[17]         Layout (T(vj,vj−1,vj+1),λ,k−1,σ,pos)
[18]     Layout (T(vs,vs−1),λ,k−1,σ,pos)
[19]     return
[20]     end
```

FIG. 18. *Layout procedure.*

A procedure for computing the layout from the labels is shown in Fig. 18. T is a tree for which a cutwidth k layout is required and λ is the set of labels produced by one of the labeling algorithms from the previous sections. Upon return from *Layout* $(T, \lambda, k, \sigma, i)$ each vertex in T is assigned a unique position in the layout σ starting at position i. More precisely $\forall x \in T, i \leqq \sigma(x) < i+|T|$ and $\forall x, y \in T, \sigma(x) \neq \sigma(y)$. Note that the notation for undirected trees is used to specify the subtrees in the recursive calls to *Layout*. The procedure *chain* $(\cdot)$ locates the path discussed above, and is shown in Fig. 19. Figure 20 shows the layout obtained for the tree whose labels were given in Fig. 20.

```
[1]  procedure chain(T,v,λ)
[2]      Let v_0=v and let v_0, ..., v_r be a maximum length
[3]      path in T satisfying the following conditions for 0 ≤ i < r.
[4]          (a) v_i is the parent of v_{i+1} and
[5]          (b) for all children x of v_i, λ(v_{i+1}) ≥ λ(x).
[6]      if v has less than two children then
[7]          return <v_1, ..., v_r>
[8]      Let u_1 be a child of v that satisfies u_1 ≠ v_1 and
[9]      λ(u_1) ≥ λ(x) for all children x of v such that x ≠ v_1.
[10]     Let u_1, ..., u_s be a maximum length path in T satisfying
[11]     the following conditions for 1 ≤ i < s.
[12]         (a) u_i is the parent of u_{i+1} and
[13]         (b) for all children x of u_i, λ(u_{i+1}) ≥ λ(x).
[14]     return <v_r, ..., v_1, v, u_1, ..., u_s>
[15]     end
```

FIG. 19. *Procedure for finding a chain.*

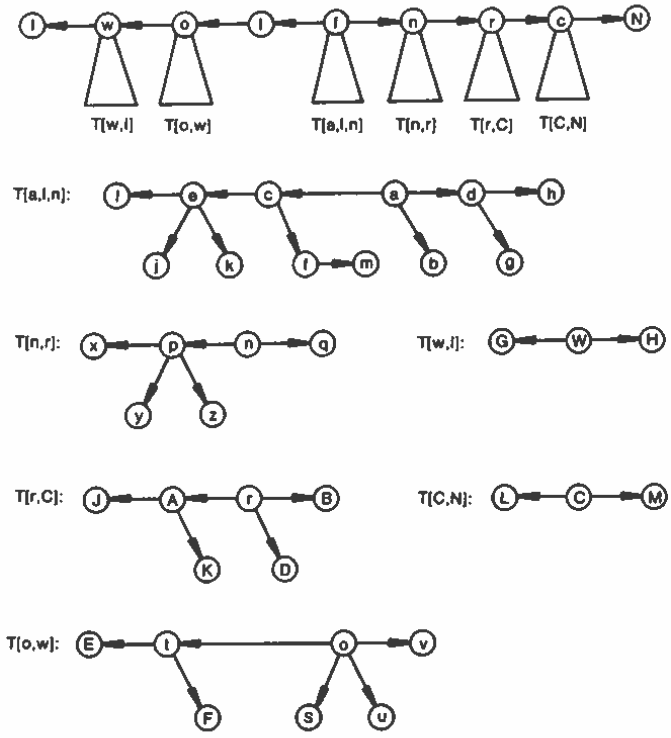

FIG. 20. *Example of layout procedure.*

6. Open problems. Until recently, the main open question was whether the MIN CUT problem could be solved in polynomial time for unrestricted trees. Yannakakis [42] has now reported a polynomial time algorithm for this problem.

