A Polynomial Algorithm for the Min-Cut Linear Arrangement of Trees

MIHALIS YANNAKAKIS

AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. An algorithm is presented that finds a min-cut linear arrangement of a tree in $O(n \log n)$ time. An extension of the algorithm determines the number of pebbles needed to play the black and white pebble game on a tree.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures; routing and layout*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms; trees*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Cutwidth, min-cut linear arrangement, pebbling

1. *Introduction*

1.1. THE PROBLEM. A *layout* (or *linear arrangement*) of a graph $G = (N, E)$ is a one-to-one mapping L of the nodes of the graph to the first $|N|$ positive integers $\{1, \ldots, |N|\}$. If p is a point of the real line, the *cut* of the layout L at p, $cut_L(p)$, is the number of edges that cross over p; that is, the number of edges (u, v) of G with $L(u) < p < L(v)$. The *cutwidth* of L, denoted $\gamma(L)$, is the maximum cut of L over all real points. The min-cut linear arrangement problem (MINCUT for short) is to find a linear arrangement for a given graph with the minimum cutwidth. The cutwidth of such an optimal linear arrangement for a graph G is called the cutwidth of G, denoted $\gamma(G)$. In Figure 1, we show a layout L of a tree. The cutwidth of L is 2 and it occurs at several points; for example, between nodes c and d.

1.2 BACKGROUND. The MINCUT problem for general graphs is NP-complete [11, 13]. The restriction of the problem to trees has been open for some time. Lengauer described an approximation algorithm in [16] that produces a layout with cutwidth at most twice the optimal. He also determined exactly the cutwidth of complete k-ary trees. F. R. K. Chung studied the properties of optimal layouts [2]. M. Chung et al. present in [5] an algorithm that solves the MINCUT problem on trees in time $O(n(\log n)^{d-2})$ where d is the maximum degree of the tree. Thus, their algorithm works in polynomial time for bounded degree trees, but exponential time in general. Dolev and Trickey [7] study the MINCUT problem on trees and give an $O(n \log n)$ algorithm for a planar version of it, where no edge crossings are allowed if the tree is drawn as in Figure 1 with all the edges above the line where the nodes lie. (For example, the layout of Figure 1 is not planar because the edges

Author's address: AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974.

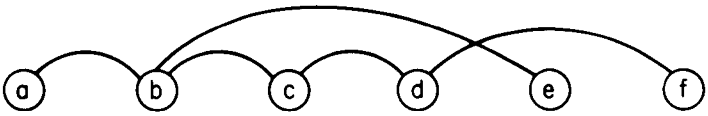


FIGURE 1

(b, e) and (d, f) cross.) We improve the solution to this planar version to linear time.

There are two other well-studied linear arrangement problems: the BANDWIDTH and the MINSUM problem (sometimes called the Optimal Linear Arrangement problem). If L is a layout, the *length* $l(e)$ of an edge $e = (u, v)$ is $|L(u) - L(v)|$ (i.e., the distance between u and v in L). The *bandwidth* of L is the maximum length of an edge in L. The BANDWIDTH problem is to find a layout with the smallest bandwidth. The MINSUM problem is to find a layout that minimizes the sum of the lengths of the edges. Both problems are NP-complete for general graphs [11]. When restricted to trees, the BANDWIDTH problem remains NP-complete [12], but the MINSUM problem can be solved in polynomial time: References [4], [14], and [25] give algorithms that run, respectively, in time n^3, $n^{2.2}$, and $n^{1.58}$. The three linear arrangement problems differ in the cost function that has to be optimized. From a formal point of view, the MINSUM problem stands in the same relationship with the MINCUT and the BANDWIDTH problems: The MINSUM problem minimizes the sum of a set of quantities (the lengths of the edges) whose maximum has to be minimized in BANDWIDTH. It follows easily from the definitions that the cut of a layout in the interval $(i, i + 1)$ between two consecutive integers i, $i + 1$ remains constant and is at least as large as the cut at i or $i + 1$. Let $cut_L(I_i)$ be the cut of L over the interval $I_i = (i, i + 1)$. Minimizing the $\max_p[cut_L(p)]$ over all layouts L is equivalent to minimizing $\max_{i=1,\dots,n}[cut_L(I_i)]$. It is not difficult to see that the sum of the lengths of the edges in a layout is equal to the sum of the cuts over the intervals I_i. Thus, the MINSUM problem calls for the minimization of the sum of a set of quantities $(cut_L(I_i))$ whose maximum has to be minimized in the MINCUT problem. Despite this similarity between BANDWIDTH and MINCUT, the latter problem can be solved in polynomial time (on trees), as we shall show.

There has been much work on the problem of embedding graphs in the plane motivated by VLSI applications [1, 7, 9, 15, 26]. The graph models a circuit with the nodes as the active elements and the edges as the connecting wires. In some approaches to VLSI design, the active elements are placed in rows or on a single line [8, 10, 24]. When the nodes are placed on a line (the bottom row), the cutwidth of the layout gives the number of tracks needed to route the wires above the line. In Figure 2, we show such an assignment of tracks for the layout of Figure 1.

Of course, several simplifying assumptions are made in this modeling: The active elements are assumed to have the same height, the wires have the same width, and their required spacing is the same. Another abstraction of this model is that we ignore the order in which wires enter the active elements (or assume that we can choose this ordering and rearrange the wires inside the box that represents the active element). If the nodes have small degree, this does not make a substantial difference in the width of the circuit.

Another source of interest in the MINCUT problem is its close relationship to a number of other arrangement problems, especially when restricted to trees: the black and white pebble game [5, 6, 17–19, 23], graph separation [17], topological bandwidth [3, 21], and graph searching [20, 22]. We describe the pebble game in a later section; we do not talk about the other problems in this paper. The black

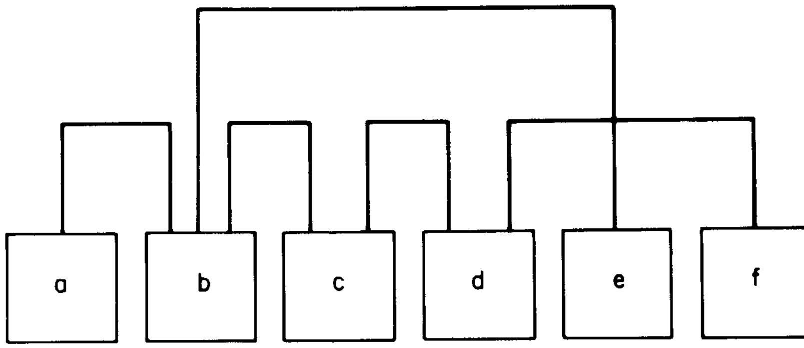


FIGURE 2

pebble game models register allocation in straight-line programs. The black and white pebble game is the nondeterministic version of the black pebble game. The black and white pebble game on trees has been studied in [5], [18], [19], and [23]. It turns out that the problem of determining the minimum number of pebbles needed to play the black and white pebble game on a tree is a special case of a generalized version of the cutwidth problem. In this generalized version, the nodes have heights associated with them; the cut of a layout at a point where a node is embedded is the sum of the height of the node and the number of edges that cross over the point. The problem is again to find a layout that minimizes the maximum cut. The usual cutwidth problem corresponds to the case where all nodes have height 0.

1.3 OVERVIEW OF THE PAPER. In Section 2 we describe a simple dynamic programming strategy for laying out a tree; we call it the *disjoint* strategy. It is essentially the same strategy as the one used by Lengauer [16] for his approximation algorithm and by Dolev and Trickey [7] for the planar MINCUT problem. We show how to solve the planar problem in linear time. Sections 3–5 prepare the ground for the cutwidth algorithm. We deal directly with the generalized problem where the nodes have heights. The reason is that the algorithm for the usual cutwidth problem (all heights 0) is already quite complicated; the generalization itself does not introduce any further significant complications, but is needed for the black and white pebble result. In Section 3 we introduce cut-functions; these are functions that look like the functions $cut_L(\cdot)$ of layouts L. Cut-functions play a central role in the development of the algorithm and the proof of its correctness. In Section 4 we study some of the properties of the disjoint strategy. In Section 5 we define a more complicated cost function of layouts. This cost function includes the cutwidth of the layout but, in general, contains many more (a linear number) parameters. Section 6 contains the algorithm and the proof of correctness. In Section 7 we analyze its running time. In Section 8 we see how the generalized algorithm can be used to lay out optimally on a line a tree circuit whose active elements have different heights. In Section 9 we show that the black and white pebble problem is a special case of the cutwidth problem with heights.

2. *The Disjoint Strategy and the Planar MINCUT Problem*

The reason that many problems that are NP-complete on general graphs become polynomial when restricted to trees is that trees have a nice recursive structure. This structure can often be used to design a dynamic programming algorithm. The tree is rooted at some node and then the algorithm proceeds bottom up in the tree.

At every node an optimal solution and some information is computed for the subtree rooted at the node. The problem for the whole tree reduces then to a one-level problem. Assuming we are at a node v with sons $x_1, \ldots, x_d$ (in the rooted tree), the following problem has to be solved: Given the optimal solution and the information for each subtree rooted at a son x_i, compute efficiently the optimal solution and the information for the subtree rooted at v. The task then is to find what is the right information in order to be able to solve the one level problem and which node should be chosen to be the root of the tree. (Sometimes, as in the algorithm, we give for the MINCUT problem, the choice of the root is irrelevant.)

Let us give some terminology first for layouts of rooted trees. Given a layout of a rooted tree, the *heavy side* with respect to the root is the one in which the maximum cut occurs; the other side is the *light side*. If the maximum cut occurs on both sides, then we say that the layout is *balanced*; otherwise, it is *unbalanced*. If the layout is balanced, then we choose arbitrarily heavy and light sides. (This choice will be refined in a later section.) For example, the layout of the trivial tree, a tree that contains only one node, the root, is balanced because the cutwidth 0 occurs on both sides. If we have a layout of a rooted tree, we can reverse the order of the nodes to obtain another layout with the same cutwidth. We do not distinguish a layout from its reverse. When layouts of different subtrees are combined, we only specify the orientation of the heavy sides of the layouts.

Suppose now that we want to use the dynamic programming method to solve the MINCUT problem and that we have already picked the root. For the one-level problem, we have a root v from which hang a number of subtrees $T_1, \ldots, T_d$. We have computed for each subtree T_i a layout L_i and some additional information and we want to combine the L_i's in an optimal fashion. A first approach for the one-level problem is to place the computed solutions (layouts) for the subtrees in disjoint intervals on either side of the root v in some order. In order to minimize the cutwidth of the resulting layout, we sort the layouts of the subtrees according to their cutwidth with ties broken according to whether they are balanced; that is, $i < j$ if $\gamma(L_i) > \gamma(L_j)$ or $\gamma(L_i) = \gamma(L_j)$ and [L_j balanced implies L_i balanced]. The *disjoint strategy* places the layouts of the subtrees as in Figure 3, where the light side of each subtree faces the root v. That is, the one side contains the odd-numbered subtrees, the other side contains the even-numbered subtrees, within each side the subtrees are ordered according to their index, and the layout L_i of each subtree is placed so that its cutwidth occurs in the side with respect to x_i that does not include v. It can be easily seen by an induction that the layout L of a tree produced by the disjoint strategy is planar (for any choice of the root); that is, there is no pair of edges (a, b), (c, d) with $L(a) < L(c) < L(b) < L(d)$.

We denote by δ the cutwidth of the layout produced by the disjoint strategy. Let the bit b_i indicate if T_i is balanced. The maximum cut δ_o of the disjoint layout in the side with the odd subtrees is

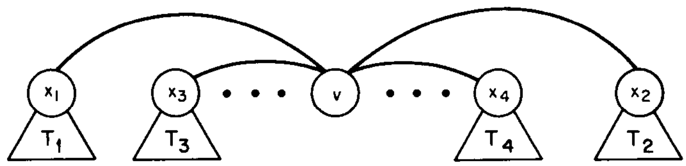
$$\delta_o = \max\{\gamma(L_{2t-1}) + t - 1 + b_{2t-1} \mid t \geq 1\}, \tag{1}$$

and the maximum cut δ_e in the side with the even subtrees is

$$\delta_e = \max\{\gamma(L_{2t}) + t - 1 + b_{2t} \mid t \geq 1\}. \tag{2}$$

From the ordering of the subtrees it follows that $\delta_o \geq \delta_e$. Therefore, the cutwidth of the disjoint layout is $\delta = \max\{\delta_o, \delta_e\} = \delta_o$. We say that the cutwidth δ occurs over a tree T_i if $\delta = \gamma(L_i) + \lceil i/2 \rceil - 1 + b_i$. It occurs on the outside of T_i if $\delta = \gamma(L_i) + \lceil i/2 \rceil - 1$; in this case, L_i cannot be balanced.

