SIAM J. COMPUT.
Vol. 17, No. 5, October 1988

012

A LINEAR ALGORITHM FOR TOPOLOGICAL BANDWIDTH IN DEGREE-THREE TREES*

ZEVI MILLER†

Abstract. Let G be a graph, and f a one-to-one map of G into the positive integers. The *bandwidth of* G is $B(G) = \min_f \max\{|f(x)-f(y)|: xy \in E(G)\}$, where the max is taken over all edges xy in G and the min over all maps f. $B(G)$ is related to the matrix bandwidth $B(M)$ for a symmetric matrix M, and knowledge of the latter parameter is important for the efficient execution of certain matrix operations. The problem of determining $B(G)$ for arbitrary G was shown by Papadimitriou to be NP-complete, and it was subsequently proved NP-complete even when $G \in \Omega$, where Ω is the set of trees with maximum degree 3. Let $B^*(G)$ be the minimum possible bandwidth of any subdivision of G, i.e., any graph obtained from G by inserting degree-2 points along edges of G. We present an $O(n)$ algorithm for computing $B^*(G)$ when $G \in \Omega$.

Key words. topological bandwidth

AMS(MOS) subject classification. 05C99

1. Introduction. Let $G = (V, E)$ be a finite graph with no loops or multiple edges. A one-to-one map $f: V(G) \to \mathbb{Z}$ of the vertex set of G into the integers will be called a *layout* of G. We let $|f| = \max\{|f(v)-f(w)|: vw \in E(G)\}$, and we define the *bandwidth* of G, $B(G)$, to be $B(G) = \min_f |f|$. For convenience, we adopt the convention $B(K_1) = 1$, where K_1 is the singleton graph.

The problem of determining $B(G)$ for any G has importance in the theory of sparse matrices. Given an $n \times n$ matrix $A = (a_{ij})$, define a graph $G(A)$ by letting $V(G(A)) = \{1, 2, \cdots, n\}$ and declaring that ij is an edge if and only if $a_{ij} \neq 0$ or $a_{ji} \neq 0$. Suppose we can find a symmetric permutation of the rows and columns of A so that all nonzero entries lie within the first k superdiagonals or the first k subdiagonals about the main diagonal. Then clearly $B(G(A)) \leqq k$. Conversely if $B(G(A)) \leqq k$, then the stated permutation exists. The problem of finding the permutation with the smallest k, equivalently, finding $B(G(A))$, is important in efficiently operating on and storing matrices, e.g., for Gaussian elimination, systolic processes, etc. Relevant work can be found in [1], [3], [9].

Recently the importance of bandwidth for complexity theory has been demonstrated in the work of Sudborough and Monien [1], [15]. The main kind of result has been that when certain well-known graph problems which are complete for some complexity class C are restricted to graphs with bounded bandwidth, the resulting problem is complete for space-bounded or simultaneous space- and time-bounded subclasses of C. Of course the bandwidth problem for arbitrary graphs [13], and even for trees of maximum degree 3 [6], is known to be NP-complete.

The problem of given a graph G to determine whether $B(G) \leqq k$ when k is fixed was proved polynomial time solvable in [14], with an improvement in [7].

The subject of this paper is the following topological version of bandwidth, attributed in [3] to R. Graham. For any graph G, let $S(G)$ be the set of all subdivisions of G, i.e., the set of all graphs obtainable from G by finitely many compositions of the operation, called elementary subdivision, of replacing some edge xz by two edges xy and yz, where y is a vertex not previously in the graph. Obviously any $H \in S(G)$

* Received by the editors July 30, 1984; accepted for publication (in revised form) October 20, 1987. This work was supported in part by Office of Naval Research grant N0000 14-85-K-0621.

† Department of Mathematics and Statistics, Miami University, Oxford, Ohio 45056.

is homeomorphic with G and hence has the same underlying topology as G. The *topological bandwidth* of G, $B^*(G)$, is defined by $B^*(G) = \min\{B(H): H \in S(G)\}$.

Topological bandwidth is related to sparse matrices as follows. Given the system $Ax = b$, construct an equivalent system by replacing some term $a_{ij}x_j$ by the new variable y and add a new equation $a_{ij}x_j = y$. This operation is equivalent to inserting a point of degree 2 in the edge ij of $G(A)$. Clearly $B^*(G(A))$ is the smallest k such that for some system $A'x = b'$ equivalent to $Ax = b$ by a sequence of the above operations there is a permutation of the rows and columns which yields a $2k+1$ band form. The new system may be more efficient to work with if it is not too large, i.e., if the number of additional degree-2 vertices is not too large.

Another application is in regarding a graph G as a circuit with gates (corresponding to points of G) laid out linearly for automation purposes [12], [16]. Inserting degree-2 points along edges of G corresponds to inserting drivers along wires of the circuit. Hence $B^*(G)$ is the minimum length of the longest connection over all linear layouts allowing drivers. This minimum gives a lower bound on the time delay in transmission between elements of the circuit.

The cutwidth, $cw(G)$, of a graph G is defined as the minimum, over all layouts f, of $\max_i |\{uv \in E(G): f(u) \leqq i < f(v)\}|$. The relation between $B^*(T)$ and $cw(T)$ for trees T was explored by Chung in [5]. It was shown that $cw(T)$ and $B^*(T)$ can be arbitrarily far apart, but that $cw(T) \leqq B^*(T) - \log_2 B^*(T) + 2$. Recently Yannakakis has found an $O(n \log n)$ algorithm for $cw(T)$ for arbitrary trees T [17].

The relation between $cw(G)$ and $B^*(G)$ was also investigated by Makedon, Papadimitriou, and Sudborough in [10]. They show that for any graph G we have $B^*(G) \leqq mcw(G) + 1$, where mcw is the "modified cutwidth" considered in [10] and defined earlier by Lengauer in [18]. Since for any graph G we have $mcw(G) \leqq cw(G) - 1$, it follows that $B^*(G) \leqq cw(G)$. Makedon et al. also show that for any graph with maximum degree 3, $B^*(G) = mcw(G) + 1$. Thus an algorithm for topological bandwidth in trees of maximum degree 3 is also an algorithm for mcw in such trees.

The main result of this paper is an $O(n)$ algorithm for computing $B^*(T)$, where T is any tree of maximum degree 3. First we develop an $O(n \log n)$ algorithm which contains all the essential graph-theoretic ideas. Some additional data structure then leads to the $O(n)$ solution. Our result is a contrast to the NP-completeness of bandwidth for trees of maximum degree 3, and hence shows that topological structure "alone" is not the reason for this NP-completeness.

We remark that an $O(n \log n)$ algorithm for $B^*(T)$ in degree-3 trees is obtained independently by Makedon, Papadimitriou, and Sudborough, in [10]. They show that the topological bandwidth problem for arbitrary graphs is NP-complete, they characterize graphs with topological bandwidth two, and they explore relations with pebbling and searching.

We follow standard graph-theoretic notation, as may be found in [2] or [8] for example. In particular, if $G = (V, E)$ is a graph and H is a subgraph of G, then $G - H$ is the graph obtained from G by deleting all points of H and all edges of G incident on those points.

2. The main algorithm. Let P be a path joining two end vertices of T. A subtree α of T is called a *hanging from* P if it is one of the connected components of $T - P$. Thus each such α contains a unique vertex x_α such that $x_\alpha y \in E(T)$ for some $y \in P$. We denote by $Z(P)$ the set of all hangings from P. Let $\|P\|$, the *norm of* P *in* T, be defined by $\|P\| = 1 + \max\{B^*(\alpha): \alpha \in Z(P)\}$. To emphasize the role of T, we sometimes

write this as $\|P\|_T$. We let $K_{1,2}$ (respectively, $K_{1,3}$) be the tree with a central point joined to two (respectively, three) endpoints.

We begin with some lemmas.

LEMMA 1. *For any tree T there exists a layout of T satisfying $|f| = B(T)$ such that $f^{-1}(\min f(x))$ and $f^{-1}(\max f(x))$ are both end vertices of T.*

Proof. Let g be a layout of T such that $|g| = B(T)$. Let $p = g^{-1}(\min g(x))$ and $q = g^{-1}(\max g(x))$. We may suppose that at least one of p and q is not an end vertex (or leaf). Let P be the path in T joining p and q, and let Y_p (respectively, Y_q) be the set of branches at p (respectively, q) having empty intersection with P. Now define a new layout g_1 of T as follows:

$$g_1(x) = \begin{cases} g(x), & x \in V(T)\backslash[Y_p \cup Y_q], \\ 2g(p) - g(x), & x \in Y_p, \\ 2g(q) - g(x), & x \in Y_q. \end{cases}$$

Notice that g_1 just reflects Y_p and Y_q about p and q, respectively, and $|g_1| = |g|$. If $g_1^{-1}(\min g_1(x))$ and $g_1^{-1}(\max g_1(x))$ are both end vertices, then we are done. If not, then repeat the above process with g_1 in place of g. Eventually we obtain a layout for T of the required kind. □

A path P joining two end vertices of the tree T is called a *backbone of T*. If f is a layout of T such that $z_1 = f^{-1}(\min f(x))$ and $z_2 = f^{-1}(\max f(x))$ are both end vertices of T, then the backbone P joining z_1 and z_2 is called a *backbone of f*. We also say that f has backbone P.

Let $T^* \in S(T)$, and let P be a backbone of T. A layout f of T^* will be called a *t-layout* of T (t for topological). We call P an *optimal backbone of T* if there exists such an f and T^* satisfying

(i) $|f| = B^*(T)$, and

(ii) the path joining $f^{-1}(\min f(x))$ and $f^{-1}(\max f(x))$ corresponds to P under the subdivision operation (so that $f^{-1}(\min f(x))$ and $f^{-1}(\max f(x))$ are endpoints of T^* corresponding to the endpoints of P).