To our knowledge, no good approximation algorithms have been proposed for the MIN CUT problem on graphs. At the same time there is no evidence that such algorithms do not exist. No effort has been made to bound the cutwidth of random graphs. This is a necessary first step in understanding the probabilistic performance of cutwidth minimization algorithms.

A natural extension of the MIN CUT problem is to edge weighted graphs; the weight of a cut being the sum of the weights of all edges in the cut. There is a straightforward log space reduction from the PARTITION problem to this weighted MIN CUT problem on trees. Hence even for trees the problem is NP-complete. However it is not known to be strongly NP-complete. In particular the complexity of the MIN CUT problem on trees with edge weights in $\{1, 2\}$ is open.

Perhaps the most surprising aspect of current research on the MIN CUT problem is the connections that have been found to seemingly unrelated problems. The equivalence of cutwidth and black/white pebble demand on degree three trees in which the sink has in-degree one was completely unexpected and we thank Nick Pippenger for pointing it out to us. As we noted above, this connection not only yields an exact pebbling algorithm for degree three trees, it also yields approximation algorithms for pebbling degree d trees. A closer study of this relationship could deepen our understanding of both problems. Another surprise was the connection between cutwidth and search number. We have shown their equivalence for degree three trees. Makedon and Sudborough [22] have strengthened this to degree three graphs. Recently Chung [4] has shown a connection between the topological bandwidth and the cutwidth of trees. This could lead to efficient algorithms for the topological bandwidth problem.

Appendix. The proof of Theorem 3.3 requires the following two lemmas.

LEMMA 3.1. *Let R be the tree consisting of the single vertex u, let S and S' be trees with roots v, v' and let $T = R \cdot S$, $T' = R \cdot S'$. $\Gamma(S) \leqq \Gamma(S') \Rightarrow \gamma(T) \leqq \gamma(T')$.*

Proof. If $\gamma(S) < \gamma(S')$ then the result is immediate. Assume then that $\gamma(S) = \gamma(S') = k$. Note that $k = \delta(T) = \delta(T')$. The proof proceeds by induction on $|\Gamma(S)|$. If $\Gamma(S) = [0]$, then $\gamma(T) = 1 \leqq \gamma(T')$. Assume then that $k > 0$. If $\Gamma(S) = [k]$ then S has no k-critical vertex. Consequently T has no k-critical vertex, and by Theorem 3.1, $\gamma(T) = k \leqq \gamma(T')$. This establishes the basis of the induction.

Now assume that S and S' violate the lemma, where $|\Gamma(S)| > 1$. Also, assume that there is no pair of trees H, H' that also violate the lemma where $|\Gamma(H)| < |\Gamma(S)|$. We continue to let $\gamma(S) = \gamma(S') = k$. Note that since $\Gamma(S) \leqq \Gamma(S')$, $|\Gamma(S')| > 1$. Hence, S contains a k-critical vertex x with children y, z such that $\Gamma(S[v, y, z]) = \Gamma(S) - [k]$ and S' contains a k-critical vertex x' with children y', z' such that $\Gamma(S'[v', y', z']) = \Gamma(S') - [k]$. This is illustrated in Fig. 21. Since S and S' violate the lemma, $\gamma(T) > \gamma(T')$. In fact, we must have $\gamma(T) = k+1$ and $\gamma(T') = k$. Now, since x is the only k-critical vertex in T, $\gamma(T) = k+1 \Rightarrow \gamma(T[u, y, z]) \geqq k$, by Theorem 3.1. Also, since x' is k-critical, $\gamma(T') = k \Rightarrow \gamma(T'[u, y', z']) < k$. Thus $\gamma(T[u, y, z]) > \gamma(T'[u, y', z'])$. Since $\Gamma(S) \leqq \Gamma(S')$, $\Gamma(S[v, y, z]) \leqq \Gamma(S'[v', y', z'])$. Hence $S[v, y, z]$ and $S'[v', y', z']$ also violate the lemma, giving the desired contradiction. □

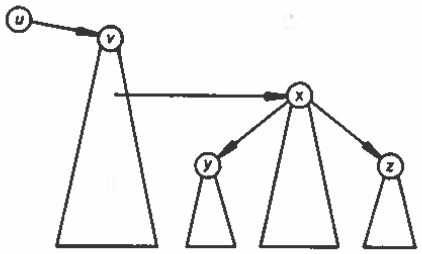

FIG. 21. *Illustration for Lemma* 3.1.