FIGURE 3



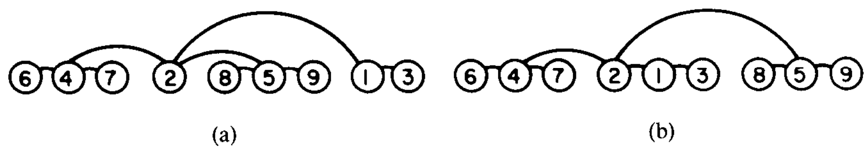


FIGURE 4

The disjoint layout is not optimal because it may either give the wrong maxcut, or it might achieve the cutwidth but be balanced when the tree has another unbalanced layout. In Figure 4a we show a tree rooted at node 1 layed out by the disjoint strategy; the layout has a maximum cut of 3. Figure 4b shows an optimal layout with cutwidth 2.

The following lemma shows that the disjoint strategy is optimal among strategies that place consecutively the nodes of each subtree.

LEMMA 1. *Let T be a tree rooted at node v and L a layout of T in which the nodes of every subtree rooted at a son of v are arranged consecutively. For each son x_i of v let L_i be the layout in L of the subtree T_i rooted at x_i. Let DL be the layout obtained by sorting the L_i's according to $\gamma(L_i) + b_i$ and arranging them as in Figure* 3. *Then,* (1) $\gamma(L) \geq \gamma(DL)$, *and* (2) *if* $\gamma(L) = \gamma(DL)$ *and DL is balanced then so is L.*

PROOF. Assign a rank to every subtree in L by numbering them on each side from the outside in towards the root v. Thus, the outermost subtree in each side gets rank 1; the second outermost, rank 2; and so forth. The maximum cut over a subtree T_i of rank r_i in L is at least $\gamma(L_i) + b_i + r_i - 1$. Suppose that the maximum cut of DL occurs over a subtree T_{2t-1} (the maximum cut of DL always occurs on the odd side). From the pigeonhole principle, at least one of the first $2t - 1$ subtrees (in the ordering by $\gamma(L_i) + b_i$) must have rank at least t in L. It follows then from the ordering and the formulas that $\gamma(DL) = \gamma(L_{2t-1}) + b_{2t-1} + t - 1 \leq \gamma(L)$.

Suppose that DL is balanced, and let T_{2t} be a subtree in the even side of DL over which the maximum cut occurs. If one of the first $2t$ subtrees has rank greater than t in L, then $\gamma(L) > \gamma(DL)$. Thus, $\gamma(L) = \gamma(DL)$ implies that the first $2t$ subtrees have all rank at most t. From the pigeonhole principle, two of these subtrees have rank t, and therefore are on opposite sides. From the ordering and the formulas it follows that $\gamma(L)$ occurs over both of these subtrees, and thus L is balanced. □

From the lemma it follows that the cutwidth of the disjoint layout would not increase if the subtrees were ordered according to $\gamma(L_i) + b_i$ rather than the lexicographic ordering of the pairs $(\gamma(L_i), b_i)$ that we used. (The latter ordering is a refinement of the first.) The reason for using the lexicographic ordering becomes apparent in the following sections.

Lengauer [16] showed that the disjoint strategy produces a layout within a factor of 2 of the optimal, for any choice of the root. Furthermore, for a complete binary tree this ratio of 2 is achieved (no matter which node we choose to root the tree in applying the disjoint strategy): The disjoint layout has cutwidth equal to the height of the tree, whereas the optimal layout has cutwidth approximately half of the height. Dolev and Trickey [7] proved that for an appropriate choice of the root, the layout produced by the disjoint strategy has minimum cutwidth among all planar layouts. This is based on the following two facts: (1) If L is a planar layout of a tree T, there is a node v such that if we root T at v the nodes of every rooted subtree are consecutive in L, and (2) the disjoint strategy is optimal among strategies that place the nodes of every rooted subtree consecutively (Lemma 1). The algorithm of [7] for the PLANAR MINCUT works in two phases. In the first phase the tree is rooted arbitrarily at some node and the cutwidth of the disjoint layout with this root is computed. In the second phase the rooted tree is traversed from the top down to determine the node whose choice as a root would minimize the cutwidth. Both [16] and [7] give $O(n \log n)$ as the running time of the disjoint strategy. In the remainder of this section, we give a simple linear time algorithm to solve the PLANAR MINCUT problem.

If L is a layout of a rooted tree, denote the quantity $\gamma(L) + b(L)$ by $\gamma b(L)$ (b is the balance bit), and call it the *modified cutwidth* of L. To find the optimal planar layout, we apply the disjoint strategy with the ordering according to modified cutwidths. However, we do not fix a priori the root. The nodes are processed in some order (described below); the last node to be processed acts as the root. To achieve the linear running time, we use bucket sorting to sort the modified cutwidths of the subtrees at each node. All the nodes share the same buckets. More specifically, we have one bucket for every possible modified cutwidth i, $1 \leq i \leq n$. Each node is assigned a number i at some stage of the algorithm and is then inserted to the appropriate bucket. If a node u has degree $d(u)$ and is not the root of the tree, there are $d(u)$ possibilities for the subtree rooted at u depending on the choice of the root v of the tree; that is, there are $d(u)$ choices for the father of u. The number assigned to u is the modified cutwidth for one of these possible subtrees. For every node there is a queue in which the nodes adjacent to it are inserted. As soon as there are $d(u) - 1$ nodes in the queue, the modified cutwidth of the node is calculated from those of the nodes in its queue, using the formulas (1) and (2) of the disjoint strategy.

The algorithm is as follows:

(1) Initialization: Insert every leaf in the bucket 1. The father of a leaf is the unique node adjacent to it.
(2) While there is a nonempty bucket, do the following. Delete a node x from the smallest nonempty bucket, mark it, and insert it into the queue of its father y. If the number of elements in the queue is one less than the degree of y, then compute the modified cutwidth for y according to the formulas (1) and (2) of the disjoint strategy and insert y in the appropriate bucket. The father of y is the unique node adjacent to y that is not marked.
(3) Let v be the node that is marked last and i its modified cutwidth. Node v is chosen as the root of the tree and the minimum planar cutwidth is i.

For example, consider the execution of the algorithm on the tree of Figure 4. Initially, the leaves 3, 6, 7, 8, 9 are placed in bucket 1. When they are deleted and marked, the modified cutwidths of nodes 1, 4, 5 are computed: $\gamma b(1) = 1$, $\gamma b(4) = \gamma b(5) = 2$. Thus, node 1 is marked before nodes 4 and 5, and 4, 5 are marked after all the leaves. When one of 4, 5 is marked, say node 4, the modified cutwidth of node 2 is computed: $\gamma b(2) = 2$. The last node to be marked is either 2 or 5, and the planar cutwidth is 2. Figure 4b shows the disjoint layout of the tree rooted at node 2 or node 5.

THEOREM 1. *The algorithm computes correctly the planar cutwidth of the tree and runs in linear time.*

PROOF. Let $\delta_u(T)$ be the cutwidth of the disjoint layout of T rooted at node u. The planar cutwidth of T is $\min_u \delta_u(T)$ [7]. For the correctness of the algorithm, we have to show the following facts: (1) The number returned by the algorithm (the modified cutwidth of the chosen root v) is $\delta_v(T)$; (2) for all nodes u, $\delta_u(T) \geq \delta_v(T)$.

Form a directed graph D as follows: D has the same nodes as T and has an arc $x \rightarrow y$ if x was used in the calculation of the modified cutwidth assigned to y. That is, there is an arc from x to y if y was assigned as the father of x and x was marked at the time that we computed the modified cutwidth of y and inserted it in a bucket. Clearly, D is a directed subgraph of T. Every node u has $d(u) - 1$ incoming arcs in D, where $d(u)$ is the degree of u in T. If T has n nodes and $e = n - 1$ edges, the number of arcs of D is $\sum_u d(u) - 1 = 2e - n = n - 2$. Therefore, D is obtained from T by deleting one edge and directing every other edge from a node to its assigned father. Since every node has outdegree at most one in D, it follows that D consists of two rooted trees. An easy induction can show that the number $\gamma b(u)$ assigned at step 2 of the algorithm to a node u is equal to the modified cutwidth of the subtree of D rooted at u. Since the chosen root v is marked last, it is not used in the calculation of the modified cutwidth of another node; thus v is a root of D. Let w be the other root of D. Since every node u other than v and w has $d(u)$ incident arcs in D, the edge of T missing from D is the edge (w, v). Since D does not contain the arc $w \rightarrow v$, w was marked after the other nodes adjacent to v.

From the formulas (1) and (2) for the disjoint layout, we can deduce that the cutwidth of a rooted tree is at least as large as the modified cutwidth of any of its proper subtrees. From this it follows that the modified cutwidth of a node is at least as large as that of any of its sons in D. Consequently, the minimum nonempty bucket does not ever decrease in the course of the algorithm. Therefore, the nodes are marked in the order of their assigned modified cutwidths. Thus, if we root T at v, the subtree rooted at w has the largest modified cutwidth among all subtrees rooted at the sons of v; that is, in Figure 3, w is the first son x_1 of v. Let D_v, D_w denote the two trees of D rooted, respectively, at v and w. From the definition of the disjoint strategy, if we delete the first subtree D_w, from the disjoint layout of the whole tree T rooted at v, we obtain the disjoint layout of the remaining tree D_v. The maximum cut of the disjoint layout of T rooted at v over D_w is $\gamma b(w)$; the maximum cut over the rest of the layout is $\gamma b(v)$. Since w was marked before v, we have $\gamma b(w) \leq \gamma b(v)$, and therefore $\delta_v(T) = \max\{\gamma b(w), \gamma b(v)\} = \gamma b(v)$. Thus, the algorithm correctly computes $\delta_v(T)$.

It remains to show that $\delta_u(T) \geq \delta_v(T)$ for any other node u. If u is a descendant of w in T rooted at v, then T rooted at u contains D_v as a proper subtree. Therefore, $\delta_u(T) \geq \gamma b(v) = \delta_v(T)$. Suppose that in T rooted at v, node u is a descendant of another son x_i of v. Let S be the (rooted) tree obtained from D_v by replacing the subtree T_i rooted at x_i by D_w; S is a proper subtree of T rooted at u. Therefore, $\delta_u(T) \geq \gamma b(S)$. Since w was marked after x_i, the modified cutwidth of D_w is at least as large as that of T_i. From expressions (1) and (2) it follows then that $\gamma b(D_v) \leq \gamma b(S)$. Therefore, $\delta_u(T) \geq \delta_v(T)$.

The linear running time of the algorithm follows from the fact that the minimum nonempty bucket does not decrease in the course of the algorithm and the nodes are marked (and inserted in the queues of the nodes) in the order of their modified cutwidth. □

The algorithm can be extended in a straightforward way to compute in linear time the actual layout that achieves the planar cutwidth: We just have to keep for every node the layout of the subtree rooted at this node as a doubly linked list. In step 2, when the modified cutwidth of a new node y is computed, we concatenate in the appropriate order (dictated by the disjoint strategy) the lists of y's children.

3. *Cut-Functions*

First, we redefine terms for the general case when the nodes have heights. Let T be a tree, where every node u has an integer $h(u)$ associated with it; $h(u)$ is called the *height* of u. Let L be a layout of the tree. The cut-function $cut_L(\cdot)$ of L is a function from the reals to the integers defined as follows. If p is a point of the real line where no node is embedded, then $cut_L(p)$ is the number of edges that cross over p; that is, the number of edges (u, v) with $L(u) < p < L(v)$. If a node w is embedded at p, then $cut_L(p)$ is equal to the sum of $h(w)$ and the number of edges that cross over p. The cutwidth $\gamma(L)$ of L is defined as before to be the maximum cut over all real points, and the cutwidth $\gamma(T)$ of T is the minimum cutwidth of a layout of T.