COROLLARY 1.1. *Every tree T has an optimal backbone.*

Proof. Let f be a t-layout of T with $|f| = B^*(T)$. By Lemma 1 we can find a t-layout g such that $g^{-1}(\min g(x))$ and $g^{-1}(\max g(x))$ are end vertices v_1 and v_2 (respectively) while $|g| = B^*(T)$. The path in T joining the vertices v_1 and v_2 is the required optimal backbone. □

Given a layout f of G and $H \subseteq G$, *the layout of H induced by f* is the layout $f_H : V(H) \to \{1, 2, \cdots, |V(H)|\}$ which maps the points of H in the same order as they are mapped by f.

LEMMA 2. *Let T be a nontrivial tree of maximum degree 3, P be an optimal backbone of T, and $Z(P) = \{H_1, H_2, \cdots, H_k\}$. Then $B^*(T) = \|P\| = 1 + \max B^*(H_j)$.*

Proof. We start with the lower bound $B^*(T) \geqq \|P\|$. Consider a t-layout of T, say $f: T^* \to \mathbb{Z}$ for some $T^* \in S(T)$, such that $|f| = B^*(T)$. Let $\bar{H}_i \in S(H_i)$, $1 \leqq i \leqq k$, be the hanging from P^* (the subdivision of P in T^*) corresponding to H_i. Let f_i be the layout of $\bar{H}_i$ induced by f. Clearly $|f_i| \geqq B(\bar{H}_i) \geqq B^*(H_i)$, so there exist $c, d \in V(H_i)$ (say with $f(c) < f(d)$) such that $f_i(d) - f_i(c) \geqq B^*(H_i)$. After translating f_i to correspond to the restriction of f to $\bar{H}_i$, we find that there is an edge $ab \in E(P^*)$ such that without loss of generality either

(i) $f(a) < f_i(c)$ and $f(b) > f_i(d)$, or

(ii) $f_i(c) < f(a) < f_i(d)$.

In either case we get $|f| \geqq 1 + B^*(H_i)$, so by the arbitrariness of i we get $|f| \geqq \|P\|$.

The bound $B^*(T) \leqq \|P\|$ follows from the layout indicated in Fig. 1. For each i, $1 \leqq i \leqq k$, we take a layout f_i of some $\bar{H} \in S(H_i)$ satisfying $|f_i| = B^*(H_i)$. Assuming without loss of generality that the H_i occur along P in indicial order, we place the layouts f_i along $\mathbb{Z}$ in the same order so that f_i and f_{i+1} have no overlap for all i. We then lay out some $P^* \in S(P)$ so that consecutive points on P^* are $\|P\|$ units apart, $\min\{P^*\} < \min f_1(x)$, and $\max\{P^*\} > \max f_k(x)$ (where we identify P^* with points of $\mathbb{Z}$). Of course this requires us to "squeeze in" points of P^* into the range of the layouts f_i. We make room for this by iteratively translating values $f_i(x)$ greater than the squeezed-in point one unit to the right. Since consecutive points on P^* are $\|P\|$ units apart, and since $B^*(H_i) < \|P\|$, we will never have to stretch any pair $f_i(x)$, $f_i(y)$, $xy \in E(\bar{H}_i)$, more than $\|P\|$ units apart. This defines a layout $f: T^* \to \mathbb{Z}$ for some $T^* \in S(T)$. By construction we have $|f(x) - f(y)| \leqq \|P\|$ for any $xy \in E(P^*) \cup (\cup E(\bar{H}_i))$. Now for each i there is a unique edge $w_i p_i \in E(T)$, $w_i \in H_i$, $p_i \in P$. We choose a $w_i \in \bar{H}_i$ (to correspond to w_i and B_i), and then find a $p_i \in P^*$ (to correspond to $p_i \in P$) that is at most distance $\|P\|$ from w_i. Letting these $w_i p_i$ be the remaining edges of T^*, we get $|f| \leqq \|P\|$ as required. □

In order to make use of the lemma in the algorithm we need some further notation.

Let T be a tree of maximum degree 3. If the vertices of T are numbered in the order of a depth-first search (DFS) starting at r, then the resulting numbers s can be used to define the usual relations on rooted trees: if x and y belong to the same branch at r and the path in T from r to y passes through x, then we write $x \leqq y$. If $x \leqq y$ then we call x an *ancestor* of y and we call y a *descendant* of x. If y is a descendant of x and $xy \in E(T)$, then y is a *son* of x and x is a *father* of y. If Q is a subtree of T, then we define *head*(Q) to be the vertex $x \in Q$ such that $x \leqq y$ for all $y \in Q \backslash \{x\}$. For $x \in V(T)$, let $B_x(T)$ denote the subtree of T consisting of x and all descendants of x in T. We write B_x instead of $B_x(T)$ when T is understood by context. For any subtree $\gamma \subseteq T$ with head$(\gamma) = s$, we let $(\gamma)_{\bar{s}}$ be the tree with vertex set $V(\gamma) \cup \{\bar{s}\}$, where $\bar{s}$ is a new symbol, and edge set $E(\gamma) \cup \{s\bar{s}\}$. Thus $(\gamma)_{\bar{s}}$ is obtained by attaching a new endpoint to s. Accordingly, if β is a backbone of γ which remains a backbone of $(\gamma)_{\bar{s}}$, then we let $\|\beta\|_{\bar{s}}$ be the norm of β in $(\gamma)_{\bar{s}}$. We say that B_x is *k-primitive* (or just primitive when k is understood) if $B^*(B_x) = k$ and $B^*(B_y) < k$ for all $y \in B_x$. Note that if B_x and B_z are distinct k-primitive branches, then $B_x \cap B_z = \varnothing$. If P is a path in T, we say that P *threads* the branch B_x if P passes through x and some end vertex of B_x.

COROLLARY 2.1. (a) *Let B_x and B_y be branches of T such that $B_x \cap B_y = \varnothing$ and $B^*(B_x) = B^*(B_y) = B^*(T)$. Then any optimal backbone of T threads both B_x and B_y.*

(b) *If T has three branches B_{x1}, B_{x2}, B_{x3} having pairwise empty intersection, then $B^*(T) \geqq 1 + \min_i B^*(B_{xi})$.*

Proof. (a) Let P be an optimal backbone of T violating the conclusion. Then at least one of B_x or B_y is contained in some hanging C of P. Thus by the lemma $B^*(T) \geqq 1 + B^*(C) \geqq 1 + B^*(B_x) > B^*(T)$, a contradiction.

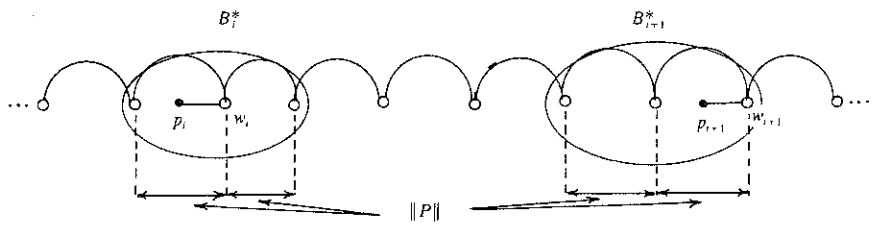


FIG. 1. *The optimal t-layout of T from Lemma 2.*

(b) By Corollary 1.1 we know that T must have an optimal backbone P. Now observe that for any optimal backbone P of T one of the B_{xi}, say B_{x1}, must be contained in a hanging from P. Hence $B^*(T) \geqq 1 + B^*(B_{x1}) \geqq 1 + \min B^*(B_{xi})$. □

COROLLARY 2.2. *For any backbone β of T we have $B^*(T) \leqq \|\beta\|$. Hence by Corollary 1.1 and Lemma 2, $B^*(T) = \min\{\|\beta\| : \beta$ a backbone of $T\}$.*

Proof. Let $Z(\beta) = \{B_j : 1 \leqq j \leqq k\}$. Using the proof of the upper bound in the lemma we can construct a t-layout g of T such that $|g| = \|\beta\|$. □

Thus a backbone β of T is *optimal* (in the sense defined previously) if and only if $\|\beta\|$ is a minimum among all backbones of T.

Before proceeding to the algorithm we introduce some notation. Let A and C be trees on which a DFS has been performed with roots x and y, respectively. Denote by $A \vee C$ the tree with $V(T) = V(A) \cup V(C) \cup \{r\}$, $E(T) = E(A) \cup E(C) \cup \{rx, ry\}$. The basic idea in our main procedure LABEL(A, C, r) is to use an inductive knowledge of $B^*(A)$, $B^*(C)$ and some additional information (described below) to find an optimal backbone β of $A \vee C$ and compute $B^*(A \vee C) = \|\beta\|$.

Now let τ be any tree of maximum degree 3 rooted at a point u. Define $h(\tau)$ to be the point of τ closest to u which can be on an optimal backbone of τ. Also let $\nu(\tau)$ be the number of $B^*(\tau)$-primitive branches of τ. Thus $\nu(\tau)$ equals 1 or 2 by Corollary 2.1(b).

COROLLARY 2.3. *Let τ be a tree of maximum degree 3 rooted at u.*

(a) *If $\nu(\tau) = 1$ then $h(\tau) = u$.*

(b) *Suppose $\nu(\tau) = 2$, and let B_x and B_y be the $B^*(\tau)$-primitive branches of τ. Then $h(\tau)$ is the point closest to u on the unique path in τ joining x and y.*

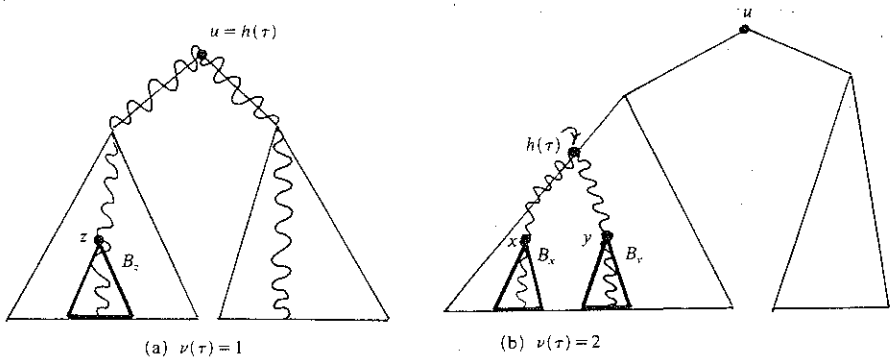
Proof. (a) Let B_z be the unique $B^*(\tau)$-primitive branch of τ. Clearly there exists a backbone β of τ threading B_z and passing through u. Let α be any hanging of β. If $\alpha \subset B_z$, then $B^*(\alpha) \leqq B^*(\tau) - 1$ by primitivity of B_z. If $\alpha \subset \tau - B_z$ then $B^*(\alpha) \leqq B^*(\tau) - 1$ since otherwise $B^*(\alpha) = B^*(\tau)$ and hence α would contain a $B^*(\tau)$-primitive branch, contradicting the uniqueness of B_z. Thus $\|\beta\| \leqq B^*(\tau)$ so β is optimal and $h(\tau) \leqq \text{head}(\beta) = u$. But $h(\tau) \geqq u$ since u is the root, so $h(\tau) = u$.