LEMMA 3.2. *Let R, S, S' be trees with roots u, v, v' and let $T = R \cdot S$, $T' = R \cdot S'$. $\Gamma(S) \leqq \Gamma(S') \Rightarrow \gamma(T) \leqq \gamma(T')$.*

Proof. If $\gamma(S) < \gamma(S')$ then we can take any optimal layout of T' and substitute S for S' without increasing the cutwidth of the layout. Assume then that $\gamma(S) = \gamma(S') = k$ and let $\gamma(R) = m$. The proof is by induction on $|R \cdot S|$. The lemma is clearly true if S consists of a single vertex. This together with Lemma 3.1 provides the basis of the induction. Assume that R, S, S' violate the lemma and that there are no H, J, J' that also violate the lemma, where $|H \cdot J| < |R \cdot S|$. We consider three cases.

Case 1. $m<k$. Note that $\delta(T)=k$, and since R, S, S' violate the lemma, $\gamma(T)=k+1$, $\gamma(T')=k$. If S has no k-critical vertex then neither does T and by Theorem 3.1 $\gamma(T)=k$, which is a contradiction. If, on the other hand, S does have a k-critical vertex then so does S'. Assume then that S has a k-critical vertex x with children y, z such that $\Gamma(S[v, y, z])=\Gamma(S)-[k]$ and S' has a k-critical vertex x' with children y', z' such that $\Gamma(S'[v', y', z'])<\Gamma(S')-[k]$. Now, since x is the only k-critical vertex in T, $\gamma(T)=k+1\Rightarrow\gamma(T[u, y, z])\geqq k$ by Theorem 3.1. Also, since x' is k-critical, $\gamma(T')=k\Rightarrow\gamma(T'[u, y', z'])<k$. Thus, $\gamma(T[u, y, z])>\gamma(T'[u, y', z'])$. Since $\Gamma(S[v, y, z])=\Gamma(S)-[k]$ and $\Gamma(S'[v', y', z'])=\Gamma(S')-[k]$, $R, S[v, y, z], S'[v', y', z']$ also violate the lemma giving a contradiction.

Case 2. $m>k$. Note that $\delta(T)\leqq m$, and since R, S, S' violate the lemma, $\gamma(T)=m+1$ and $\gamma(T')=m$. If R has no m-critical vertex then the only possible m-critical vertex in T is u. But if u is not m-critical in R then u has two children r, s in T such that $\gamma(T[u, r, s])<m$. Then, by Theorem 3.1, $\gamma(T)=m$, which is a contradiction. Thus R must contain an m-critical vertex x. Since x is also m-critical in T', $\gamma(T')=m\Rightarrow$that x has children y, z such that $\gamma(T'[u, y, z])<m$. If x is the only m-critical vertex in T, then by Theorem 3.1, $\gamma(T)=m+1\Rightarrow\gamma(T[u, y, z])\geqq m$. If x is not the only m-critical vertex in T, then the other must be u, and by the definition of m-criticality, $\gamma(T[u, y, z])\geqq m$ (since y and z are both in the same subtree of u). Thus $R[u, y, z], S, S'$ violate the lemma giving a contradiction.