The cut-function of a layout is a function with the following properties:

(1) It maps the reals to the integers.
(2) It is piecewise constant with a finite number of breakpoints. The breakpoints are the points at which the nodes are embedded. Note that the cut-function does not necessarily change value at a point where a node is embedded. However, we still consider such a point as a breakpoint; that is, we have a 1–1 correspondence between breakpoints and nodes.
(3) If a is the smallest breakpoint and b the largest, then the function is 0 outside the (closed) interval $[a, b]$. Furthermore, the function is positive inside the interval $[a, b]$, except possibly at some breakpoints. This property follows from the fact that a tree is a connected graph, and therefore, at least one edge crosses over any point at which no node is embedded.

We call any function f satisfying these properties, a *cut-function*. The interval $[a, b]$ between the smallest and the largest breakpoint of f is called the *support* of f. The breakpoints partition the support of f into a number of open intervals, called the *gaps* of f. The value of f is constant over each gap. We also use terms from layouts for (arbitrary) cut-functions. Thus, we also refer to the breakpoints as the nodes of the function, to the value of the function at a point p as the cut at p, and so on. The cutwidth $\gamma(f)$ of a cut-function f is the maximum cut (value) of f. The cutwidth is a nonnegative number. From property (3), if the cutwidth is 0, then the cut-function f must have only one node, and the cut of f at this single node must be 0 or negative; such a cut-function is called *degenerate*. For example, the cut-function of a layout of a tree that has a single node u with height $h(u) \leq 0$ is degenerate. If f is not degenerate, then $\gamma(f) > 0$. Usually, one of the breakpoints will be distinguished as the *root* of the function f; if f is the cut-function of a layout of a rooted tree T, then the root of f is the breakpoint that corresponds to the root of T. The *sides* of a rooted cut-function f are the two *open* intervals from the root to $\pm\infty$. The function is *balanced* if the cutwidth occurs on both sides; that is, there are points p, q, with $p <$ root $< q$ such that $f(p) = f(q) = \gamma(f)$. As in Section 2, we distinguish between a heavy and a light side. If the cutwidth occurs on one side but not the other, then the heavy side is the one with the cutwidth, and the other one is the light side. In all other cases we choose arbitrarily heavy and light sides. Note that, now that nodes have heights, there is the possibility that the cutwidth occurs only at the root; in this case the function is not balanced, but the designation of sides is chosen arbitrarily.

We are going to develop the algorithm and argue about its correctness using cut-functions rather than layouts. The reason is that we need to perform certain operations that are not physically realizable on layouts. For example, one such operation is restriction of a cut-function f to an interval. The *restriction* of f to the interval $[x, y]$ is the function g, which agrees with f in the interval $[x, y]$, and is 0 outside the interval. If f has a root v, it will always be the case that v is in $[x, y]$; node v is also the root of g. It follows immediately from the definition that g is also a cut-function. The restriction operation cannot be carried out directly on a layout. That is, if L is a layout with cut-function f, then it may not be possible to delete nodes and edges from L in order to obtain another layout with cut-function g.

3.1 COMBINATIONS OF CUT-FUNCTIONS AND LAYOUTS. Let T be a tree with root v. Let $x_1, \ldots, x_d$ be the children of the root, and let $T_1, \ldots, T_d$ be the subtrees rooted, respectively, at $x_1, \ldots, x_d$. Let $L_1, \ldots, L_d$ be layouts for these subtrees with cut-functions $F_1, \ldots, F_d$. The nodes are assumed, in general, to have heights; the heights of nodes in the subtrees are taken into account in the corresponding cut-functions. We say that a layout L of T is a *combination* of the root v and the layouts $L_1, \ldots, L_d$ if the nodes of each subtree T_i are ordered by L either in the same or in the reverse way as by L_i; that is, the *relative* order of the nodes in each subtree is the same in L as in L_i. We call such an ordering of all the nodes, a *proper* ordering. Of course, every layout L of T is a combination of the root v and some layouts L_i of the subtrees; just let L_i be the layout of T_i that orders the nodes of T_i in the same way as L.

Given a node v with height $h(v)$, and d arbitrary cut-functions, $F_1, \ldots, F_d$ with roots $x_1, \ldots, x_d$, respectively, we define a combination of v and the F_i's. This notion is defined in such a way that the cut-function of a combination L of v and the L_i's is a combination of v and the cut-functions of the L_i's. Formally, a cut-function F is a combination of v and the F_i's if there are mappings $\pi_1, \ldots, \pi_d$ from R to R and a point $\pi(v)$ on the real line satisfying the following conditions:

(i) Each π_i is 1–1, onto and monotone (i.e., either, for all $x > y$, $\pi_i(x) > \pi_i(y)$, or, for all $x > y$, $\pi_i(x) < \pi_i(y)$).
(ii) If x is a node of F_i and y a node of F_j, then $\pi_i(x) \neq \pi_j(y) \neq \pi(v)$.
(iii) For p a point on the real line, let $e(p)$ be the number of open intervals $(\pi_i(x_i), \pi(v))$ or $(\pi(v), \pi_i(x_i))$ that contain p. If $p \neq \pi(v)$, then $F(p) = e(p) + \sum_i F_i(\pi_i^{-1}(p))$; if $p = \pi(v)$, then $F(p) = h(v) + e(p) + \sum_i F_i(\pi_i^{-1}(p))$.

The function $F_i \circ \pi_i^{-1}$ is just a "stretched" and possibly reversed version of F_i: Let x, y be two consecutive nodes of F_i; since π_i is 1–1, onto and monotone, the image under π_i of the gap (x, y) is the open interval between $\pi_i(x)$ and $\pi_i(y)$. The (constant) value of F_i in the gap (x, y) is equal to the value of $F_i \circ \pi_i^{-1}$ in the interval between $\pi_i(x)$ and $\pi_i(y)$. Thus, $F_i \circ \pi_i^{-1}$ is a cut-function, whose nodes (respectively, gaps) are in 1–1 correspondence with the nodes (respectively, gaps) of F_i. It follows that a combination F is a cut-function. The nodes (breakpoints) of F are the images under π and the π_i's of v and the nodes of the F_i's. The point $\pi(v)$ is designated as the root of F.

A combination F is determined by the images of v and the nodes of the F_i's. Once we have chosen these images in a 1–1 monotone way, we can extend each π_i from the nodes of F_i to a 1–1, onto, monotone function on all the real numbers. The functions $F_i \circ \pi_i^{-1}$, and therefore also F, do not depend on what particular extensions of the π_i's we choose.

PROPOSITION 1. *Let T be a tree with root v, which has children $x_1, \ldots, x_d$. For each $i = 1, \ldots, d$, let T_i be the subtree rooted at x_i, L_i a layout of T_i, and F_i the cut-function of L_i.*

(1) *If a layout L of T is a combination of v and the L_i's, then the cut-function F of L is a combination of v and the F_i's.*
(2) *Conversely, if a cut-function F is a combination of v and the F_i's, then F is the cut-function of some combination L of v and the L_i's.*

PROOF

(1) We define $\pi(v)$ and the mappings π_i in the obvious way: Take $\pi(v) = L(v)$. Let π_i map every node of F_i to the point where L embeds the corresponding node of L_i. Since L preserves the relative order of the nodes in each L_i, π_i maps the nodes of F_i in a 1–1 monotone (increasing or decreasing) way. Clearly, each π_i can be extended to an 1–1, onto, monotone function from R to R. Thus, condition (i) is satisfied. Since L embeds the nodes of T into distinct points, condition (ii) is also satisfied. As for condition (iii), let p be a point on the real line. The cut of L at p due to a subtree T_i is the number of edges of T_i that cross over p if no node of T_i is embedded at p; if some node u of T_i is embedded at p, then this number is increased by the height of u. Since π_i is monotone, an edge of T_i crosses over p in L iff the same edge crosses over $\pi_i^{-1}(p)$ in L_i. Thus, the cut of L at p due to T_i is equal to $F_i(\pi_i^{-1}(p))$. The cut of L at p is the sum of the following quantities: (a) the cut due to the subtrees—this is equal to $\sum_i F_i(\pi_i^{-1}(p))$; (b) the number of edges (v, x_i) incident to the root that cross over p—this is equal to $e(p)$; (c) if v is embedded at p, the height $h(v)$ of v. Thus, $cut_L(p) = F(p)$.

(2) Let F be a combination of v and the F_i's, and let $\pi(v)$ and the mappings π_i be as in the definition. Construct a layout L of T in the obvious way: The root v is embedded at $\pi(v)$, and each node of subtree T_i is embedded at the point where the corresponding node of F_i is mapped by π_i. From conditions (i) and (ii), all nodes are embedded at distinct points. Also, since each π_i is monotone, L preserves the relative order of the nodes in each subtree. By the same argument as in part (1), $cut_L = F$. □

The basic problem we solve is: Given a node v with height $h(v)$ and (rooted) cut-functions $F_1, \ldots, F_d$, find their optimal combination F. The figures of merit that we have seen so far (cutwidth and balance), as well as those we define later on, are not affected by "stretching" and/or reversal of a cut-function (i.e., composing a cut-function with a 1–1, onto, monotone mapping). In the next section we start our investigation by examining the properties of the disjoint strategy. To simplify notation, we drop the mappings π_i, and informally treat a combination F simply as the cut-function determined by a proper ordering of all the nodes (v and the nodes of the F_i's), that is, an ordering that preserves the relative order of the nodes of each F_i. We call the F_i's, the subfunctions of F, and when no confusion will arise, we usually identify them with their transformed versions $F_i \circ \pi_i^{-1}$ within F. Thus, a combination F is obtained by choosing a proper ordering of all the nodes and then superimposing (summing) the subfunctions F_i, the "edges" (v, x_i), and the height $h(v)$ at the root v.

4. *Properties of the Disjoint Combination*

Although the disjoint strategy can produce a layout of cutwidth as much as twice the optimal, in the one-level problem it is never that far from optimal, and in some cases it is even optimal. In this section we prove some properties of the disjoint strategy for the one-level problem, in the general setting where we combine arbitrary cut-functions. Let v be a (root) node with height $h(v)$, and let $F_1, \ldots, F_d$ be cut-functions with roots $x_1, \ldots, x_d$. Let γ_i be the cutwidth of F_i, and b_i the balance bit (the bit that indicates whether F_i is balanced). We assume that the F_i's are ordered by cutwidth with ties broken according to the balance bit. The *disjoint combination DC* of v and the F_i's is defined as in the case of layouts. Namely, the nodes of the subfunctions F_i are placed in disjoint intervals on either side of the root v, ordered according to their index as in Figure 3. Every subfunction is oriented so that it faces v with its light side. The maximum cut of DC on the odd side is, as in Section 2, $\delta_o = \max\{\gamma_{2t-1} + t - 1 + b_{2t-1} \mid t \geq 1\}$, and the maximum cut on the even side is $\delta_e = \max\{\gamma_{2t} + t - 1 + b_{2t} \mid t \geq 1\}$. Now that the root v has height, the cutwidth of DC is $\delta = \max\{h(v), \delta_o, \delta_e\} = \max\{h(v), \delta_o\}$. Terms that we defined for the disjoint layout are extended to the disjoint combination of cut-functions in a straightforward way. For example, the cutwidth δ occurs in the disjoint combination DC over a subfunction F_i if $\delta = \gamma_i + \lceil i/2 \rceil - 1 + b_i$. It occurs on the outside of F_i if $\delta = \gamma_i + \lceil i/2 \rceil - 1$; in this case F_i is not balanced. We denote the cutwidth of the optimal combination F_{opt} by γ. Of course, $\gamma \leq \delta$.

LEMMA 2. *If δ occurs at the root or on the outside of some subfunction, then $\delta = \gamma$.*

PROOF. If δ occurs at the root, then $\delta = h(v) \geq \gamma$. Suppose δ does not occur at the root. Without loss of generality, we can assume that it occurs on the outside of an odd subfunction F_{2t-1} ($t \geq 1$); if it occurs on the outside of an even subfunction F_{2t}, then also it occurs on the outside of F_{2t-1}. Thus, $\gamma_{2t-1} + t - 1 = \delta$.