(b) By Corollary 2.1 any optimal backbone β must thread B_x and B_y. Hence β passes through x and y and the claim follows. □

We illustrate the conclusion of the lemma in Fig. 2. The curled line indicates an optimal backbone.

For any $s \in \tau$, let τ/s be the tree $\tau - B_s(\tau)$. We will be interested in $\tau/h(\tau)$. We define the *signature* of τ, sign (τ), to be the 4-tuple $(B^*(\tau), \nu(\tau), h(\tau), B^*((\tau)_u))$.

We define a nested sequence Tree (τ) of subtrees $\{\tau_i\}$ of τ, where $u \in \tau_i$ for all i, as follows. Let $\tau_1 = \tau$. If τ is a path, then Tree $(\tau) = \{\tau_1\}$. Otherwise assume inductively



(a) $\nu(\tau) = 1$ (b) $\nu(\tau) = 2$

FIG. 2. *$h(\tau)$ depending on $\nu(\tau)$.*

that $\tau_1, \tau, \cdots, \tau_i$ have been defined. If $h(\tau_i) = u$, then Tree $(\tau) = \{\tau_1, \tau_2, \cdots, \tau_i\}$. Otherwise let $\tau_{i+1} = \tau_i / h(\tau_i)$. Note that if u has only one son, then $\tau_{|\text{Tree}(\tau)|}$ is a path with endpoint u. We define deck (τ) as the ordered list deck $(\tau) = \{\text{sign}(\tau_1), \text{sign}(\tau_2), \cdots, \text{sign}(\tau_{|\text{Tree}(\tau)|})\}$, where sign (τ_1) is the top and sign $(\tau_{|\text{Tree}(\tau)|})$ is the bottom. If $L = (l_1, l_2, \cdots, l_m)$ and $N = (n_1, n_2, \cdots, n_j)$ are two ordered lists, we let $L \oplus N$ be the ordered list $(l_1, l_2, \cdots, l_m, n_1, n_2, \cdots, n_j)$. Typically L and N will be of the form deck (τ) for some τ. If $k \leqq m$, we let $L_k = (l_1, l_2, \cdots, l_k)$, so deck $(\tau)_k$ refers to the top k entries of deck (τ). We call these entries *frames.*

We now introduce the two procedures used in our algorithm. The main one LABEL (A, C, r) had deck (A) and deck (C) as input, where A and C are nonempty trees, and deck $(A \vee C)$ as output. We use it in working our way up T in postorder, where $A = B_s$, $C = B_t$ for $s, t \in T$, and $A \vee C = B_u$, where u is the father of s and t. The second, MAKEDECK (T, r) has deck (T) as input, where T is rooted r, and deck $((T)_r)$ as output. It is called by LABEL in a certain case. The validity of LABEL will be proved in its description while that of MAKEDECK will follow from some later lemmas.

Procedure LABEL (A, C, r).
Input. deck (A), deck (C) *Output.* deck $(A \vee C)$

We let Tree $(A) = \{\tau_1, \tau_2, \cdots, \tau_k\}$, and Tree $(C) = \{\mu_1, \mu_2, \cdots, \mu_l\}$. Initially we scan sign $(\tau_1) =$ sign (A) and sign $(\mu_1) =$ sign (C) in deck (A) and deck (C), respectively.

Case 1. $B^*(A) = B^*(C) \equiv b$.

Since $\nu(A) + \nu(C) \geqq 2$, we have $\nu(A) + \nu(C) \geqq 3$ or $\nu(A) = \nu(C) = 1$. We consider these subcases in turn.

Subcase 1(a). $\nu(A) + \nu(C) \geqq 3$.

By Corollary 2.1(b) it follows that $B^*(A \vee C) \geqq 1 + b$. But any backbone β of $A \vee C$ containing r satisfies $\|\beta\| \leqq 1 + b$ since any hanging α of β is a subtree of A or C so by monotonicity of B^* satisfies $B^*(\alpha) < b$. Hence by Corollary 2.2 we have $B^*(A \vee C) = 1 + b$. Clearly $\nu(A \vee C) = 1$ since $A \vee C$ is its own $(1+b)$-primitive branch, and $h(A \vee C) = r$ since (as remarked above) any backbone containing r is optimal. The same β satisfies $\|\beta\|_r \leqq 1 + b$ since the only additional hanging it has in $(A \vee C)_r$ is the single point $\bar{r}$. Thus $B^*((A \vee C)_r) \leqq 1 + b$. The opposite inequality follows from $(A \vee C)_r \supset A \vee C$ and monotonicity of B^*. The procedure thus ends with deck $(A \vee C) = \{(1 + b, 1, r, 1 + b)\}$.

Subcase 1(b). $\nu(A) = \nu(C) = 1$.

Let $B_x \subset A$ and $B_y \subset C$ be the unique b-primitive branches of A and C, respectively. Consider any backbone β of $A \vee C$ which threads both B_x and B_y. Each of its hangings α is either properly contained in B_x or B_y, or contains no b-primitive branches. Hence $B^*(\alpha) \leqq b - 1$, so $\|\beta\| \leqq b$ and thus $B^*(A \vee C) \leqq b$ by Corollary 2.2. The opposite inequality is immediate by monotonicity of B^*. Thus $B^*(A \vee C) = b$. Obviously $\nu(A \vee C) = 2$ since B_x and B_y are two b-primitive branches, and $h(A \vee C) = r$ since any optimal backbone must thread B_x and B_y and hence contains r.

As for $B^*((A \vee C)_r)$ we have two cases. If $B^*(A) = 1$, then the assumption $\nu(A) = \nu(C) = 1$ implies that each of A and C has only one endpoint in $A \vee C$, and hence that $A \vee C$ is a subdivision of a path. Thus $(A \vee C)_r$ is a subdivision of $K_{1,3}$, so $B^*((A \vee C)_r) = 2$. We then return deck $(A \vee C) = \{(1, 2, r, 2)\}$. If $B^*(A) \geqq 2$, then the same β above satisfies $\|\beta\|_r \leqq b$, so $B^*((A \vee C)_r) \leqq b$. The opposite inequality again follows from monotonicity of B^*. Hence we return deck $(A \vee C) = \{(b, 2, r, b)\}$.

Case 2. $b = B^*(A) > B^*(C)$.

Subcase 2(a). $\nu(A) = 1$.

Here we can find a backbone β of $A \vee C$ which threads the unique b-primitive branch B of A and passes through r. Such a β satisfies $\|\beta\| \leqq b$, so that $B^*(A \vee C) = b$ and $h(A \vee C) = r$. Also $\nu(A \vee C) = 1$ since B is unique. Again the same β satisfies $\|\beta\|_r \leqq b$ so $B^*((A \vee C)_r) \leqq b$ and by monotonicity $B^*((A \vee C)_r) = b$. We then return deck $(A \vee C) = \{(b, 1, r, b)\}$.

Subcase 2(b). $\nu(A) = 2$.

Let B_x and B_y be the two b-primitive branches of A. Clearly any backbone β of $A \vee C$ satisfying $\|\beta\| = b$ (if such exists) must thread both B_x and B_y, and head $(\beta) = h(A)$. Also, its hanging $(A \vee C)/h(A)$ must have B^* value $\leqq b-1$.

It follows that if $B^*((A \vee C)/h(A)) = b$ then $B^*(A \vee C) \geqq b+1$. Equality follows since for any backbone δ of $A \vee C$ containing r the hangings α all satisfy $B^*(\alpha) \leqq \max\{B^*(A), B^*(C)\} = b$ so such a δ satisfies $\|\delta\| \leqq b+1$. Also $\|\delta\|_r \leqq b+1$ since the only additional hanging is a single point, and $\nu(A \vee C) = 1$ since $A \vee C$ is its own $(b+1)$-primitive branch. Thus we output deck $(A \vee C) = \{(b+1, 1, r, b+1)\}$.

Suppose on the other hand that $B^*((A \vee C)/h(A)) \leqq b-1$. Take any backbone β of $A \vee C$ threading the two b-primitive branches B_x and B_y of A. Then every hanging α of it satisfies exactly one of the following:

(a) $\alpha \subset B_x$ or $\alpha \subset B_y$ with proper containment.

(b) α contains no b-primitive branches.

(c) $\alpha = (A \vee C)/h(A)$.

Each of these implies that $B^*(\alpha) \leqq b-1$ so $\|\beta\| \leqq b$ and hence $B^*(A \vee C) = b$. Also head $(\beta) = h(A)$ since any optimal backbone of A threads B_x and B_y and thus has a uniquely determined head which must therefore be the same as head (β). Clearly $\nu(A \vee C) = 2$. As for $B^*((A \vee C)_r)$, the argument given above for $A \vee C$ can be repeated for $(A \vee C)_r$ to show that if $B^*((A \vee C)_r/h(A)) \leqq b-1$, then $B^*((A \vee C)_r) = b$ while if $B^*((A \vee C)_r/h(A)) = b$, then $B^*((A \vee C)_r) = b+1$. We conclude that sign $(A \vee C) = (b, 2, h(A), z)$, where $z = B^*((A \vee C)_r)$ depends on $B^*((A \vee C)_r/h(A))$ as indicated. Thus the first frame of deck $(A \vee C)$ is sign $(A \vee C)$ as given, so deck $(A \vee C) = (b, 2, h(A), z) \oplus$ deck $((A \vee C)/h(A))$.