Case 3. $m=k$. Note that $\delta(T)=k$. Since R, S, S' violate the lemma, $\gamma(T)=k+1$, $\gamma(T')=k$. Now, if u is k-critical in R, then by Theorem 3.1, $\gamma(T')=k+1$, a contradiction. If neither R nor S has a k-critical vertex then the only possible k-critical vertex in T is u. But since u is not k-critical in R, u has children r, s in T such that $\gamma(T[u, r, s])<k$. Then, by Theorem 3.1, $\gamma(T)=k$, which is a contradiction. Thus, if neither R nor S has a k-critical vertex, then neither does T, and again by Theorem 3.1, $\gamma(T)=k$. Thus either R or S must have a k-critical vertex. By Theorem 3.1, they cannot both have a k-critical vertex since, that would imply $\gamma(T')>k$. If S contains a k-critical vertex x then S' contains a k-critical vertex x' and we proceed as in Case 1. If R contains a k-critical vertex we proceed as in Case 2. □

Proof of Theorem 3.3. The proof is by induction on $|\Gamma(T)|$. Lemma 3.2 provides the basis Assume then that R, S, S' violate the theorem and that there are no H, J, J' that violate the theorem such that $|\Gamma(H\cdot J)|<|\Gamma(T)|$. By Lemma 3.2, $\gamma(T)\leqq\gamma(T')$, thus for $\Gamma(T)>\Gamma(T')$ to be true we must have $\gamma(T)=\gamma(T')=k$ and T must have some k-critical vertex x. We consider three cases.

Case 1. $x\in S$. In this case x has children y, z such that $\Gamma(S[v, y, z])=\Gamma(S)-[k]$. Since $\Gamma(S)\leqq\Gamma(S')$, S' must have a k-critical vertex x' with children y', z' such that $\Gamma(T'[u, y', z'])=\Gamma(T')-[k]$. Hence $\Gamma(T'[u, y', z'])<\Gamma(T)-[k]\leqq\Gamma(T[u, y, z])$ and $\Gamma(S[v, y, z])\leqq\Gamma(S')-[k]<\Gamma(T[u, y, z])\leqq\Gamma(S'[v', y', z'])$. Thus $R, S[v, y, z]$ and $S'[v', y', z']$ violate the theorem giving a contradiction.

Case 2. $x\in R-\{u\}$. In this case x is k-critical in both T and T' and has children y, z, y', z' such that $\Gamma(T[u, y, z])=\Gamma(T)-[k]$ and $\Gamma(T'[u, y', z'])=\Gamma(T')-[k]$. Since $\Gamma(T')<\Gamma(T)$, $\Gamma(T'[u, y', z'])<\Gamma(T[u, y, z])\leqq\Gamma(T[u, y', z'])$. Thus $R[u, y', z'], S, S'$ violate the theorem giving a contradiction.

Case 3. $x=u$. Since u is k-critical in T, it has no child y such that $\gamma(T[u, y])<k$. If u is not k-critical in T' then u has a child y' in T' such that $\gamma(T'[u, y'])<k$. If $y'\in R$ then $R[u, y'], S, S'$ violate Lemma 3.2. If $y'=v'$ then $\gamma(T[u, v])=k>\gamma(T'[u, v'])$, but this is clearly absurd, since $T[u, v]=R=T[u, v']$. Thus u is k-critical in both T and T'. Further u has children y, z, y', z' such that $\Gamma(T[u, y, z])=\Gamma(T)-[k]$ and $\Gamma(T'[u, y', z'])=\Gamma(T')-[k]$. Consider two subcases.

Subcase 3a. $y' \neq v' \neq z'$. Since $\Gamma(T') < \Gamma(T)$, $\Gamma(T'[u, y', z']) < \Gamma(T[u, y, z]) \leqq \Gamma(T[u, y', z'])$. Hence $R[u, y', z'])$, S, S' violate the theorem giving a contradiction.