Let F_{opt} be an optimal combination with maxcut γ. Look at the subfunctions $F_1, \ldots, F_{2t-1}$ within F_{opt}; for each one of them there is a point u_i (in the domain of F_{opt}) where F_i contributes a cut γ_i to F_{opt}. We can take, without loss of generality, the points u_i to be distinct from each other, from v, and from the nodes of other subfunctions (i.e., u_i may be a node of F_i, but not v or a node of another F_j). In the trivial case in which F_i is a degenerate cut-function, that is, it has only one node, its root x_i, with cut ≤ 0, we take u_i to be a point right next to x_i in the interval (x_i, v). In all other cases, u_i will be a point in the support of F_i.

Let u_i be the first point to the left of the root v, and u_j the first point to the right of v. Let l be the number of all subfunctions (not only the first $2t - 1$) that have at least one node to the left of u_i (inclusive) and r the number of subfunctions with a node to the right of u_j (inclusive). If a subfunction F_k with $k \leq 2t - 1$ had all its nodes between u_i and u_j, then u_k would have to lie between u_i and u_j contradicting our definition of u_i, u_j. Thus, $l + r \geq 2t - 1$.

Now, if F_k, $k \neq i$, has a node to the left of u_i, then either x_k is to the left of u_i, in which case u_i is in the interval (x_k, v), or x_k is to the right of u_i, in which case F_k has nodes on both sides of u_i and therefore, its cut at u_i is at least 1. Therefore, the value of F_{opt} at u_i is at least $\gamma_i + l - 1$: γ_i from F_i, and 1 for each of the subfunctions F_k, $k \neq i$, which have a node to the left of u_i. Similarly, the value of F_{opt} at u_j is at least $\gamma_j + r - 1$. Therefore,

$$\gamma \geq \max\{\gamma_i + l - 1, \gamma_j + r - 1\}$$

$$\geq \left\lceil \frac{\gamma_i + l + \gamma_j + r - 2}{2} \right\rceil \geq \left\lceil \frac{2\gamma_{2t-1} + 2t - 3}{2} \right\rceil = \gamma_{2t-1} + t - 1 = \delta. \qquad \square$$

COROLLARY 1. $\gamma \geq \delta - 1$.

PROOF. If δ occurs at the root, then $\delta = \gamma$. Otherwise, let F_{2t-1} be a function over which the maxcut occurs; $\delta \le \gamma_{2t-1} + t$. Arguing as in Lemma 2, we have $\gamma \ge \gamma_{2t-1} + t - 1 \ge \delta - 1$. □

LEMMA 3. *If $\delta = \gamma + 1$, then in the disjoint combination the maxcut occurs over exactly one subfunction, and the optimal combination is balanced.*

PROOF. Suppose that $\delta = \gamma + 1$ and let F_{2t-1} be a subfunction over which the maxcut occurs; F_{2t-1} is balanced and $\delta = \gamma_{2t-1} + t$. Let F_{opt} be an optimal combination. For each one of the first $2t - 1$ subfunctions, choose a point u_i where the maximum cut of the subfunction is achieved; if F_i is balanced, then we choose a second point u_i' on the other side of its root x_i with the same cut. Note that in case F_i is balanced, we can take u_i and u_i' to be distinct from the root x_i of the subfunction. If F_i is degenerate, we take u_i' to be a point right next to x_i on the other side. In all other cases, u_i' is also a point in the support of F_i. We choose these points to be distinct from each other, from v, and from nodes of other subfunctions as in Lemma 2. Consider the two points closest to v on each side, and let l, r be defined as in Lemma 2.

Case 1. The two points closest to the root belong to different subfunctions.

Let u_i, u_j be these points. We have, $\gamma \ge \gamma_i + l - 1$, $\gamma \ge \gamma_j + r - 1$ and $l + r \ge 2t - 1$, as in Lemma 2. Since $\gamma_i, \gamma_j \ge \gamma_{2t-1}$, we can assume, without loss of generality, that $\gamma_i = \gamma_{2t-1}$ and $l = t$, $t - 1 \le r \le t$.

From our ordering of the functions, it must be the case that F_i is balanced. If the other point u_i' was to the left of u_i, then the same would be true of the root x_i of F_i. Then, the cut at u_i is at least $\gamma_{2t-1} + t = \delta$: a cut of $\gamma_{2t-1} + l - 1 = \gamma_{2t-1} + t - 1$ that we already counted, and an additional cut of 1 from the interval (x_i, v). Therefore u_i' lies to the right of v, and consequently to the right of u_j. Since F_i has nodes both to the left of u_i and the right of u_j, $l + r \ge 2t$, and $r \ge t$. Thus, $r = t$, $\gamma_j = \gamma_{2t-1}$ and F_j is also balanced. If the other point u_j' of F_j lies to the right of u_j, then we are led again to a contradiction. If u_j' lies to the left of u_j (and u_i), then $l + r \ge 2t + 1$, which is impossible. Thus, Case 1 cannot happen.

Case 2. The two points closest to the root belong to the same subfunction F_i.

Then $l + r \ge 2t$. Since $\gamma \ge \gamma_i + l - 1$, $\gamma \ge \gamma_i + r - 1$, it follows that $l = r = t$, $\gamma_i = \gamma_{2t-1}$ and F_{opt} is balanced.

It remains to show that the maxcut does not occur in the disjoint combination over two or more subfunctions. Suppose first that it occurs over two subfunctions F_{2s-1}, F_{2t-1} $(s < t)$ on the odd side; since $\delta \ne \gamma$, the subfunctions are balanced and $\delta = \gamma_{2s-1} + s = \gamma_{2t-1} + t$. From our previous analysis, if we restrict attention in the optimal combination F_{opt} to the first $2t - 1$ functions, there must be exactly one function F_i with maxcut points on both sides of v, and $\gamma_i = \gamma_{2t-1}$ (Case 2 of the previous analysis). But then, among the first $2s - 1$ subfunctions there is no subfunction with maxcut points on both sides of v (since $\gamma_{2s-1} > \gamma_{2t-1}$), contradiction.

Suppose now that the maxcut in the disjoint combination occurs over an even function F_{2t}; then it also occurs over F_{2t-1}. From our analysis for the first $2t - 1$ functions, there is exactly one function F_i among the first $2t - 1$ with maxcut points u_i, u_i' on both sides of the root v, $l = r = t$ and $\gamma_i = \gamma_{2t-1}$. If F_{2t} has in the optimal combination nodes outside the interval (u_i, u_i'), then $l + r \ge 2t + 1$ and $\gamma = \delta$. Thus, F_{2t} lies entirely in this interval and the maxcut point of F_{2t} has a cut of at least $\gamma_{2t} + t = \delta$. □

LEMMA 4. *Suppose that the disjoint combination is balanced. If δ occurs at the root v, or occurs on the even side over more than one subfunction or on the outside of a subfunction, then the optimal combination is balanced and $\gamma = \delta$.*

PROOF. If δ occurs on the outside of F_{2t}, then it occurs also on the outside of F_{2t-1} and $\delta = \gamma_{2t} + t - 1 = \gamma_{2t-1} + t - 1$. The proof is similar to that of Lemma 2. We take the optimal combination and choose maxcut points for the first $2t$ subfunctions. If u_i, u_j are the two maxcut points closest to v, we have $l + r \geq 2t$, $\gamma \geq \gamma_i + l - 1$, $\gamma_j + r - 1$. Therefore, $l = r = t$ and the cut of the optimal combination at u_i and u_j is at least $\gamma = \delta$.

Suppose now that δ occurs on the even side over the balanced subfunction F_{2t}; $\delta = \gamma_{2t} + t$. Consider the optimal combination. Choose maxcut points for the first $2t$ subfunctions as before. The proof is similar to that of Lemma 3.

Case 1. The two points closest to the root belong to different subfunctions.

Let u_i, u_j be these points. Let l, r be defined as before to be the number of subfunctions with nodes to the left of u_i inclusive, and the number of subfunctions with nodes to the right of u_j inclusive. We have $\gamma \geq \gamma_i + l - 1$, $\gamma \geq \gamma_j + r - 1$, and $l + r \geq 2t$. First, suppose that $l \geq t + 1$. Since $\gamma_i \geq \gamma_{2t}$ and $\gamma = \gamma_{2t} + t$, we have $\gamma_i = \gamma_{2t}$, $l = t + 1$, $r \geq t - 1$. From the ordering of the functions, F_i is balanced. If the other point u_i' of F_i occurred to the left of u_i, then the cut at u_i would be at least $\gamma_{2t} + t + 1$: a cut of $\gamma_{2t} + l - 1 = \gamma_{2t} + t$ that we already counted, and 1 from the interval (x_i, v). Thus, u_i' lies to the right of u_i and consequently of u_j. Therefore, $l + r \geq 2t + 1$ and $r \geq t$. For F_{opt} not to be balanced, we must have $r = t$, $\gamma_j = \gamma_{2t}$. Therefore, F_j is balanced. Since $l + r = 2t + 1$ and $j \neq i$, the other point u_j' of F_j must lie to the right of u_j. Thus, u_j is in the interval (v, x_j), and the cut at u_j is at least $\gamma_{2t} + t = \gamma$.

Suppose now that $l \leq t$, $r \leq t$. Then $l = r = t$. Either $\gamma_i > \gamma_{2t}$, in which case there is a cut of at least $\gamma_{2t} + t = \delta$ at u_i, or $\gamma_i = \gamma_{2t}$, in which case the other point u_i' of F_i is to the left of u_i and again the cut at u_i is δ. Similarly with u_j. We conclude therefore that if the optimal combination is not balanced, then Case 1 is impossible. Note that, if in the disjoint combination, δ occurs at the root, then $\delta = h(v)$, and therefore, no subfunction can have nodes on both side of v in F_{opt}. Thus, if in DC the cutwidth occurs at the root, then we must have Case 1, and F_{opt} is balanced.

Case 2. The two closest points u_i, u_i' belong to the same subfunction F_i.

Then $l + r \geq 2t + 1$. Since $\gamma \geq \gamma_i + l - 1$, $\gamma_i + r - 1$, it follows that for F_{opt} not to be balanced we must have $\gamma_i = \gamma_{2t}$, $i = t + 1$, $r = t$ (or vice versa).

The proof now that the optimal combination is balanced if δ occurs more than once on the even side follows as in Lemma 3. □

To summarize: (1) We know that the disjoint combination achieves the cutwidth unless the maximum occurs exactly over one subfunction on the inside, in which case the cutwidth could be one less and balanced. (2) If the disjoint combination is balanced, we know that the optimal combination has the same maxcut; furthermore, the optimal combination will have to be balanced unless the maxcut occurs in the disjoint combination exactly over one even subfunction on the inside.

5. *The Cost Function*

The information (cutwidth, balance bit) that is propagated in the disjoint strategy is not sufficient to solve the one level problem, and thus to compute the cutwidth of a tree. So we define a more involved cost function on layouts and cut-functions.

At the end of this section, we will see that essentially all the information contained in this cost function is necessary for a dynamic programming approach to work, if the tree is rooted arbitrarily, Let F be a (rooted) cut-function with root v. From F we compute a (finite) sequence $c(F) = \langle \gamma_1, \eta_1, \dots \rangle$ as follows. The first entry γ_1 is the maximum cut of F. If F is not balanced, then $c(F) = \langle \gamma_1 \rangle$. (Note that this includes also the case that the maximum cut occurs only at the root, or at the root and on one side but not the other side.) Suppose that γ_1 occurs on both sides. Let p_1, p_1' be two points closest to the root v on each side where the cut of γ_1 occurs. If p_1 and p_1' are right next to the root (i.e., the cut immediately to the left and right of v is γ_1), then $c(F) = \langle \gamma_1, \gamma_1 \rangle$. Otherwise, let η_1 be the smallest cut at a gap (i.e., at a point other than a node) between p_1 and p_1'; $\eta_1 < \gamma_1$. If η_1 occurs only on one side of v in the interval $[p_1, p_1']$, or if the cut at the root is γ_1, then $c(F) = \langle \gamma_1, \eta_1 \rangle$. Otherwise, let q_1, q_1' be the two points (other than nodes) closest to v in the interval $[p_1, p_1']$ where a cut of η_1 occurs. Let γ_2 be the maximum cut between q_1 and q_1'. (If q_1 and q_1' are right next to the root v, then $\gamma_2 = \max\{\eta_1, F(v)\}$.) If $\gamma_2 = \eta_1$, or γ_2 occurs only on one side of v in the interval $[q_1, q_1']$, then $c(F) = \langle \gamma_1, \eta_1, \gamma_2 \rangle$. Otherwise, we continue similarly by taking the two points closest to v where γ_2 occurs. Or to put it recursively, we let F' be the restriction of F to the interval $[q_1, q_1']$. If the cutwidth of F' is η_1, then $c(F) = \langle \gamma_1, \eta_1, \eta_1 \rangle$. (Note that, since q_1 and q_1' were taken as close to the root as possible, $\gamma(F') = \eta_1$ implies that q_1, q_1' are right next to the root and $c(F') = \langle \eta_1, \eta_1 \rangle$.) Otherwise ($\gamma(F') > \eta_1$), the cost of F is the sequence formed by appending the cost $c(F')$ of F' to $\langle \gamma_1, \eta_1 \rangle$; we denote this sequence as $\langle \gamma_1, \eta_1, c(F') \rangle$.