To summarize, we see that in Subcase 2(b) if $B^*((A \vee C)/h(A)) = b$, then deck $(A \vee C) = \{(b+1, 1, r, b+1)\}$ while if $B^*((A \vee C)/h(A)) \leqq b-1$, then deck $(A \vee C) = (b, 2, h(A), z) \oplus$ deck $((A \vee C)/h(A))$. Hence the output of LABEL (A, C) is given directly in the first case, while in the second the procedure must still produce deck $((A \vee C)/h(A))$. When this is done z is determined from the last entry, corresponding to $B^*((A \vee C)_r/h(A))$, of the top frame of deck $((A \vee C)/h(A))$, and the given deck $(A \vee C)$ is returned.

Consider then the construction of deck $((A \vee C)/h(A))$ which, since $(A \vee C)/h(A) = (A/h(A)) \vee C$, is deck $((A/h(A)) \vee C)$. If $A/h(A) \neq \varnothing$, then we do this by making a recursive call on LABEL $(A/h(A), C, r) =$ LABEL (τ_2, μ_1, r) after popping sign $(\tau_1) =$ sign (A) from the top of deck (A) so that sign $(\tau_2) =$ sign $(A/h(A))$ is available for scanning. We get as output the required deck $((A/h(A)) \vee C)$. If $A/h(A) = \varnothing$ we cannot make this call and instead use procedure MAKEDECK as follows.

Suppose then that $A/h(A) = \varnothing$. Then $(A/h(A)) \vee C = (C)_y$, where y is the root of C, i.e., the son of r in C. Hence we get deck $((A/h(A)) \vee C)$ by calling on procedure MAKEDECK (C, y).

This completes procedure LABEL (A, C, r).

We illustrate the Subcase 2(b) requiring recursion in Figs. 3 and 4. The pointers in Fig. 4 show the entries to be scanned in the indicated calls to LABEL.

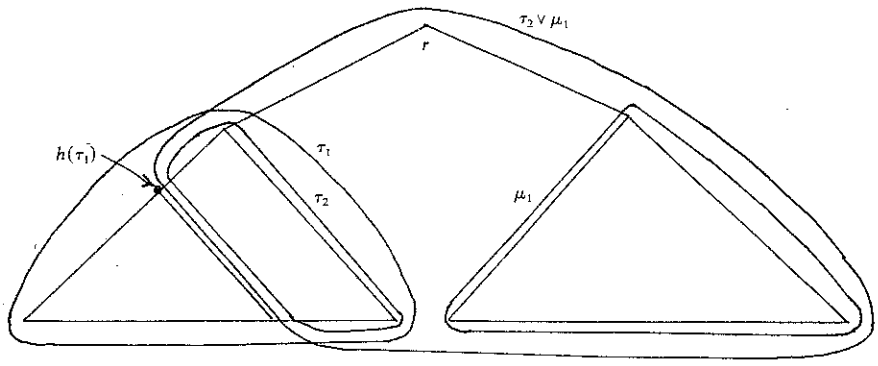


FIG. 3. *Determining* $B^*(\tau_1 \vee \gamma_1)$ *from* $B^*(\tau_2 \vee \gamma_1)$.

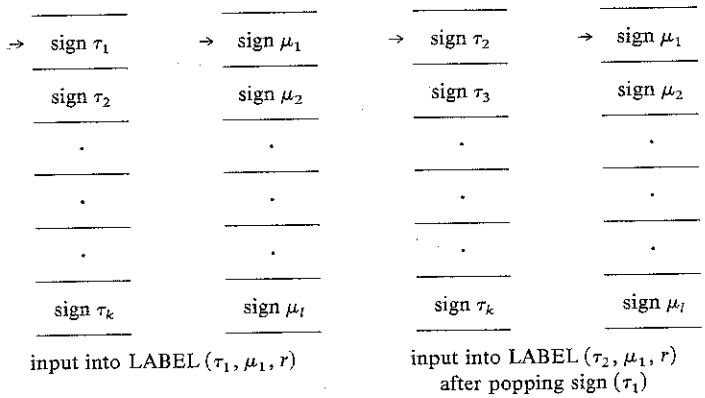


FIG. 4. *The deck structure in recursion.*

The procedure may continue calling itself through possibly $k+l$ levels in recursion (after popping a sign (τ_i) or sign (μ_j) in preparation each time) until either Case 1 or Subcase 2(a) obtains, where of course A and C are replaced by some τ_i or μ_j. At this point the answer cascades back up as in the discussion of Subcase 2(b) to give the final output of LABEL (A, C, r).

We must now develop the procedure MAKEDECK (T, r) for an arbitrary tree T of maximum degree 3 rooted at a point r of degree 1 or 2. We begin with some lemmas.

A *pendant path* in T is a path P in T with endpoints x and y at least one of which is an endpoint in T and such that $\deg(z, T) = 2$ for all $z \in P \backslash \{x, y\}$.

LEMMA 3. *Let T be a tree. Let P be a pendant path in T from x to y such that $\deg_T(x) = 1$. Suppose T_p is a copy of T except that edges in P may be subdivided. Then $B^*(T) = B^*(T_p)$.*

Proof. First note that $B^*(T_p) \geqq B^*(T)$ because any subdivision of T_p is also a subdivision of T. Thus we need only show the other direction, which we will do by induction on $|T|$. Let Q be an optimal backbone of T.

Case 1. $P \cap Q = \varnothing$. Then P is contained in H, a hanging of Q. By the induction hypothesis, $B^*(H_p) = B^*(H)$ since H is smaller than T. Also the hangings of Q other than H do not contain P, so they have the same B^* values whether viewed as subtrees of T or T_p. Thus, all the hangings of Q in T_p have the same B^* values as the hangings of Q in T. Hence by Lemma 2 there is a t-layout of T_p having bandwidth $\|Q\| = B^*(T)$.

Case 2. $P \cap Q = (y)$. Let $H = P' = P - y$, so that H is a hanging of Q and P' is a pendant path in H (which happens to be all of H). This gives the same situation as Case 1 with $B^*(H) = B^*(H_{p'}) = 1$. The same argument as in Case 1 then gives $B^*(T_p) \leqq B^*(T)$.

Case 3. $P \subset Q$. Then P is the tail of Q and it has no hangings, so by the construction of the optimal layout of T given by Lemma 2, the points on P will be numbered with integers spaced $\|Q\|$ apart at the high end (or low end—it does not matter) of the range of the layout. But then it is easy to push out the high end of the range in order to fit in the additional points of T_p. This does not change the bandwidth.

Note that there can be no other way for P and Q to intersect because P is pendant. The proof is thereby completed. □

We now pass to a description of deck $((T)_r)$ in terms of deck (T). As notation we let Tree $(T) = (\gamma_1, \gamma_2, \cdots, \gamma_m)$ and Tree $((T)_r) = (\gamma'_1, \gamma'_2, \cdots, \gamma'_p)$. We recall that $\bar{r}$ is the endpoint not in T attached to r. We view it as the father of r in a depth-first search of $(T)_r$.

LEMMA 4. *Suppose T is rooted at a point r satisfying* $\deg(r, T) = 1$. *Then* deck $((T)_r) =$ deck $(T)_{m-1} \oplus (1, 1, \bar{r}, 1)$.

Proof. We begin by showing that $\gamma'_i = (\gamma_i)_r$, $1 \leqq i \leqq m$, and that $m = p$. For the first claim we proceed by induction on i. It is trivially true for $i = 1$. Suppose it is true for all $i \leqq i_0$, where $i_0 < m$. If $h(\gamma_{i_0}) \neq r$, then γ_{i_0} has two $B^*(\gamma_{i_0})$-primitive branches. Since $\gamma'_{i_0} = (\gamma_{i_0})_r$ by induction, we have $B^*(\gamma'_{i_0}) = B^*(\gamma_{i_0})$ by Lemma 3 and hence any optimal backbone of γ'_{i_0} must thread the same two branches. Thus $h(\gamma'_{i_0}) = h(\gamma_{i_0})$ and hence $\gamma'_{i_0+1} = (\gamma_{i_0+1})_r$, so the induction is completed. The fact that $m = p$ follows immediately since $\gamma'_m = (\gamma_m)_r$ and $h(\gamma_m) = r$ imply that $h(\gamma'_m) = \bar{r}$. Thus $\gamma'_{m+1} = \varnothing$ so γ'_m is the last element of Tree $((T)_r)$.

We can now complete the proof. First we show that sign $(\gamma'_j) =$ sign (γ_j) for $j < m$. The conditions $\deg(r, T) = 1$, $\gamma'_j = (\gamma_j)_r$, and Lemma 3 imply that $B^*((\gamma'_j)_r) = B^*(\gamma'_j) = B^*((\gamma_j)_r) = B^*(\gamma_j)$, and that the $B^*(\gamma'_j)$-primitive branches of γ'_j are identical to the $B^*(\gamma_j)$-primitive branches of γ_j. The condition $\gamma'_j = (\gamma_j)_r$ for all j implies $h(\gamma'_j) = h(\gamma_j)$ for $j < m$. It follows that sign $(\gamma'_j) =$ sign (γ_j) for $j < m$.

For $j = m$, just observe that γ_m is a path with endpoint r so $\gamma'_m = (\gamma_m)_r$ is a path with endpoint $\bar{r}$. Thus sign $(\gamma'_m) = \{(1, 1, \bar{r}, 1)\}$.

The lemma is thus proved. □

We now make some observations that lead to a description of deck $((T)_r)$, where r has degree 2.