Subcase 3b. $y' = v'$. Since $\Gamma(T') < \Gamma(T)$, $\Gamma(T'[u, v', z']) < \Gamma(T[u, y, z]) \leqq \Gamma(T[u, v, z'])$, but this is contradictory since $T'[u, v', z'] = T[u, v, z']$. □

Acknowledgments. The authors wish to thank John Ellis and Manrique Mata, participants in the algorithms and complexity seminar at Northwestern University for their valuable suggestions. We would also like to thank F. R. K. Chung for pointing out an error in an earlier version of Theorem 2.2.

REFERENCES

[1] M. A. BREUER, *Min-cut placement*, J. Design Automation and Fault Tolerant Computing, 1 (1977), pp. 343–362.

[2] F. R. K. CHUNG, *On linear arrangements of trees*, Bell Laboratories TM-80-1216-31, Murray Hill, NJ, 1980.

[3] ———, *Some problems and results on labelings of graphs*, in The Theory and Applications of Graphs, G. Chartrand, ed., John Wiley, New York, 1981, pp. 255–264.

[4] ———, *On the cutwidth and the topological bandwidth of a tree*, SIAM J. Alg. Disc. Meth., 6 (1985), to appear.

[5] D. DOLEV AND H. TRICKEY, *Embedding a tree on a line*, IBM Technical Report RJ3368, 1982.

[6] A. FELLER, *Automatic layout of low-cost quick turnaround random-logic custom* LSI *devices*, Proc. Thirteenth Design Automation Conference, 1976, pp. 79–85.

[7] M. J. FOSTER AND H. T. KUNG, *Recognizing regular languages with programmable building-blocks*, in VLSI-81 Conference, Aug. 1981.

[8] MICHAEL R. GAREY, DAVID S. JOHNSON AND L. J. STOCKMEYER, *Some simplified* NP*-complete graph problems*, Theoret. Computer Sci., 1 (1976), pp. 237–267.

[9] MICHAEL R. GAREY, R. L. GRAHAM, DAVID S. JOHNSON AND D. E. KNUTH, *Complexity results for bandwidth minimization*, SIAM J. Appl. Math., 34 (1978), pp. 477–495.

[10] MICHAEL R. GAREY AND DAVID S. JOHNSON, *Computers and Intractability—A Guide to the Theory of* NP*-Completeness*, W. H. Freeman, San Francisco, 1979.

[11] F. GAVRIL, *Some* NP*-complete problems on graphs*, Proc. 11th Conference on Information Sciences and Systems, John Hopkins Univ., Baltimore, MD, pp. 91–95.

[12] M. K. GOLDBERG AND I. A. KLIPKER, *Minimal placing of trees on a line*, Technical Report, Physico-Technical Institute of Low Temperatures, Academy of Sciences of Ukrainian SSR, 1976. (In Russian.)

[13] EITAN M. GURARI AND I. H. SUDBOROUGH, *Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement Problem*, J. Algorithms, to appear.

[14] A. S. LAPAUGH, *Recontamination does not help*, Technical Report, Computer Science Dept., Princeton Univ., Princeton, NJ, 1983.

[15] M. A. IORDANSKII, *Minimal numberings of the vertices of trees*, Soviet Math. Doklady (1974), pp. 1311–1315.

[16] T. LENGAUER, *Black-white pebbles and graph separation*, Acta Inform., 16 (1981), pp. 465–475.

[17] T. LENGAUER AND R. E. TARJAN, *The space complexity of pebble games on trees*, Inform. Processing Lett., 10 (1980), pp. 184–188.

[18] ———, *Asymptotically tight bounds on time-space trade-offs in a pebble game*, J. Assoc. Comput. Mach., 29 (1982), pp. 1087–1130.

[19] THOMAS LENGAUER, *Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees*, SIAM J. Alg. Disc. Meth., 3 (1982), pp. 99–113.

[20] A. D. LOPEZ AND H-F. S. LAW, *A dense gate matrix layout method for* MOS VLSI, IEEE Trans. Electronic Devices, ED-27 (1980), pp. 1671–1675.