If L is a layout, the cost $c(L)$ of L is the cost of the cut-function of L. We want to stress the fact that in determining the even entries (the η_i's) of the cost, we look only at the gaps of the function and not at the nodes. Thus, if the value of the cut-function at a node is smaller than the value in (at least) one of the gaps bordering the node, then the node does not play any role in the cost sequence.

Example. If F is a degenerate cut-function, then $c(F) = \langle 0, 0 \rangle$. For any other cut-function, all entries are positive. For the disjoint layout L_a of Figure 4a with root 1 and the heights of all nodes 0, we have $c(L_a) = \langle 3 \rangle$. For the optimal layout L_b with root 1 in Figure 4b, $c(L_b) = \langle 2, 2 \rangle$.

Consider the layout in Figure 5 with root v. If all nodes have height 0, then the cost is $\langle 4, 1, 3, 2, 2 \rangle$. If node v has height 3 and all other nodes have height 0, then the cost is $\langle 4, 1, 3, 2 \rangle$. □

The cost of a cut-function F, $c(F)$, is a sequence of numbers $\langle \gamma_1, \eta_1, \gamma_2, \eta_2, \dots \rangle$, strictly decreasing in the odd entries (the γ's), strictly increasing in the even entries, and such that all the odd entries are at least as large as the even entries. Furthermore, all entries are positive unless the sequence is $\langle 0, 0 \rangle$. Any sequence that satisfies these properties is called a *legal* cost sequence. The example in Figure 5 can be easily generalized to show that for any legal cost sequence c there is a layout L of a rooted tree (with all node heights 0) such that $c(L) = c$. From the way we described how the sequence can terminate, it follows that only the last two entries can be equal. In such a case we say that F (and $c(F)$) is *completely balanced.*

We denote the ith entry of a cost sequence c by $\gamma_t(c)$ or $\eta_t(c)$ if i is $2t - 1$ (odd) or $2t$ (even), respectively. Let F be a cut-function with n nodes and cost sequence c. If F is degenerate, then $n = 1$ and c has length $2 = n + 1$. In all other cases the length of c is at most n: The n nodes of F partition the support into $n - 1$ gaps. Every even order entry $\eta_i(c)$ of c, except possibly the last entry of c, is the cut of F

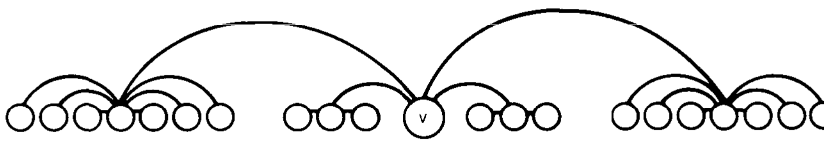


FIGURE 5

at two such gaps, one on each side of the root; all these gaps must be distinct since the η_i's are different. Therefore, if the length of c is $2t + 1$, then there are at least $2t$ gaps, and $|c| \leq n$. If c has length $2t + 2$, then there are at least $2t$ gaps from the first t even order entries and one gap from the last entry. Thus, again $|c| \leq n$. We can also give an upper bound on the length of c in terms of the cutwidth $\gamma = \gamma_1(c)$ of F. Since all the entries of c are at most γ and only the last two entries can be equal, $|c| \leq \gamma + 2$.

We refine the choice of heavy and light sides of a cut-function F from the previous sections as follows. The designation depends on how the cost sequence of F terminates. If F is completely balanced, or its cost sequence ends with a γ_i, which occurs only at the root (in the interval under consideration at that moment in the construction of $c(F)$), then we choose arbitrarily heavy and light sides. If the sequence ends with a γ_i that occurs on one but not the other side, the heavy side is the one in which γ_i occurs and the light side is the other one. If the sequence ends with an η_i, the light side is the one that contains the cut of η_i.

If c is a cost sequence we can add or subtract a positive number k componentwise. If $c = \langle \gamma_1, \eta_1, \gamma_2, \dots \rangle$, then $c \pm k = \langle \gamma_1 \pm k, \eta_1 \pm k, \dots \rangle$. In the case of subtraction the restriction is that the resulting sequence be a legal cost sequence; that is, either $k < \eta_1$ or $k = \eta_1 = \gamma_1$, and $c = \langle k, k \rangle$. These operations on costs reflect corresponding operations on cut-functions. Let F be a nondegenerate cut-function. Suppose that H is another cut-function with the same root and support as F, and such that $H(p) = F(p) + k$ for all p in the support of F, except possibly for a finite number of points p that are not nodes of F and at which $H(p) < F(p) + k$. Clearly, if $H(p) < F(p) + k$, p is a node of H that has cut smaller than the cut in an adjacent gap of H. Since such nodes do not play any role in the cost, we can disregard them, and treat H as if it exceeds F by a cut of k in all of the support. It follows from the definition that $c(H) = c(F) + k$. Suppose now that F has cutwidth at least $k + 1$, and that, furthermore, F has value at least $k + 1$ at all points in its support, except possibly at some nodes. Let G be the function with the same root as F and value $G(p) = F(p) - k$ at all points p in the support of F, and 0 everywhere else; we say that G is obtained from F by subtracting a cut of k. Clearly, G is a nondegenerate cut-function with the same support as F. Since the cut of F exceeds the cut of G by k in all of the support, the cost of G is $c(F) - k$. We define also the subtraction of a cost sequence $c = \langle \gamma_1, \eta_1, \gamma_2, \dots \rangle$ of length at least 2 from a positive number $k \geq \gamma_1$ as follows. If $\gamma_1 > \eta_1$, then $k - c = \langle k - \eta_1, k - \gamma_2, \dots \rangle$; if $\gamma_1 = \eta_1$, then $k - c = \langle k - \eta_1, k - \eta_1 \rangle$. Clearly, $k - c$ is a legal cost sequence when the operation is defined.

We compare legal cost sequences as follows. Let α, β be two such sequences, $\alpha \neq \beta$. If neither is an initial subsequence (prefix) of the other, then $\alpha < \beta$ iff α is lexicographically smaller than β. If α is a prefix of β, then $\alpha < \beta$ or $\alpha > \beta$ provided that α is of odd or even length. Note that if α is a prefix of β, α cannot be completely balanced since only the last two entries can be equal. It follows immediately from the definitions that, if two cut-functions F, G have the same root and $F(p) \geq G(p)$ for all $p \in R$, then $c(F) \geq c(G)$.

LEMMA 5. *$<$ is transitive.*

PROOF. From a cost sequence α define another sequence $\bar{\alpha}$ as follows. If α is of odd (respectively, even) length, $\bar{\alpha}$ is obtained from α by adding $-\infty$ (respectively $+\infty$) at the end. It is easy to see that $\alpha < \beta$ iff $\bar{\alpha}$ is lexicographically smaller than $\bar{\beta}$. Since the lexicographic ordering is transitive, so is $<$. $\square$

Thus, $c(F)$ is a valid cost function. The *cost* of a rooted tree T, $c(T)$, is the minimum cost of a layout of T. We are going to compute a layout $L_{op}(T)$ of minimum cost for a rooted tree T. If T' is a rooted subtree of T, the optimal layout of T constructed by the algorithm, when restricted to the nodes of T' will be identical to $L_{op}(T')$ (although the nodes of T' may not be consecutive). This is (partly) justified by the following crucial fact. Let T be a tree that consists of two rooted trees T_1, T_2, rooted respectively at nodes x_1, x_2 and the edge (x_1, x_2) (see Figure 6). The cutwidth of T depends only on the costs c_1, c_2 of T_1, T_2. Furthermore, the cutwidth of T is a monotonic function of both c_1 and c_2; that is, if we replace T_1 by another tree T_1' with cost $c(T_1') \geq c_1$ and replace T_2 by another tree T_2' with $c(T_2') \geq c_2$, then the cutwidth of the resulting tree T' is at least as large as the cutwidth of T. We prove this fact in the context of cut-funtions.

Let F_1, F_2 be two rooted cut-functions with roots x_1, x_2. (Think of F_1 and F_2 as the cut-functions of layouts for the two subtrees T_1 and T_2 in Figure 6.) A *join* of F_1 and F_2 is the cut-function obtained by choosing a proper ordering of the nodes of F_1, F_2, and then superimposing (summing) the functions F_1, F_2 and the "edge" (x_1, x_2). More formally, a cut-function F is a join of F_1 and F_2 if there are mappings π_1, π_2 from R to R satisfying the following conditions:

(1) The mappings π_1 and π_2 are 1–1, onto and monotone.
(2) They map the nodes of the two functions into distinct points.
(3) If a point p is in the open interval between $\pi_1(x_1)$ and $\pi_2(x_2)$, then

$$F(p) = 1 + F_1(\pi_1^{-1}(p)) + F_2(\pi_2^{-1}(p));$$

otherwise

$$F(p) = F_1(\pi_1^{-1}(p)) + F_2(\pi_2^{-1}(p)).$$

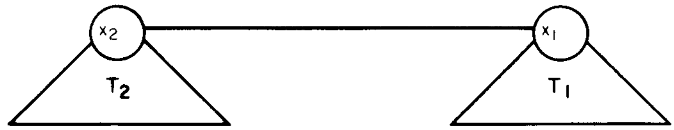
As usual, we drop the π_i's, and identify the subfunctions F_i with their transformed versions within the join F. We are interested only in the cutwidth of F; for this purpose it is not necessary to distinguish a root for F. Suppose without loss of generality that $c(F_1) \geq c(F_2)$. If we place F_1, F_2 (i.e., their nodes) in disjoint intervals and facing each other with their light sides, we get a join F with cutwidth at most $\gamma(F_1) + 1$. (Figure 6 suggests such an arrangement.) On the other hand, clearly, any join has cutwidth at least $\gamma(F_1)$. The following lemma tells us which of the two is the minimum cutwidth of a join.

LEMMA 6. *Let F_1, F_2 be two (rooted) cut-funtions with cost sequences $c_1 = c(F_1)$, $c_2 = c(F_2)$, where $c_1 \geq c_2$. There is a join F of F_1, F_2 with cutwidth $\gamma(F) = \gamma(F_1)$ if and only if either F_1 is not balanced, or F_1 is balanced and $c_2 < \gamma(F_1) - c_1$, or $c_2 = \gamma(F_1) - c_1$ and c_1, c_2 are not completely balanced.*

PROOF. We prove the lemma by induction on $\gamma(F_1) + \gamma(F_2)$. The basis $(\gamma(F_1) + \gamma(F_2) = 0)$ is trivial. For the induction step, we prove first the (if) part.

(*if*) Suppose that c_1, c_2 satisfy the conditions of the lemma. We construct a join F with cutwidth $\gamma(F_1)$. The two subfunctions are oriented in F so that their heavy sides are disjoint (i.e., they face each other with their light sides). For

FIGURE 6



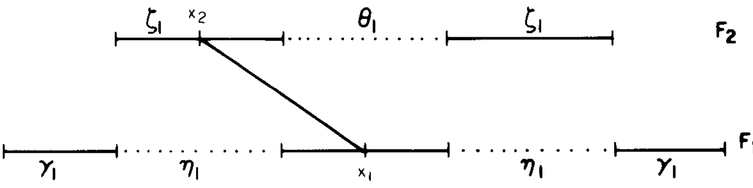


FIGURE 7

example, if we assume, without loss of generality, that the left side of F_1 is the light one and the right side of F_2 is the light one, the join F that we construct will preserve the order of the nodes (the orientation) of F_1 and F_2, and will place x_1 to the right of x_2 (as in Figures 6 and 7); the root of each subfunction lies on the light side of the other subfunction.