LEMMA 5. *Suppose that for some integer $j < m$ we have $B^*(\gamma_q) = B^*((\gamma_q)_r)$ for $1 \leqq q \leqq j$. Then*

(1) $\gamma'_q = (\gamma_q)_r$ *for* $1 \leqq q \leqq j+1$, *and*

(2) sign $(\gamma'_q) =$ sign (γ_q) *for* $1 \leqq q \leqq j$.

Proof. We start with (1). Proceeding by induction, suppose there is an integer $q_0 \leqq j$ for which (1) is true for all $q \leqq q_0$, the case $q_0 = 1$ being trivially true. Since $q_0 < m$ we have $h(\gamma_{q_0}) \neq r$. Thus any optimal backbone of γ_{q_0} threads two $B^*(\gamma_{q_0})$-primitive branches of γ_{q_0}, and since $B^*(\gamma_{q_0}) = B^*((\gamma_{q_0})_r)$, any optimal backbone of $(\gamma_{q_0})_r$ must thread these same two branches. Hence $h(\gamma'_{q_0}) = h((\gamma_{q_0})_r) = h(\gamma_{q_0})$, so $\gamma'_{q_0+1} = (\gamma_{q_0+1})_r$ and the inductive step is done.

Now consider (2). We may suppose that $m > 1$ since otherwise the statement is vacuously true. Take any q in the range $1 \leqq q \leqq j$. By hypothesis $B^*(\gamma_q) = B^*((\gamma_q)_r)$ and by (1) we have $B^*(\gamma'_q) = B^*((\gamma_q)_r)$, so $B^*(\gamma'_q) = B^*(\gamma_q)$. By Lemma 3 we have $B^*((\gamma'_q)_{\bar{r}}) = B^*(\gamma'_q) = B^*((\gamma_q)_r)$. Hence the first and last entries of sign (γ'_q) are identical to those of sign (γ_q). Since $q < m$ we have $h(\gamma_q) < m$, and the argument in (1) can be

repeated to give $h(\gamma'_q) = h(\gamma_q)$ and $\nu(\gamma'_q) = \nu(\gamma_q) = 2$. Hence sign $(\gamma'_q) =$ sign (γ_q). Part (2) is proved and we are done. □

LEMMA 6. *Suppose T is rooted at a point r satisfying* deg $(r, T) = 2$.

(1) *Let j be the first index, if it exists, such that $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$ (so that $B^*((\gamma_j)_r) = B^*(\gamma_j)+1$). Then*

$$\text{deck}\,((T)_r) = \text{deck}\,(T)_{j-1} \oplus (B^*(\gamma_j)+1, 1, \bar{r}, B^*(\gamma_j)+1).$$

(2) *If no such j exists, then*

(2a) *If* deg $(r, \gamma_m) = 1$, *then*

$$\text{deck}\,((T)_r) = \text{deck}\,(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m)).$$

(2b) *If* deg $(r, \gamma_m) = 2$, *then*

(2b′) *If $\nu(\gamma_m) = 1$, then*

$$\text{deck}\,((T)_r) = \text{deck}\,(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m)).$$

(2b″) *If $\nu(\gamma_m) = 2$, then*

$$\text{deck}\,((T)_r) = \text{deck}\,(T) \oplus (1, 1, \bar{r}, 1).$$

Proof. (1) Suppose that the stated j exists. Then by Lemma 5 $\gamma'_q = (\gamma_q)_r$ for $1 \leqq q \leqq j$ and sign $(\gamma'_q) =$ sign (γ_q) for $1 \leqq q \leqq j-1$. As for sign (γ'_j), note that any backbone β of $\gamma'_j = (\gamma_j)_r$ containing $\bar{r}$ satisfies $\|\beta\| \leqq B^*(\gamma_j)+1$ since all its hangings α satisfy $\alpha \subset \gamma_j$ and hence $B^*(\alpha) \leqq B(\gamma_j)$. Thus $h(\gamma'_j) = \bar{r}$, so sign (γ'_j) is the last frame of deck $((T)_r)$. We have by Lemma 3 and hypothesis $B^*((\gamma'_j)_{\bar{r}}) = B^*(\gamma'_j) = B^*((\gamma_j)r) = B^*(\gamma_j)+1$. Finally $\nu(\gamma'_j) = 1$ since γ'_j is its own $B^*(\gamma_j)$-primitive branch. Thus sign $(\gamma'_j) = (B^*(\gamma_j)+1, 1, \bar{r}, B^*(\gamma_j)+1)$ and hence deck $((T)_r) =$ deck $(T)_{j-1} \oplus (B^*(\gamma_j)+1, 1, \bar{r}, B^*(\gamma_j)+1)$ as required.

(2) Suppose no such j exists. Since γ_m is the last tree in Tree (T), it follows that $h(\gamma_m) = r$. Hence there is an optimal backbone β of γ_m containing r. We also have sign $(\gamma'_q) =$ sign (γ_q) for $q < m$ by Lemma 5.

Case 2(a). We can extend β to $\bar{r}$ thereby obtaining a backbone $\bar{\beta}$ of $(\gamma_m)_r$. Clearly $\|\bar{\beta}\| = \|\beta\|$ since $\bar{\beta}$ has the same hangings as β. Thus $B^*((\gamma_m)_r) = B^*(\gamma_m)$. By Lemma 5 we have $\gamma'_m = (\gamma_m)_r$ so $B^*(\gamma'_m) = B^*(\gamma_m)$, and by Lemma 3, $B^*((\gamma'_m)_{\bar{r}}) = B^*(\gamma'_m) = B^*(\gamma_m)$. Finally $h(\gamma'_m) =$ head $(\bar{\beta}) = \bar{r}$, and $\nu(\gamma'_m) = 1$ since otherwise $h(\gamma'_m) \neq \bar{r}$. Thus sign $(\gamma'_m) = (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m))$. Clearly γ'_m is the last element of Tree $((T)_r)$. Thus deck $((T)_r) =$ deck $(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m))$.

Case 2(b). Assume first that $\nu(\gamma_m) = 1$. Then any backbone $\bar{\beta}$ of $\gamma'_m = (\gamma_m)_r$ which threads the unique $B^*(\gamma_m)$-primitive branch of γ_m satisfies $\|\bar{\beta}\| = B^*(\gamma_m)$. We can choose such a $\bar{\beta}$ which contains $\bar{r}$. Hence we get $B^*(\gamma'_m) = B^*(\gamma_m)$ and $h(\gamma'_m) = \bar{r}$. As above $B^*((\gamma'_m)_{\bar{r}}) = B^*(\gamma'_m)$, and $\nu(\gamma'_m) = 1$ since γ'_m has the same unique $B^*(\gamma_m)$-primitive branch as γ_m. Thus deck $((T)_r) =$ deck $(T)_{m-1} \oplus (B^*(\gamma_m), 1, r, B^*(\gamma_m))$.

Now suppose $\nu(\gamma_m) = 2$. Since $B^*(\gamma'_m) = B^*(\gamma_m)$ any optimal backbone $\bar{\beta}$ of γ'_m must thread the same two $B^*(\gamma_m)$-primitive branches of γ_m as does β. Hence $h(\gamma'_m) =$ head $(\bar{\beta}) =$ head $(\beta) = r$. Clearly $\nu(\gamma'_m) = \nu(\gamma_m) = 2$, and $B^*((\gamma'_m)_r) = B^*(\gamma'_m)$ as before. Thus sign $(\gamma'_m) =$ sign (γ_m). Also we have $\gamma'_{m+1} = \{\bar{r}\}$, so sign $(\gamma'_{m+1}) = (1, 1, \bar{r}, 1)$. Thus deck $((T)_r) =$ deck $(T) \oplus (1, 1, \bar{r}, 1)$. The lemma is therefore proved. □

The procedure MAKEDECK (T, r) is now essentially given by Lemmas 4 and 6. For completeness we describe it explicitly.

Procedure MAKEDECK (T, r).

Input. deck (T), where T is a tree of maximum degree 3 rooted at r.

Output. deck $((T)_r)$

(1) determine deg (r, T)

(2) If deg $(r, T) = 1$, then

$$\text{deck}((T)_r) \leftarrow \text{deck}(T)_{m-1} \oplus (1, 1, \bar{r}, 1).$$

(3) If deg $(r, T) = 2$, then

(a) Search deck (T) from top to bottom until you find the minimum j, if it exists, such that $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$.

(b) If such a j exists, then

$$\text{deck}((T)_r) \leftarrow \text{deck}(T)_{j-1} \oplus (B^*(\gamma_j)+1, 1, \bar{r}, B^*(\gamma_j)+1).$$

(c) If no such j exists, then determine deg (r, γ_m). Depending on the answer, output deck $((T)_r)$ according to the conditions defining cases (2a), (2b′), and (2b″) of Lemma 6.

This completes MAKEDECK (T, r).

We remark that to determine deg (r, T) and deg (r, γ_m) we must maintain a new fifth entry, having value "1" or "2" to denote deg (r, γ_j) in sign (γ_j) for all $\gamma_j \in \text{Tree}(T)$. We describe the updating of deg (r, γ_j) here, omitting its formal inclusion in the lemmas and procedures to simplify their presentations.

We suppose then that in procedure LABEL (A, C, r) each frame sign (τ_i) and sign (μ_j) in deck (A) and deck (C), respectively, has the additional bit deg (x, τ_i) or deg (y, μ_j) (respectively), where x and y are the roots of A and C (and thus the sons of r in T). Our goal is to describe deg (r, γ_j) in all frames sign (γ_j) of deck $(A \vee C)$.

Suppose first that any of Subcases 1(a), 1(b), or 2(a) in LABEL (A, C, r) holds. Then the output deck $(A \vee C)$ has a single frame. Since A and C are nonempty we have deg $(r, A \vee C) = 2$, so we insert "2" for deg $(r, A \vee C)$ in the single frame.

Suppose then that Subcase 2(b) holds. The possible outputs are

(a) deck $(A \vee C) = \{(b+1, 1, r, b+1)\}$.

(b) deck $(A \vee C) = (b, 2, h(A), B^*((A \vee C)_r)) \oplus \text{deck}((A \vee C)/h(A))$.