[21] M. C. LOUI, *The space complexity of two pebble games on trees*, MIT Technical Report, MIT/LCS/TM-133, Massachusetts Institute of Technology, Cambridge, 1979.

[22] F. S. MAKEDON AND I. H. SUDBOROUGH, *Minimizing width in linear layouts*, Lecture Notes in Computer Science, 154, Springer-Verlag, New York, 1983, pp. 478–490.

[23] FILLIA S. MAKEDON, I. HAL SUDBOROUGH AND C. H. PAPADIMITRIOU, *Topological bandwidth*, in Proc. 8th Colloquium on Trees in Algebra and Programming, 1983; SIAM J. Alg. Disc. Meth., 6 (1985), to appear.

[24] N. MEGIDDO, *Linear time algorithm for search number in trees*, unpublished manuscript, April 1981.
[25] N. MEGIDDO, S. L. HAKIMI, MICHAEL R. GAREY, DAVID S. JOHNSON AND CHRISTOS H. PAPADIMITRIOU, *The complexity of searching a graph*, Proc. IEEE Foundations of Computer Science Symposium, 1981, pp. 376–385.
[26] F. MEYER AUF DER HEIDE, *A comparison of two variations of a pebble game on graphs*, Theoret. Comput. Sci., 13 (1981), pp. 315–322.
[27] T. OHTSUKI, H. MORI, E. S. KUH, T. KASHIWABARA AND T. FUJISAWA, *One-dimensional logic gate assignments and interval graphs*, IEEE Trans. Circuits and Systems, CAS-26 (1979), pp. 675–684.
[28] BURKHARD MONIEN AND I. H. SUDBOROUGH, *Bandwidth constrained* NP*-complete problems*, Proc. 11th ACM Symposium on Theory of Computing, 1981, pp. 207–217.
[29] JOHN W. MOON, *Counting labeled trees*, Canadian Mathematical Monographs 1, 112 (1970).
[30] CHRISTOS H. PAPADIMITRIOU, *The* NP*-completeness of the bandwidth minimization problem*, Computing, 16 (1976), pp. 263–270.
[31] T. D. PARSONS, *The search number of a connected graph*, Proc. Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing, 1978, pp. 549–554.
[32] ———, *Pursuit-evasion in a graph*, in Theory and Application of Graphs, Y. Alavi and D. R. Lick, eds., Springer-Verlag, New York, 1976, pp. 426–441.
[33] G. PERSKY, D. DEUTSCH AND D. SCHWEIKERT, LTX—*A minicomputer-based system for automated* LSI *layout*, J. Design Automation and Fault Tolerant Computing, 1 (1977), pp. 217–255.
[34] N. PIPPENGER, *Pebbling*, IBM Research Report RC8258, 1980.
[35] ———, *Advances in pebbling*, IBM Research Report RJ3466, 1982.
[36] ———, private communication.
[37] R. R. REDZIEJOWSKI, *On arithmetic expressions and trees*, Comm. ACM, 12 (1969), pp. 81–84.
[38] YOSSI SHILOACH, *A minimum linear arrangement algorithm for undirected trees*, this Journal, 8 (1979), pp. 15–32.
[39] L. STOCKMEYER, private communication to M. R. Garey and D. S. Johnson, 1974; see [10, p. 201].
[40] S. TRIMBERG, *Automating chip layout*, in IEEE Spectrum, 1982, pp. 38–45.
[41] A. WEINBERGER, *Large scale integration of* MOS *complex logic: a layout method*, IEEE J. Solid State Circuits, 2 (1967), pp. 182–190.
[42] MIHALIS YANNAKAKIS, *A polynomial algorithm for the min-cut linear arrangement of trees*, Proc. IEEE Symposium on the Foundations of Computer Science, 1983.
[43] H. YOSHIZAWA, H. KAWANISHI AND K. KANI, *A heuristic procedure for ordering* MOS *arrays*, Proc. Design Automation Conference, 1975, pp. 384–393.