If F_1 is not balanced, then either $\gamma(F_2) < \gamma(F_1)$ or F_2 is not balanced (since $c_1 \geq c_2$). The obvious join F where F_1, F_2 do not overlap and face each other with their light sides (see Figure 6) has $\gamma(F) \leq \gamma(F_1)$.

Suppose F_1 is balanced and $c_2 \leq \gamma(F_1) - c_1 = \bar{c}_1$ with c_1 not completely balanced in case of equality. Let $c_1 = \langle \gamma_1, \eta_1, \ldots \rangle$, $c_2 = \langle \zeta_1, \theta_1, \ldots \rangle$. If $\gamma_1 = \eta_1$, then $\bar{c}_1 = \gamma_1 - c_1 = \langle 0, 0 \rangle$ is completely balanced and $c_2 \geq \bar{c}_1$. Therefore, $\gamma_1 > \eta_1$. Let $\bar{c}_1 = \gamma_1 - c_1 = \langle \gamma_1 - \eta_1, \gamma_1 - \gamma_2, \ldots \rangle = \langle \bar{\gamma}_1, \bar{\eta}_1, \ldots \rangle$. We have $\zeta_1 < \bar{\gamma}_1 = \gamma_1 - \eta_1$ or $\zeta_1 = \bar{\gamma}_1 = \gamma_1 - \eta_1$. In the former case insert F_2 in a gap of F_1 where the mincut η_1 occurs; by this we mean that F orders the nodes as follows (assuming F_1, F_2 are oriented as above): first come the nodes of F_1 to the left of the point with the mincut η_1, then all the nodes of F_2, and finally the nodes of F_1 to the right of the mincut point. Note, that the reason that we insisted in the definition of a cost sequence for η_1 to occur at a point other than a node (a gap), was in order to be able to perform such insertions. This insertion does not create a cut of more than $\zeta_1 + \eta_1 + 1 \leq \gamma_1$. So assume $\zeta_1 = \bar{\gamma}_1 = \gamma_1 - \eta_1$. If ζ_1 occurs only on one side of F_2, then we can again insert F_2 in a gap of F_1 where η_1 occurs, with the light side of F_2 facing x_1. So, suppose ζ_1 occurs on both sides, and $c_2 = \langle \zeta_1, \theta_1, \ldots \rangle$ (has length ≥ 2). Since $c_2 \leq \bar{c}_1$, $\bar{c}_1$ has also length ≥ 2, $\bar{c}_1 = \langle \bar{\gamma}_1, \bar{\eta}_1, \ldots \rangle$. This means in particular that η_1 occurs on both sides of F_1. Either $\theta_1 < \bar{\eta}_1$ or $\theta_1 = \bar{\eta}_1$.

If $\theta_1 < \bar{\eta}_1$, then we form a join F by placing the part of F_2 that includes x_2 up to the point where θ_1 occurs, in a gap of F_1 where η_1 occurs; the rest of F_2 goes on the other side of x_1 where η_1 occurs again (see Figure 7). The maxcut of F_1 between the two η_1 points is γ_2 and $\theta_1 + \gamma_2 + 1 \leq \bar{\eta}_1 + \gamma_2 = \gamma_1$. Also, since $\theta_1 < \bar{\eta}_1 \leq \bar{\gamma}_1 = \zeta_1$, the point of F_2 where θ_1 occurs is well-defined ($\theta_1 < \zeta_1$) and the maxcut of F_2 between this point and x_2 is $\leq \zeta_1 - 1$.

Suppose $\theta_1 = \bar{\eta}_1$. If $\bar{\eta}_1 = \bar{\gamma}_1$ (i.e., $\eta_1 = \gamma_2$), then also $\theta_1 = \zeta_1$ and $\bar{c}_1 = \langle \bar{\gamma}_1, \bar{\eta}_1 \rangle = \langle \zeta_1, \theta_1 \rangle = c_2$ and c_1, c_2 are completely balanced. Therefore, $\eta_1 < \gamma_2$.

Write c_1 as $\langle \gamma_1, \eta_1, \hat{c} \rangle$; since $\eta_1 < \gamma_2$, $\hat{c}_1 = \langle \gamma_2, \ldots \rangle$ is the cost of the restriction of F_1 to the interval between the points closest to x_1 where η_1 occurs. These two points are right next to two nodes a and b of F_1 (it is possible that $a = b = x_1$).

Since the maximum cut in $[a, b]$ is $\gamma_2 > \eta_1$, it follows that $\hat{c}$ is also the cost of the restriction $\hat{F}_1$ of F_1 to the interval $[a, b]$. The minimum cut of $\hat{F}_1$ (except possibly at some nodes) is at least $\eta_1 + 1$. Let F_1' be the cut-function obtained from $\hat{F}_1$ by subtracting a cut of η_1; F_1' has the same nodes as $\hat{F}_1$, and has cost $c_1' = \hat{c}_1 - \eta_1 = \langle \gamma_2 - \eta_1, \ldots \rangle$. Now we have a smaller instance of our problem with F_2 (cost c_2) and F_1' (cost c_1') in place of F_1 (cost c_1) and F_2 (cost c_2). Since $\zeta_1 = \gamma_1 - \eta_1 > \gamma_2 - \eta_1$, we have $c_2 > c_1'$, and c_2 is balanced. From the inductive hypothesis, F_2 and F_1' can be joined into a function F' with cutwidth $\le \zeta_1$ if $c_1' < \zeta_1 - c_2$ or $c_1' = \zeta_1 - c_2$ and they are not completely balanced. Once we have F', we can insert it between the two points of F_1 where η_1 occurs; that is, if we assume without loss of generality that F' preserves the order of the nodes of F_1' (i.e., the nodes of F_1 in the interval $[a, b]$), then our join F orders the nodes as follows: first come the nodes of F_1 that are (strictly) to the left of a, then the nodes of F', and finally the nodes of F_1 to the right of b. Clearly, this is a proper ordering of the nodes. This insertion does not create a cut of more than $\zeta_1 + \eta_1 = \gamma_1$: the cut of F at any point in the support of F' is η_1 more than the cut (at most ζ_1) of F'. Now, $c_1' \le \zeta_1 - c_2$ iff $\hat{c}_1 = \eta_1 + c_1' \le \eta_1 + \zeta_1 - c_2 = \gamma_1 - c_2$, that is, iff $\langle \gamma_2, \eta_2, \ldots \rangle \le \langle \gamma_1 - \theta_1, \gamma_1 - \zeta_2, \ldots \rangle$. From the definition of $<$, and since $\bar{\gamma}_1 = \gamma_1 - \eta_1 = \zeta_1$, the last inequality is equivalent to $\bar{c}_1 = \gamma_1 - c_1 = \langle \gamma_1 - \eta_1, \gamma_1 - \gamma_2, \gamma_1 - \eta_2, \ldots \rangle \ge \langle \zeta_1, \theta_1, \zeta_2, \ldots \rangle = c_2$. In case of equality, c_1, c_2 are completely balanced iff the same holds for c_1', c_2.

(*only if*) Suppose now that there is a join F of F_1, F_2 with $\gamma(F) \le \gamma(F_1)$ with F_1 balanced. Let $c_1 = \langle \gamma_1, \eta_1, \ldots \rangle$, $c_2 = \langle \zeta_1, \theta_1, \ldots \rangle$. Suppose that $c_2 > \gamma_1 - c_1 = \bar{c}_1$ or $c_2 = \bar{c}_1$ but c_1, c_2 are completely balanced. Since $\gamma(F) \le \gamma_1$, F_2 must lie between the points of F_1 closest to x_1 where the cut γ_1 of F_1 occurs. Since the mincut of F_1 in this interval is η_1, we must have $\zeta_1 + \eta_1 \le \gamma_1$; i.e., $\zeta_1 \le \bar{\gamma}_1 = \gamma_1 - \eta_1$. Since $c_2 \ge \bar{c}_1$, equality must hold, $\zeta_1 = \gamma_1 - \eta_1$, and ζ_1 must occur on both sides of F_2. If $\eta_1 = \gamma_1$, then because of the "edge" (x_1, x_2) a cut of at least $\eta_1 + 1 = \gamma_1 + 1$ will be created in F. Therefore, $\eta_1 < \gamma_1$.

Let p, q be the points closest to x_2 on the two sides of F_2 where a cut ζ_1 occurs. If one of them, say p, was in the interval (x_1, x_2), then a cut of $\ge \zeta_1 + 1 + \eta_1 = \gamma_1 + 1$ would occur at p (ζ_1 from F_2, η_1 from F_1, and 1 from the "edge" (x_1, x_2)). Therefore, p and q are outside the interval (x_1, x_2). Since they lie on different sides of x_2, they lie also on different sides of x_1. If the cut of F_1 at its root x_1 was γ_1, then the cut of F at x_1 would be at least $\gamma_1 + 1$ because x_1 is in the support of F_2. Therefore, the cut of F_1 at x_1 is less than γ_1. The cut of F at p and q must be η_1 because $\zeta_1 + \eta_1 = \gamma_1$. Thus, F_1 has a mincut of η_1 on both sides, and $c_1 = \langle \gamma_1, \eta_1, \gamma_2, \ldots \rangle$, with $\theta_1 \ge \gamma_1 - \gamma_2$. If $\eta_1 = \gamma_2$, then in the interval (x_1, x_2) we have a cut of at least θ_1 (from F_2) $+ \eta_1$ (from F_1) $+ 1$ (the "edge" (x_1, x_2)); that is, a cut $\ge \theta_1 + \gamma_2 + 1 \ge \gamma_1 + 1$. Thus, $\eta_1 < \gamma_2$.

Let $\hat{F}_1$ and F_1' be defined as in the (if) part. Since $\eta_1 < \gamma_2$, the cost of $\hat{F}_1$ is again $\hat{c}_1 = \langle \gamma_2, \ldots \rangle$, the suffix of c_1 from the third entry on, and the cost of F_1' is $c_1' = \hat{c}_1 - \eta_1$. All nodes of F_2 and F_1' lie between the two points (closest to x_1) where the cut γ_1 of F_1 occurs. Thus, F contains between these two points a join, say F', of F_2 and F_1'. The maximum cut of F in this interval is at least η_1 more than the cut of F'. Since the cutwidth of F is at most γ_1, the cutwidth of F' is at most $\gamma_1 - \eta_1 = \zeta_1$. Since $\zeta_1 = \gamma_1 - \eta_1 > \gamma_2 - \eta_1$, we have again $c_2 > c_1'$, and c_2 is balanced. From the inductive hypothesis, the existence of F' implies that $c_1' < \zeta_1 - c_2$, or $c_1' = \zeta_1 - c_2$ and not completely balanced. As in the (if) part, this condition is equivalent to the condition $c_2 < \bar{c}_1$ or $c_2 = \bar{c}_1$ and not completely balanced, which we assumed did not hold. □

Note that the condition of Lemma 6 is monotonic in the costs of both F_1 and F_2. This implies in particular that there is a layout of minimum cutwidth for the tree T of Figure 6 such that its restrictions on T_1 and T_2 have minimum costs.

Suppose now that we want to compute the cutwidth of a tree T using the dynamic programming approach where we root T arbitrarily at some node. Let us denote by $I(x)$ the information that is computed for the subtree T_x rooted at a node x. For every node x, the (undirected) tree T can be regarded as being composed of two rooted trees (as in Figure 6): T_x rooted at x and the rest of the tree (call it $\bar{T}_x$) rooted at the father of x. When a dynamic programming algorithm computes $I(x)$, it does not know anything about $\bar{T}_x$. Therefore, the cutwidth of T must be a function of $I(x)$ and $\bar{T}_x$ where $\bar{T}_x$ can be arbitrary. As we mentioned earlier, for every legal cost sequence c, there is a layout of some tree whose cost is c. The tree of Figure 5 can be generalized to show that for any legal c there is a tree whose minimum cost is c. Since $\bar{T}_x$ is arbitrary, for any legal cost c, the algorithm must be able to tell from $I(x)$ and the cost c of $\bar{T}_x$ what the cutwidth of T is. From Lemma 6 it follows that, for any legal balanced c with $c \geq c(T_x)$, the algorithm can tell from $I(x)$ and c if $c(T_x) \leq \gamma_1(c) - c$ (or $c(T_x) < \gamma_1(c) - c$ in case c is completely balanced). But $\gamma_1(c) - c$ can be any legal cost sequence. Therefore, $c(T_x)$ can be derived from $I(x)$.[1] Thus, all the parameters of the cost function are necessary for the dynamic programming approach to work.