In case (a) we again insert "2" for deg $(r, A \vee C)$ as above. Suppose (b) holds. For the topmost frame we insert "2" again since A and C are nonempty so deg $(r, A \vee C) = 2$. The remaining frames (those in deck $((A \vee C)/h(A))$) are handled as follows. If $A/h(A) \neq \varnothing$, then a call to LABEL $(A/h(A) \vee C)$ was made and hence the degree entries in the frames of deck $(A/h(A) \vee C) = \text{deck}((A \vee C)/h(A))$ have been inserted by recursion. If $A/h(A) = \varnothing$, then a call is made to MAKEDECK (C, y). We then insert "1" in all frames of the resulting deck $((C)_y)$. The reason is that r (which plays the role of $\bar{y}$ in MAKEDECK (C, y)) is an endpoint of $(C)_y$ attached to y and hence has degree 1 in each element of Tree $((C)_y)$.

3. Complexity. We now discuss the complexity of procedures LABEL (A, C, r) and MAKEDECK (T, r) and the complexity of the resulting algorithm for $B^*(T)$ in trees T of maximum degree 3.

Let T be any tree of maximum degree 3 rooted at a point r. We assume a DFS on T with associated branches B_x for all $x \in T$. Let $|\text{deck}(T)|$ denote the number of frames (or signatures) in deck (T), so $|\text{deck}(T)| = |\text{Tree}(T)|$. Our first goal is to estimate $|\text{deck}(T)|$.

For the next lemma we recall that a tree T is k-primitive if and only if it is its own (and hence unique) k-primitive branch, i.e., $B^*(T) = k$ and $B^*(T') < k$ for any proper subtree T' with $r \notin T'$.

LEMMA 7. *Let T be a k-primitive tree. Then* $|T| \geqq 2^{k-1}$.

Proof. Let $f_k = \min\{p: |T| = p, T \text{ is } k\text{-primitive}\}$. We show $f_k \geqq 2^{k-1}$ by induction on k.

For $k = 1$, the one-point tree is k-primitive by convention and has minimum order. Thus $f_1 \geqq 1 = 2^0$. Now suppose $f_j \geqq 2^{j-1}$ for $j < s$.

Let T be s-primitive and have minimum possible order. We claim T must have at least two $(s-1)$-primitive branches. Suppose first that T has no $(s-1)$-primitive branches. Then any backbone β of T through r satisfies $\|\beta\| \leqq s-1$, so $B^*(A) \leqq s-1$, a contradiction. Similarly if T has only one $(s-1)$-primitive branch B_z, then any backbone β of T threading B_z and passing through r satisfies $\|\beta\| \leqq s-1$, again a contradiction. Hence T has at least two $(s-1)$-primitive branches, so by induction $|T| = f_s \geqq 2f_{s-1} \geqq 2^{s-1}$. □

A sequence of trees $T_1, T_2, \cdots, T_k$ is called *monotone* if $B^*(T_i) > B^*(T_{i+1})$, $i \geqq 1$.

LEMMA 8. *For any tree T,* Tree (T) is *monotone.*

Proof. Let Tree $(T) = \{\tau_1, \tau_2, \cdots, \tau_l\}$. Now by definition $\tau_i / h(\tau_i) = \tau_{i+1}$, $1 \leqq i \leqq l-1$, and $B^*(\tau_i) = \|\beta_i\|$ where β_i is a t-backbone of τ_i with head $(\beta_i) = h(\tau_i)$. It follows that τ_{i+1} is a hanging of β_i in τ_i. Hence $B^*(\tau_{i+1}) \leqq \|\beta_i\| - 1 = B^*(\tau) - 1$. □

COROLLARY 8.1. *Let T be a tree on n points. Then* $|\text{deck}(T)| \leqq B^*(T) \leqq O(\log n)$.

Proof. Since $|\text{deck}(T)| = |\text{Tree}(T)|$ it suffices to prove the inequalities with $|\text{deck}(T)|$ replaced by $|\text{Tree}(T)|$.

Let Tree $(T) = \{\tau_1, \tau_2, \cdots, \tau_l\}$. Clearly $B^*(\tau_1) = B^*(T)$, so τ_1 contains a $B^*(T)$-primitive branch, and hence $n = |\tau_1| \geqq 2^{B^*(T)-1}$. It follows that $B^*(T) \leqq O(\log n)$. Also $l < B^*(T)$ since Tree (T) is monotone. □

LEMMA 9. Let A and C be trees with $|A| + |C| = n$. Then LABEL(A, C, r) has complexity $O(\log n)$.

Proof. We may divide the running time of the algorithm into two parts. In the first part LABEL makes nested calls on itself of the form LABEL (τ_i, μ_j). In the second part, which may not occur, LABEL calls on MAKEDECK (μ_{j_0}, y) or MAKEDECK (τ_{i_0}, x) for some $j_0 \leqq |\text{deck}(C)|$ or $i_0 \leqq |\text{deck}(A)|$. Without loss of generality let this call be MAKEDECK (μ_{j_0}, y).

Consider the first part. Each call requires $O(1)$ time since it consists in scanning sign (τ_i) and sign (μ_j) and then defining deck $(\tau_i \vee \mu_j)$ as a single frame or the result of pushing a single frame onto the recursively computed deck $(\tau_{i+1} \vee \mu_j)$ or deck $(\tau_i \vee \mu_{j+1})$. The call pops either sign (τ_i) or sign (μ_j) before the next call begins. The total number of calls is the number of times a frame is popped before the call to MAKEDECK (μ_{j_0}, y). Hence this number is $|\text{deck}(A)| + j_0$, so the time spent on the first part is $O(|\text{deck}(A)| + j_0)$.

In the second part MAKEDECK processes from top to bottom all the remaining $|\text{deck}(C)| - j_0$ frames of deck (C). It spends at most $O(1)$ time on each frame since it only scans for the condition $B^*((\gamma)_r) \neq B(^*\gamma_j)$ or inserts a new frame at the bottom of the ones it has already checked. The time for the second part is thus $O(|\text{deck}(C)| - j_0)$.

It follows that the total time of LABEL (A, C, r) is $O(|\text{deck}(A)| + |\text{deck}(C)|) \leqq O(\log|A| + \log|C|) \leqq O(\log(|A||C|) \leqq O(\log(n^2)) = O(\log n)$. □

LEMMA 10. *Let A be a tree with $|A| = n$ rooted at r. Then* MAKEDECK (A, r) *has complexity* $O(\log n)$.

Proof. The proof is essentially the same as the second part of the proof of Lemma 9. □

We can now give an $O(n \log n)$ algorithm for $B^*(T)$. Later we sketch how it can be refined to linear time.

THEOREM 1. *Let T be a tree of maximum degree* 3. *Then $B^*(T)$ can be computed*

in $O(n \log n)$ time.

Proof. Suppose first that t has a point r of degree 2.

Now view r as the root of T. Now order $V(T)$ in reverse depth-first search from endpoints "up" to r (i.e., in postorder). If e is an endpoint, then deck $(\{e\}) = \{(1, 1, e, 1)\}$, (where we recall the convention $B^*(\{e\}) = 1$). Now for some $x \in T$, assume inductively that deck (B_y) is known for each of (the at most two) sons of x. If x has only one son y, then deck (B_x) can be obtained from deck (B_y) by calling on MAKEDECK (B_y, y). Suppose then that x has sons y_1 and y_2. We then apply LABEL (B_{y_1}, B_{y_2}, x) and get deck (B_x) as output.

Continuing to work our way up we eventually apply LABEL (B_{s_1}, B_{s_2}, r), where s_i are the sons of r. The output, deck $(B_r) =$ deck (T), has in its first entry the required $B^*(T)$. The algorithm then halts.

If T has no point of degree 2, then let r be a point of degree 1. We proceed as above, calling on either LABEL or MAKEDECK as we work our way up T (rooted at r). Eventually we call on MAKEDECK (B_s, s), where s is the unique son of r. The output, deck $((B_s)_s) =$ deck (T), gives $B^*(T)$ and the algorithm halts.

In either case there are a total of n calls of the form LABEL (B_{y_1}, B_{y_2}, x) or MAKEDECK (B_y, y), each requiring at most $O(\log n)$ time by Lemmas 9 and 10. Hence the total time is bounded by $O(n \log n)$. □

4. A $O(n)$ time bound. In this section we describe informally a way of lowering the complexity of our algorithm to $O(n)$.

Let *Tree* $(A) = \{\tau_i : 1 \leqq i \leqq l\}$ and Tree $(C) = \{\gamma_j : 1 \leqq j \leqq k\}$. The main idea is to modify procedure LABEL (A, C, r). In the worst case it processes all $|\text{deck}(A)| + |\text{deck}(C)| \leqq \log|A| + \log|C|$ signatures in deck (A) and deck (C), giving a time bound $O(\log n)$, $n = |A \vee C|$. We indicate how the addition of new decks that point to gaps in the sequences $\{B^*(\tau_i)\}$ and $\{B^*(\gamma_j)\}$ permits an $O(\min\{\log|A|, \log|C|\})$ implementation of the procedure.

As motivation, suppose that LABEL (A, C, r) performs in recursion LABEL (τ_c, γ_d, r), say with $B^*(\tau_c) > B^*(\gamma_d)$. Suppose also that for some $e > c$ we have $B^*(\tau_e) = B^*(\gamma_d)$ and $B^*(\tau_{i+1}) = B^*(\tau_i) - 1$ for $c \leqq i \leqq e-1$. Then we have the equivalences $B^*(\tau_c \vee \gamma_d) \leqq B^*(\tau_c) \Leftrightarrow B^*(\tau_{c+1} \vee \gamma_d) \leqq B^*(\tau_c) - 1 \Leftrightarrow \cdots \Leftrightarrow B^*(\tau_e \vee \gamma_d) \leqq B^*(\tau_c) - (e-c)$. Of course the falsity of the above inequalities yields $B^*(\tau_c \vee \gamma_d) = B^*(\tau_c) + 1$ in which case deck $(\tau_c \vee \gamma_d)$ has only one frame. Thus the result of LABEL (τ_e, γ_d, r) telescopes up to decide the result of LABEL (τ_c, γ_d, r). As now constituted, LABEL (A, C, r) performs the $O(e-c)$ operations for achieving the telescoping. Our goal is to do this in $O(1)$ time by using a pair of successive pointers from a separate deck, one to $B^*(\tau_c)$ and the other to $B^*(\tau_e)$. When the inequalities are false, we advance the pointer to $B^*(\tau_c)$ one unit and delete the one to $B^*(\tau_e)$, and when they are true we leave both pointers stationary. In this way we can treat entire segments of deck (A) (or deck (C)), i.e., a sequence of successive frames having successive B^* values, in $O(1)$ time. The result will be that the time spent by LABEL in processing deck (A) and deck (C) becomes proportional to the number of gaps in the sequence of B^* values occurring in whichever of deck (A) or deck (C) has fewer frames.