Suppose that we root the tree T of Figure 6 at x_1 or x_2. It is easy to see that although the costs of T_1 and T_2 contain enough information to determine the cutwidth of T, they are not sufficient to compute the whole cost sequence of T. However, as we shall show in the next section, the cost of a rooted tree can be determined from the costs of the subtrees rooted at the root's children; and this is all that is needed for the algorithm to work.

6. *The Algorithm*

We root the tree arbitrarily at any node, and proceed bottom-up computing at every node the cost of the subtree rooted at the node. We have to show how to solve the one-level problem. In the one-level problem we have a tree T rooted at node v with height $h(v)$ and sons $x_1, \ldots, x_d$. We are given the costs $c_1, \ldots, c_d$ of the subtrees $T_1, \ldots, T_d$ rooted at the x_i's, and layouts $L_1, \ldots, L_d$ realizing these costs, and we want to compute the cost $c(T)$ of T and a layout L realizing it.

We present an algorithm OPT(h; $F_1, \ldots, F_d$), which takes as input the height h of the root v and arbitrary (rooted) cut-functions $F_1, \ldots, F_d$ with costs $c_1, \ldots, c_d$, and produces an optimal (minimum cost) combination F_{op} of v and the F_i's. If $S = (h; F_1, \ldots, F_d)$ is an input to OPT, we denote the cost of the optimal combination by $c(S)$, or simply by c_{op} when S is understood. If $S' = (h'; F'_1, \ldots, F'_d)$ is another input with $h' \leq h$ and $c(F'_i) \leq c(F_i)$ for all i, we write $S' \leq S$. The cost $c(S)$ of the optimal combination is a monotonic function of the height of the root and the costs of the subfunctions; that is, $S' \leq S$ implies $c(S') \leq c(S)$. From this monoton-

[1] More accurately: From $I(x)$ we can deduce the cost of T_x up to one bit of information; a bit that indicates whether the cost is completely balanced or not. Let us call the smallest cost sequence greater than a given sequence c, the successor of c: every cost sequence has a well-defined successor. As far as the criterion of Lemma 6 is concerned, if c is completely balanced, there is no need to differentiate between c and its successor; this is so because the criterion demands a strict inequality if the sequences are completely balanced, but only a weak inequality if one of them is not. We could have identified a completely balanced cost sequence with its successor, as long as they had the same cutwidth; however, things would become much more complicated.

icity property, and Proposition 1 of Section 3, it follows that if h is the height of the root v of T and the F_i's are the cut-functions of the optimal layouts L_i for the subtrees, then OPT will return the cut-function of an optimal layout of T.

We also employ three other procedures: ANCH(h; $F_1, \ldots, F_d$), OP1($\cdot$), AN1($\cdot$). Let F be a combination of root v with height h and cut-functions $F_1, \ldots, F_d$. Let δ be the cutwidth of their disjoint combination. Take a point q outside the support of F and define another cut-function $\bar{F}$ as follows: $\bar{F}(q) = \delta$; $\bar{F}(p) = F(p) + 1$ if p is in the open interval (v, q); $\bar{F}(p) = F(p)$, otherwise. Informally, $\bar{F}$ is obtained from F by adding an "edge" (called the *anchor*) from the root v over one side of F with a cut δ at the end. We call $\bar{F}$ an *anchored combination* of S. ANCH(S) computes an anchored combination $\bar{F}$ of S with the minimum cost $c(\bar{F})$; we denote this minimum anchored cost by $\bar{c}(S)$. OP1 performs the function of OPT in the special case that in the disjoint combination the maxcut occurs exactly once—on the inside of the first subfunction F_1. AN1 performs the function of ANCH in the special case that in the disjoint combination the maxcut occurs over exactly one subfunction, the first one F_1; the maxcut may or may not occur at the root.

Before we proceed let us collect into a lemma some observations on the possible values of $\bar{c}(S)$.

LEMMA 7. *The optimal anchored cost $\bar{c}(S)$ is between $\langle \delta \rangle$ and $\langle \delta + 1 \rangle$. In particular*

(1) $\bar{c}(S) = \langle \delta + 1 \rangle$ *iff $c(S)$ is balanced with cutwidth δ.*
(2) $\bar{c}(S)$ *is balanced with cutwidth δ iff $c(S) = \langle \delta \rangle$ and there is no combination of S in which the cutwidth δ occurs only at the root.*
(3) $\bar{c}(S) = \langle \delta \rangle$ *iff either $c(S)$ has cutwidth $\delta - 1$ (and is balanced) or $c(S) = \langle \delta \rangle$ but there is a combination of S in which the cutwidth δ occurs only at the root.*

Furthermore, in cases (2) and (3), if $\bar{F}$ is any optimal anchored combination of S, the combination F satisfies the conditions given on the right-hand side of the equivalence.

PROOF. From our analysis in Section 4, there are 3 cases for $c(S)$. Either (i) $c(S)$ is balanced with cutwidth δ, or (ii) $c(S) = \langle \delta \rangle$, or (iii) $c(S)$ is balanced with cutwidth $\delta - 1$. Divide case (ii) further into subcaces (iia) and (iib) depending on whether the cutwidth δ must occur in any optimal combination F_{op} on one side, or there is an optimal combination F_{op} in which δ occurs only at the root. Let F_{op} be an optimal combination of S; in case (iib) choose an F_{op} in which δ occurs only at the root. Add the anchor on the light side of F_{op}. In case (i) $c(\bar{F}_{\text{op}}) = \langle \delta + 1 \rangle$. In case (iia), $\bar{F}_{\text{op}}$ is balanced with cutwidth δ. In cases (iib) and (iii), $c(\bar{F}_{\text{op}}) = \langle \delta \rangle$.

For the converse, we observe that if $\bar{c}(S) = \langle \delta + 1 \rangle$, then $c(S)$ must be balanced with cutwidth δ, because otherwise we could achieve a lower anchored cost. Let $\bar{F}$ be an optimal anchored combination. If $\bar{F}$ is balanced with cutwidth δ, then the side of F without the anchor must have cut δ, and the side of F where the anchor is placed must have cut at most $\delta - 1$, because otherwise the addition of the anchor would create a cut greater than δ. Thus, $c(F) = \langle \delta \rangle$. If $c(\bar{F}) = \langle \delta \rangle$, then both sides of F must have maximum cut at most $\delta - 1$: the side with the anchor for the same reason as above, and the side without the anchor because $\bar{F}$ is not balanced. (Recall that we have a cut δ at the end of the anchor.) The cut of F at the root is δ or smaller. Finally, $c(\bar{F})$ cannot have cutwidth $\delta - 1$ (or less) because of the cut δ at the end of the anchor. □

We remark that even if we just added the anchor without the cut δ at the end, $c(\bar{F})$ could not have cutwidth $\delta - 1$ (or less) for any combination F: $\gamma(\bar{F}) \le \delta - 1$ would imply $c(F) \le \langle \delta - 1 \rangle$ which is impossible according to the results of Section 4. In other words, the cut δ at the end of the anchor does not play a role in determining the cutwidth of $\bar{c}(S)$. The only case that the cut δ at the end of the anchor plays a role (and the reason we added it) is in Case (2), and then only if it is the sole cause that the optimal $\bar{F}$ is balanced; that is, if the side of $\bar{F}$ with the anchor does not contain another cut δ.

In Cases (1) and (3) it is obvious that we can pick an optimal anchored combination $\bar{F}$ whose heavy side is the side with the anchor. (In fact, in Case (3) this must be the case because of the cut δ at the end of the anchor.) In Case (2) it is not obvious at this point that this is possible. However, we show (inductively) that ANCH and AN1 deliver always an optimal anchored combination whose heavy side is the one with the anchor.

We say that a procedure is monotonic if, for every input S in the domain of the procedure and for any S' (not necessarily in the domain) with $S' \le S$, the optimal cost for S' ($c(S')$ for OPT, OP1, $\bar{c}(S')$ for ANCH, AN1) does not exceed the cost returned by the procedure on input S. We show inductively that all the procedures are monotonic. It is important for the induction that in the definition of monotonicity, S' is not restricted to belong to the domain of the procedure. We present only the computation of the costs; the algorithms can be easily modified to compute also combinations realizing these costs. We describe in each case in detail how such a combination is formed. We present each procedure in turn and prove its correctness and monotonicity on the assumption that the other procedures are correct and monotonic for inputs of smaller size. The size of an input S can be taken to be the sum of d and the cutwidths of the subfunctions. (Other measures would also do.)

OPT(h; $c_1, \ldots, c_d$)
If $d = 0$, then return $\langle 0, 0 \rangle$ if $h \le 0$, and $\langle h \rangle$ if $h > 0$.
Sort the c_i's and assume that $c_1 \ge c_2 \ge \cdots \ge c_d$. Compute the disjoint combination and let δ be its cutwidth.

Case 1. If δ does not occur on the even side, and occurs at the root or on the outside of a subfunction or over more than one subfunction, then return $c(S) = \langle \delta \rangle$.

Case 2. If δ occurs only on the odd side, only once, and on the inside of that subfunction, then let $F_{2t-1} (t \ge 1)$ be that subfunction. Return $c(S) = \mathrm{OP1}(h - (t - 1); c_{2t-1}, c_{2t}, \ldots) + t - 1$.

Case 3. δ occurs on both sides (and possibly at the root).

Let $F_{2t} (t \ge 1)$ be the deepest subfunction (max t) on the even side over which δ occurs. Compute $\alpha = \mathrm{AN1}(h - (t - 1); c_{2t}, c_{2t+1}, \ldots)$.

3a. If $\alpha > \langle \delta - (t - 1) \rangle$ then return $c(S) = \alpha + (t - 1)$.

3b. Else, if δ occurs in the disjoint combination only once on the even side (over F_{2t}), then return $c(S) = \langle \delta \rangle$. Otherwise, let F_{2q} be the deepest subfunction (maximum q) on the even side before F_{2t} where δ occurs; return $\langle \delta, q \rangle$.

Let S be an input consisting of height h for the root and cut-functions $F_1, \ldots, F_d$ with costs $c_1, \ldots, c_d$. Let us first describe the combination F achieving the cost computed by OPT, and explain how the algorithm works. In Case 1, the combination returned is the disjoint combination DC of S. Since δ does not occur on the even side, its cost is $\langle \delta \rangle$. The heavy side is the odd one. In Case 2, let $S' = (h - (t - 1); F_{2t-1}, \ldots, F_d)$. The disjoint combination DC$'$ of S' arranges the subfunctions $F_{2t-1}, \ldots, F_d$ in the same way as DC. The cut of DC$'$ over a subfunction F_i, $i \ge 2t - 1$ or at the root v is $t - 1$ less than the cut of DC.

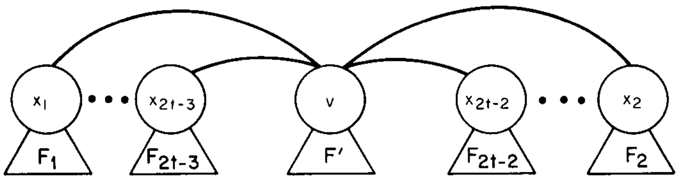


FIGURE 8

Therefore, the cutwidth of DC′ is $\delta - (t - 1)$ and occurs only on the odd side and only on the inside of F_{2t-1}. Thus, S' is in the domain of OP1. Let F' be the combination returned by OP1. Form a combination F of S as follows. The first $2(t - 1)$ subfunctions are arranged as in the disjoint combination; the root v and the rest of the subfunctions are placed between them according to the combination F' oriented so that its light side coincides with the even side (see Figure 8). Clearly, since F' orders properly the nodes of $F_{2t-1}, \ldots, F_d$, it follows that F orders properly the nodes of all the subfunctions. From Corollary 1 and Lemma 3, the optimal combination of S' either has cost $\langle \delta - (t - 1) \rangle$ or is balanced with cutwidth $\delta - t$. From the correctness of OP1, the combination F' that it returns is optimal. At each point within F', the cut of F is $t - 1$ more than the cut of F' because of the "edges" (v, x_i) with $i < 2t - 1$ and the difference in the height of the root; at each point outside F', the cut of F is at most $\delta - 1$. Therefore, the first $2t - 2$ subfunctions are irrelevant as far as the cost of the produced combination F is concerned, and $c(F) = c(F') + (t - 1)$. The heavy side of F is the odd one (the heavy side of F').