To be precise, we define two additional decks, GAP (A) and GAP (C), as follows. Suppose $R_{jm} = \{\tau_j, \tau_{j+1}, \cdots, \tau_m\}$ is a subsequence of Tree (A) satisfying $B^*(\tau_{i+1}) = B^*(\tau_i) - 1$, $j \leqq i \leqq m-1$, $B^*(\tau_{j-1}) > 1 + B(\tau_j)$, and $B^*(\tau_{m+1}) < B^*(\tau_m) - 1$. We call R_{jm} a *run* of Tree (A) (and also of deck (A) under the correspondence between deck (A) and Tree (A)). For any $\tau \in$ Tree (A), let *run*(τ) denote the run of Tree (A) to which τ belongs. For each run R_{jm} there will be two consecutive elements $b(R_{jm})$, $t(R_{jm})$ in

the list GAP (A). We let $t(jm)$ point to $B^*(\tau_j)$ in sign (τ_j) and $b(jm)$ point to $B^*(\tau_m)$ in sign (τ_m). If R_{jm} and R_{qr} are runs with $m < q$, then the pointers to R_{jm} come before (i.e., above) those to R_{qr}. The deck GAP (C) is constructed similarly.

Our new procedure $\overline{\text{LABEL}}\,(A, C, r)$ can be described informally as follows. It has input deck (A), deck (C), GAP (A), GAP (C), and output deck $(A \vee C)$ and GAP $(A \vee C)$. Without loss of generality we take $B^*(A) \geqq B^*(C)$. For brevity we consider only the first case where $\log|C| \leqq \log|A|$; the case $\log|C| > \log|A|$ is done with straightforward but tedious changes. We let *bottomdeck*(T) be the bottommost entry of deck (T) for a tree T.

(I) Work your way up deck (A) from bottomdeck (A) to sign (τ_u), where $u = \min\{j: B^*(\tau_j) \leqq B^*(\gamma_1)\}$ if $B^*(\tau_l) \leqq B^*(\gamma_1)$ and $u = l$ otherwise. Also work up GAP (A) until you reach $b(\text{run}\,(\tau_u))$.

(II) Perform LABEL (τ_u, γ_1, r).

(III) Construct deck $((A \vee C)_r)$ and GAP $((A \vee C)_r)$ as follows.

(A) Suppose $B^*(\tau_u \vee \gamma_1) = 1 + B^*(\gamma_1)$, so that deck $(\tau_u \vee \gamma_1)$ is the single frame sign $(\tau_u \vee \gamma_1) = (1 + B^*(\gamma_1), 1, r, B^*(\gamma_1))$.

(1) Check if there is a run R_{ab} in Tree (A) one of whose members τ satisfies $B^*(\tau) = 1 + B^*(\gamma_1)$. (*Note.* Such an R_{ab} must be run (τ_u) or the run preceding run (τ_u) in Tree (A).)

(2) If there is, then move up to $t(R_{ab})$ in GAP (A) (up $\leqq 2$ pointers) and follow $t(R_{ab})$ to the top frame sign (τ_a) of R_{ab}. Now do sign $(\tau_a \vee \gamma_1) \leftarrow (1 + B^*(\gamma_1), 1, r, 1 + B^*(\gamma_1))$. Now deck $(A \vee C)$ is given by deck $(A \vee C) = \text{deck}\,(A)_{a-1} \oplus \text{sign}\,(\tau_a \vee \gamma_1)$. Note that the list deck $(A)_{a-1}$ may now be regarded as $\{\text{sign}\,(\tau_j \vee \gamma_1)\}$.

(3) If there is no R_{ab}, then let deck $(A \vee C) = \text{deck}\,(A)_{u-1} \oplus$ sign $(\tau_u \vee \gamma_1)$. Again, the entries $\{\text{sign}\,(\tau_j)\}$, $1 \leqq j \leqq u-1$, may be regarded as $\{\text{sign}\,(\tau_j \vee \gamma_1)\}$.

(4) GAP $(A \vee C)$ is obtained by making corresponding changes in GAP (A). Suppose for example that (2) holds, and let run $(\tau_a \vee \gamma_1) = R_{\alpha\beta}$ in deck $(A \vee C)$. If $R_{\alpha\beta}$ is not a singleton, then $R_{\alpha\beta}$ corresponds to a run $R_{\alpha(\beta-1)}$ in deck (A) with $\tau_a \vee \gamma_1$ appended at the end. Hence we let $t(R_{\alpha\beta}) = t(R_{\alpha(\beta-1)})$ while $b(R_{\alpha\beta})$ points to sign $(\tau_a \vee \gamma_1)$. Naturally $b(R_{\alpha(\beta-1)})$ in (GAP (A)) is deleted. All elements of GAP (A) corresponding to runs about $R_{\alpha\beta}$ are retained in GAP $(A \vee C)$. We omit here the changes appropriate to (3) since similar changes are described below.

(B) Suppose $B^*(\tau_u \vee \gamma_1) = B^*(\gamma_1)$. Then deck $(A \vee C)$ is given by deck $(A \vee C) = \text{deck}\,(A)_{u-1} \oplus \text{deck}\,(\tau_u \vee \gamma_1)$. Now GAP $((A \vee C)_r)$ is constructed as follows.

(1) Create GAP $(\tau_u \vee \gamma_1)$ by processing deck $(\tau_u \vee \gamma_1)$ top to bottom and forming the pointers appropriate to each run.

(2) Attach GAP $(\tau_u \vee \gamma_1)$ to GAP (A) as follows.

Let GAP′ (A) be the subdeck of GAP (A) of pointers corresponding to run (τ_{u-1}) and all runs above run (τ_{u-1}).

(a) If $B^*(\tau_{u-1}) > 1 + B^*(\gamma_1)$, then just append GAP $(\tau_u \vee \gamma_1)$ to the bottom of GAP′ (A) and retain the pointer structure in each.

(b) If $B^*(\tau_{u-1}) = 1 + B^*(\gamma_1)$, then append as above but remove $b(\text{run}\,(\tau_{u-1}))$ from GAP′ (A) and $t(\text{run}\,(\tau_u \vee \gamma_1))$ from GAP $(\tau_u \vee \gamma_1)$. (This has the effect of merging the runs run (τ_{u-1}) and run $(\tau_u \vee \gamma_1)$ into one in GAP$(\tau_u \vee \gamma_1)$.)

We now examine the complexity of our procedure.

THEOREM 2. $\overline{\text{LABEL}}(A, C, r)$ *has complexity* $O(\min\{\log|C|, \log|A|\})$.

Proof. For brevity we again restrict ourselves to the case $B^*(A) \geqq B^*(C)$ and $\log|C| \leqq \log|A|$ treated above.

First, (I) and (II) each require $O(\log(C)|)$ time since each processes at most $O(B^*(\gamma_1)) = O(\log|C|)$ frames. The construction of deck $(A \vee C)$ takes $O(1)$ time since it consists in just appending deck $(\tau_u \vee \gamma_1)$ below a certain position in deck (A). This position is accessed in $O(1)$ time since after performing (I) and (II) we are just below the pointer in GAP (A) that leads to this position.

GAP $(A \vee C)$ is constructed in $O(1)$ time in case (A), and in $O(\log|C|)$ time in case (B). In (A) we simply adjust one or two pointers at the position of GAP (A) reached after (I) and (II). In (B) we first construct GAP $(\tau_u \vee \gamma_1)$. This takes $O(\log|C|)$ time since we are just scanning frames sign $(\tau_i \vee \gamma_j)$ of deck $(\tau_u \vee \gamma_1)$ all with B^* values $\leqq B^*(\gamma_1)$. The number of such frames is at most $B^*(\gamma_1)$ by monotonicity of Tree $(\tau_u \vee \gamma_1)$ from Lemma 8, and $B^*(\gamma_1) \leqq O(\log|\gamma_1|) = O(\log|C|)$ by Corollary 8.1. We then splice together GAP $(\tau_u \vee \gamma_1)$ and GAP$'$ (A) by at worst deleting two pointers (the position below which we have already accessed), and this is $O(1)$ time. Hence the total time of LABEL (A, C, r) is $O(\log|C|)$, as required. □

In a similar way we can define a new procedure $\overline{\text{MAKEDECK}}(T, r)$ which does the same job as the old procedure MAKEDECK (T, r); that is, it constructs deck $((T)_r)$ from deck (T). By creating additional pointer structure, we can make the new procedure run in time $O(1)$ instead of the worst-case $O(\log n)$ for the old procedure. A brief sketch of how this is done follows.

As motivation, recall that MAKEDECK (T, r) constructs deck $((T)_r)$ from deck (T) by doing one of the following operations:

(op1) inserting a frame at the bottom of deck (T),

(op2) replacing the bottom frame of deck (T) by a different frame, or

(op3) finding the topmost frame of deck (T) in which $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$ (if such a frame exists), and replacing it and the frames under it (i.e., those associated with γ_t, where $t > j$) by the single frame $(B^*(\gamma_j)+1, 1, \bar{r}, B^*(\gamma_j)+1)$.

Notice that to do any one of these operations we need only access an "extreme" frame of some type: either the bottom frame in deck (T), or the topmost relative to the property $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$.

This suggests the construction of a new deck; call it FRAME (T). The elements of FRAME (T) will be pointers to those frames of deck (T) in which $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$, and, if not already present among the elements just mentioned, an additional pointer to the bottom frame of deck (T). These elements will be ordered top to bottom in FRAME (T) in the same relative order as the frames of deck (T) to which they point. We can then access the required frame of deck (T) in each case by using either the topmost or bottommost element of FRAME (T). Finding either of these elements takes $O(1)$ time since they are at the top or bottom of FRAME (T).

Now the procedure $\overline{\text{MAKEDECK}}(T, r)$ operates by invoking MAKEDECK (T, r) and using FRAME (T) to access the frame in deck (T) under which the insertion specified by MAKEDECK will take place. The resulting construction of deck $((T)_r)$ takes $O(1)$ time since it involves placing a single frame under or at the accessed frame of deck (T).

Finally we sketch how the time required to maintain FRAME (T) is dominated by the complexity of procedures already in use, so that the total complexity of our algorithm is unaffected (up to a constant term) by the additional pointer structure. To update FRAME (T) we need to do the following:

(1) Construct FRAME $((T)_r)$ from FRAME (T) (after $\overline{\text{MAKEDECK}}(T, r)$ has constructed deck $((T)_r)$ from deck (T)).

(2) Construct FRAME $(A \vee C)$ from FRAME (A) and FRAME (C) (after $\overline{\text{LABEL}}(A, C, r)$ has constructed deck $(A \vee C)$ from deck (A) and deck (C)).

Upon starting task (1), note that we would have already performed one of the operations (op1), (op2), or (op3) during $\overline{\text{MAKEDECK}}(T, r)$. If (op2) was performed then FRAME $((T)_r)$ is obtained from FRAME (T) by reorienting the bottom element of FRAME (T) to point to the bottom of deck $((T)_r)$. The remaining elements of FRAME (T) remain the same, except now regarded as members of FRAME $((T)_r)$. If (op3) was performed, then FRAME $((T)_r)$ is just a single element pointing to the bottom frame of deck $((T)_r)$. If (op1) was performed, then the updating depends on whether

(i) the bottom element of deck (T) has the property $B^*((\gamma_m)_r) \neq B^*(\gamma_m)$, or

(ii) the bottom element of deck (T) does not have this property.

If (i) holds then FRAME $((T)_r)$ is gotten by inserting a new element at the bottom of FRAME (T), and pointing this element to the bottom of deck $((T)_r)$. If (ii) holds, then FRAME $((T)_r)$ is obtained by reorienting the bottom element of FRAME (T) to point to the bottom of deck $((T)_r)$.

Notice that any of these updates requires at most $O(1)$ time since it involves a $O(1)$ manipulation (insertion and orientation of a pointer to the bottom of some deck) at or directly below an element of FRAME (T) already accessed in $O(1)$ time. It is also easily checked that the deck FRAME $((T)_r)$ so produced has the necessary properties, namely, elements pointing to the frames of deck $((T)_r)$ in which $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$ and ordered in the natural way, and a bottom element (if not already present) pointing to the bottom of deck $((T)_r)$.

We can achieve (2) in a manner analogous to the construction of GAP $(A \vee C)$ from GAP (A) and GAP (C) outlined previously. This requires time $O(\min\{\log|A|, \log|C|\})$ which is already required by the procedure $\overline{\text{LABEL}}$. Details of the construction and time bound in (2) are omitted here since they are nearly identical with the above-mentioned construction and time analysis for GAP $(A \vee C)$.

Our procedures $\overline{\text{LABEL}}$ and $\overline{\text{MAKEDECK}}$ can now be used to give our linear algorithm.

THEOREM 3. *Let T be a tree of maximum degree 3 on n points. Then $B^*(T)$ can be computed in $O(n)$ time.*

Proof. We carry out our procedures in the order specified by the proof of Theorem 1. That is, we work our way up T from bottom to top in postorder. Thus we start by defining deck $((e))$ for each endpoint e as before. Now as we meet each new point $x \in T$ in postorder, our job is to compute deck (B_x) given deck (B_y) for each of the (at most) two children y of x. If x has two children y_1 and y_2, then we apply $\overline{\text{LABEL}}(B_{y_1}, B_{y_2}, x)$ to get deck (B_x) at a cost of $\log(\min(|B_{y_1}|, |B_{y_2}|))$ time by Theorem 2. If x has one child y, then we apply $\overline{\text{MAKEDECK}}(B_y, y)$ to get deck (B_x) at a cost of $O(1)$ time by the above discussion. At the end we finally obtain deck $(B_r) =$ deck (T), where r is the root of T and from that we get $B^*(T)$ by scanning the topmost frame of this deck.

Now let $T(n)$ be the total time required by the algorithm described in the above paragraph. It follows from the description that $T(n+1) \leqq \max_{1 \leqq k \leqq n/2}\{T(k) + T(n-k) + \log(k)\}$.

We now claim that $T(n)$ satisfies

$$T(n) \leqq An - B\log(n) + C$$

for some constants A, B, and C. Of course, the theorem follows from this claim. Proceeding by induction, suppose the inequality holds for all $n \leqq p$. Using the inductive assumption we then have

$$T(p+1) \leqq \max_{1 \leqq k \leqq p/2} \{T(k)+T(n-k)+\log(k)\}$$

$$(*) \qquad \leqq \max_{1 \leqq k \leqq p/2} \{Ak - B\log(k) + A(p-k) - B\log(p-k) + \log(k) + 2C\}$$

$$= \max_{1 \leqq k \leqq p/2} \{Ap - \log(k^{B-1}(p-k)^B) + 2C\}.$$

We now show that there are constants B and C such that the inequality

$$(**) \qquad \max_{1 \leqq k \leqq p/2} \{-\log(k^{B-1}(p-k)^B) + 2C\} \leqq -B\log(p+1) + C$$

holds for p sufficiently large. Clearly (**) is equivalent to $\min_{1 \leqq k \leqq p/2}\{k^{B-1}(p-k)^B\} \geqq (p+1)^B 2^C$. Choosing $B=2$, we find that for $p \geqq 4$ the minimum in the left-hand side is achieved at $k=1$ and is therefore $(p-1)^2$. Now with $C=-2$ we get (**) to hold for $p \geqq 4$. Thus (**) holds with $B=2$ and $C=-2$ for all $p \geqq 4$.

Now combining (*) and (**) and observing that $Ap < A(p+1)$ we see that the inductive step is completed. It remains only to choose A sufficiently large relative to $B=2$ and $C=-2$ ($A \geqq 20$ will do) so that the claimed bound for $T(n)$ holds for $n \leqq 3$, thereby providing the base for the induction. The theorem is thus proved. □

Acknowledgments. The author thanks I. H. Sudborough and J. Ellis for discussions which stimulated the idea for the additional data structure used in the last section, and Robert Tarjan for suggesting the method of proving that the linear time bound follows from the recurrence satisfied by the time complexity. Thanks also go to Ruth Engel for executing the drawings.

REFERENCES

[1] I. Arany, L. Szoda, and W. Smith, *An improved method for reducing the bandwidth of sparse symmetric matrices*, Proc. 1971 IFIP Congress, pp. 1246–1250.

[2] M. Behzad, G. Chartrand, and L. Lesniak-Foster, *Graphs and Digraphs*, Prindle, Weber, Schmidt, Boston, MA, 1979.

[3] J. Chvatalova, *On the bandwidth problem for graphs*, Ph.D. thesis, Dept. of Combinatorics and Optimization, Univ. of Waterloo, Waterloo, Ontario, Canada, 1981.

[4] P. Chinn, J. Chvatalova, A. K. Dewdney, and N. E. Gibbs, *The bandwidth problem for graphs and matrices*, J. Graph Theory, 6 (1982), pp. 223–254.

[5] F. R. K. Chung, *On the cutwidth and topological bandwidth of a tree*, SIAM J. Algebraic Discrete Methods, 6 (1985), pp. 268–277.

[6] M. R. Garey, R. L. Graham, D. S. Johnson, and D. Knuth, *Complexity results for bandwidth minimization*, SIAM J. Appl. Math., 34 (1978), pp. 477–495.

[7] E. Gurari and I. H. Sudborough, *Improved dynamic programming algorithms for the bandwidth minimization problem and the min cut linear arrangement problem*, J. Algorithms, 5 (1984), pp. 531–546.

[8] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[9] W. Liu and A. B. Sherman, *Comparative analysis of the Cuthill–McKee ordering algorithms for sparse matrices*, SIAM J. Numer. Anal., 13 (1976), pp. 198–213.

[10] F. S. Makedon, C. H. Papadimitriou, and I. H. Sudborough, *Topological bandwidth*, SIAM J. Algebraic Discrete Methods, 6 (1985), pp. 418–444.

[11] B. Monien and I. H. Sudborough, *Bandwidth constrained* NP*-complete problems*, Theoret. Comput. Sci., 41 (1985), pp. 141–167.

[12] 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, 26 (1979), pp. 675–684.

[13] C. H. PAPADIMITRIOU, *The NP-completeness of the bandwith minimization problem*, Computing, 16 (1976), pp. 237-267.
[14] J. B. SAXE, *Dynamic programming algorithms for recognizing small bandwidth graphs in polynomial time*, SIAM J. Algebraic Discrete Methods, 1 (1980), pp. 363-369.
[15] I. H. SUDBOROUGH, *Bandwidth constraints on problems complete for polynomial time*, Theoret. Comput. Sci., 26 (1983), pp. 25-52.
[16] S. TRIMBERG, *Automated chip layout*, IEEE Spectrum, 19 (1982), pp. 38-45.
[17] M. YANNAKAKIS, *A polynomial algorithm for the min cut linear arrangement of trees*, J. Assoc. Comput. Mach., 32 (1985), pp. 950-959.
[18] T. LENGAUER, *Black-white pebbles and graph separation*, Acta Inform., 16 (1981), pp. 99-113.