In case 3, let $S' = (h - (t - 1); F_{2t}, \ldots, F_d)$. We observe that the disjoint combination DC′ of S' arranges the subfunctions $F_{2t}, \ldots, F_d$ in the same way as DC, except that the even and odd sides are switched. The maximum cut of DC′ over a subfunction F_{2i} is $t - 1$ less than the cut of DC over F_{2i}, and the cut of DC′ over a subfunction F_{2i+1} is t less than the cut of DC over F_{2i+1}. Therefore, the maximum cut of DC′ on its heavy side is $\delta - (t - 1)$ and occurs only over F_{2t}, while the maximum cut on its light side is at most $\delta - t$. Also, since $\delta \geq h$, the cut of DC′ at the root v is $h - (t - 1) \leq \delta - (t - 1)$. Thus, S' is in the domain of AN1. Let $\bar{F}'$ be the combination returned by AN1. From the correctness of AN1, $c(\bar{F}') = \alpha = \bar{c}(S')$. Form a combination F of S as follows: Arrange the first $2t - 1$ subfunctions as in the disjoint combination. The "edge" (v, x_{2t-1}) plays the role of the anchor. The rest of the subfunctions are placed inside them (between F_{2t-2} and F_{2t-1}) according to the combination F' of S' returned by AN1; $\bar{F}'$ is oriented so that the side of the anchor faces x_{2t-1}. The cut of F at any point in the support of F' is $t - 1$ more than the cut of $\bar{F}'$. Since $c(\mathrm{DC}') = \langle \delta - (t - 1) \rangle$, it follows from Lemma 7 that α has cutwidth $\delta - (t - 1)$. Therefore, if $\alpha > \langle \delta - (t - 1) \rangle$, then α is balanced with cutwidth $\delta - (t - 1)$, the first $2t - 1$ subfunctions do not enter in the computation of the cost of F, and $c(F) = c(\bar{F}') + (t - 1) = \alpha + (t - 1)$. In this case, the heavy side of F is the same as the heavy side of $\bar{F}'$. Assuming inductively that the heavy side of $\bar{F}'$ is the one with the anchor, the heavy side of F is the odd one.

If $\alpha \leq \langle \delta - (t - 1) \rangle$, then $\alpha = \langle \delta - (t - 1) \rangle$ (Case 3 of Lemma 7), which means that the maximum cut of F' on each side is at most $\delta - t$. (The cut at the root may be $\delta - (t - 1)$.) Therefore, the cut of F at a point on the even side within F' is at most $\delta - t + (t - 1) = \delta - 1$. If δ does not occur on the even side of DC over any subfunction other than F_{2t}, then F is not balanced and $c(F) = \langle \delta \rangle$. Otherwise, the minimum cut of F on the even side before a cut of δ occurs is q (between F_{2q+2}

and F_{2q}), where q is as defined in Case 3b. Since δ occurs on the odd side over F_{2t-1}, the mincut of F on the odd side (before a cut of δ) is at least $t > q$. Thus, $c(F) = \langle \delta, q \rangle$. The heavy side of F is the odd one.

LEMMA 8. *OPT is correct and monotonic assuming that OP1 and AN1 are correct and monotonic for inputs of equal or smaller size.*

PROOF. We first prove the monotonicity and then the correctness of OPT.

(a) *Monotonicity.* Let c be the cost returned by OPT on input S. Let $\hat{S} = (\hat{h}; \hat{F}_1, \ldots, \hat{F}_d)$ be another input with $\hat{S} \le S$. We form a combination $\hat{F}$ of $\hat{S}$ that has in each case the same structure as the combination F of S computed by OPT; $\hat{F}$ may be a suboptimal combination of $\hat{S}$, but we shall show that its cost (which is $\ge c(\hat{S})$) does not exceed c.

Since $\hat{S} \le S$, after sorting the costs of the $\hat{F}_i$'s, we have $\hat{c}_1 \ge \cdots \ge \hat{c}_d$, with $\hat{c}_i \le c_i$ for all i. Thus $\hat{\gamma}_i \le \gamma_i$, and $\hat{\gamma}_i + \hat{b}_i \le \gamma_i + b_i$ for all i. It follows that the maximum cut of the disjoint combination $\widehat{DC}$ of $\hat{S}$ over a subfunction $\hat{F}_i$ is at most equal to the cut of DC over the corresponding subfunction F_i, and the cutwidth $\hat{\delta}$ of $\widehat{DC}$ is at most δ. If $d = 0$, then clearly $c(\hat{S}) \le c$ because $\hat{h} \le h$. If Case 1 applies for input S, then, either $\hat{\delta} < \delta$, in which case $c(\widehat{DC}) < \langle \delta \rangle$, or $\hat{\delta} = \delta$, in which case $\widehat{DC}$ is not balanced and has cost $\langle \delta \rangle$. If Case 2 applies for S, then we form a combination $\hat{F}$ similar to the combination F of S. That is, we take the optimal combination $\hat{F}'$ of $\hat{S}' = (\hat{h} - (t-1); \hat{F}_{2t-1}, \ldots)$ and insert it between $\hat{F}_{2t-3}$ and $\hat{F}_{2t-2}$ (as in Figure 8). Note that $\hat{S}'$ may not be in the domain of OP1. From the monotonicity of OP1, $c(\hat{F}') \le c(F')$. If $\hat{F}'$ has cutwidth $\delta - (t-1)$, then $c(\hat{F}') = c(F') = \langle \delta - (t-1) \rangle$, and $c(\hat{F}) = c = \langle \delta \rangle$. If $\hat{F}'$ has cutwidth $\delta - t$ and is balanced, then $c(\hat{F}) = c(\hat{F}') + t - 1 \le c$. Otherwise ($c(\hat{F}') \le \langle \delta - t \rangle$), the cutwidth of $\hat{F}$ is at most $\delta - 1$, and the maximum cut of $\hat{F}$ in the support of $\hat{F}'$ on the even side is at most $\delta - 2$. It follows that the cost of $\hat{F}$ is at most $\langle \delta - 1, t - 1 \rangle$, which is strictly smaller than c. This is so because, if c has cutwidth $\delta - 1$, then its mincut $\eta_1(c)$ is equal to the mincut of F' increased by $t - 1$, which is at least t.

Suppose that Case 3 applies for S. Assume first that in the disjoint combination $\widehat{DC}$ of $\hat{S}$ a cut of δ occurs over $\hat{F}_{2t}$. Then, Case 3 of OPT applies also for the input $\hat{S}$, and the algorithm will construct a combination $\hat{F}$ for $\hat{S}$, as we described before. From the monotonicity of AN1, the cost $\hat{\alpha}$ of the best anchored combination of v with height $\hat{h} - (t-1)$ and $\hat{F}_{2t}, \ldots, \hat{F}_d$ is at most α. If $\hat{S}$ falls into Case 3a, then so does S, and $c(\hat{F}) \le c$. If $\hat{S}$ falls into Case 3b, and $c(\hat{F}) = \langle \delta, \hat{q} \rangle$, that is, δ occurs in $\widehat{DC}$ over $\hat{F}_{2\hat{q}}$, then δ occurs also in DC over the corresponding subfunction; so, $\hat{q} \le q$, and $c(\hat{F}) \le c$. If $c(\hat{F}) = \langle \delta \rangle$, then clearly, $c(\hat{F}) \le c$. Assume now that the cut of $\widehat{DC}$ over $\hat{F}_{2t}$ is less than δ. We show that the cost of $\widehat{DC}$ does not exceed c. If $\widehat{DC}$ does not have a cut of δ on the even side, then $c(\widehat{DC}) \le \langle \delta \rangle \le c$. If $\widehat{DC}$ has a cut of δ on the even side, and $\hat{F}_{2\hat{q}}$ is the deepest subfunction where δ occurs, then $\hat{q} < t$ and $\hat{q} \le q$. As before, $c(\widehat{DC}) \le \langle \delta, \hat{q} \rangle \le \langle \delta, q \rangle \le c$.

(b) *Correctness.* Let $S = (h; F_1, \ldots, F_d)$ be an input with $c(F_i) = c_i$ for all i. Let F_{op} be the optimal combination of S. We have to show that $c_{\text{op}} = c(F_{\text{op}}) \ge c$, where c is the cost returned by OPT. The proof is a continuation of the analysis of Section 4.

Case 1. From Lemmas 2 and 3, $\gamma(F_{\text{op}}) = \delta$ and, therefore, $c_{\text{op}} \ge \langle \delta \rangle = c$.

Case 2. The computed cost c is not minimal if either $c = \langle \delta \rangle$ but there is a (balanced) combination F_{op} with $\gamma(F_{\text{op}}) = \delta - 1$, or $c = \langle \delta - 1, \ldots \rangle$, but F_{op} has a better cost. In either case F_{op} must have cutwidth $\delta - 1$. From the analysis in

Lemma 3, if we take the maxcut points in F_{op} of the first $2t - 1$ subfunctions, the two points u_i, u_i' closest to the root v must belong to the same subfunction F_i, with $\gamma_i = \gamma_{2t-1}$, and $l = r = t$. That is, every subfunction beyond the $(2t - 1)$st is contained entirely within the interval (u_i, u_i'). Let F_i' be the restriction of F_i to the interval $[u_i, u_i']$. Since the cut of F_i at u_i and u_i' is γ_i, the cost of F_i' is equal to the cost c_i of F_i. Consider the combination F^* of v with height $h - (t - 1)$ and F_i', F_{2t}, $\ldots$, F_d, which is contained in F_{op}. The support of this combination is $[u_i, u_i']$, and at every point in this interval the cut of F_{op} exceeds the cut of F^* by at least $t - 1$ $(= l - 1 = r - 1)$. Therefore, $c_{op} \geq c(F^*) + (t - 1)$. Since $c_i \geq c_{2t-1}$, it follows from the correctness of OP1 and monotonicity that $c(F^*) \geq \mathrm{OP1}(h - (t - 1); c_{2t-1}, \ldots)$. Therefore, $c_{op} \geq c$.

Case 3. Now the disjoint combination is balanced with maxcut δ. As in the proof of Lemma 4, look at the maxcut points of the first $2t$ subfunctions. If the two points u_i, u_j closest to v belong to different subfunctions, then from the analysis of Lemma 4, the cut at these points is at least δ, and $l, r \geq t$. The mincut between v and u_i is at least l, and the mincut between v and u_j is at least r. Thus, the mincut (second entry) of c_{op} is at least t. If c was computed in Case 3b, that is, $\alpha \leq \langle \delta - (t - 1) \rangle$ and $c = \langle \delta \rangle$ or $c = \langle \delta, q \rangle$ with $q < t$, then we would have $c_{op} > c$, contradicting the optimality of F_{op}. Therefore, $\alpha > \langle \delta - (t - 1) \rangle$. Also we have $c_{op} \geq \langle \delta, t \rangle$ unless $l = r = t$. But the cost of the disjoint combination is at most $\langle \delta, t \rangle$, and (from the correctness of AN1) $c \leq \langle \delta, t \rangle$. Thus, if $c_{op} < c$, we must have $l = r = t$. This implies that the subfunctions F_k with $k > 2t$ lie entirely in the interval (u_i, u_j). The "edge" (v, x_j) and the subfunction F_j contribute to F_{op} a cut of at least 1 in the interval (v, u_j) and a cut $\delta - (t - 1)$ at u_j. Let F_j' be a cut-function with root u_j, cut $\delta - (t - 1)$ at its root, and 0 everywhere else; the cost of F_j' is $\langle \delta - (t - 1) \rangle \geq c_{2t}$. Let F^* be the combination of v with height $h - (t - 1)$ and F_j', F_{2t+1}, $\ldots$, F_d contained in F_{op}. The "edge" (v, x_i) and the subfunction F_i contribute to F_{op} a cut of at least 1 in the interval (u_i, v) and a cut $\delta - (t - 1)$ at u_i; regard this as an anchor over F^*. The support of $\bar{F}^*$ is $[u_i, u_j]$. The cut of F_{op} exceeds the cut of $\bar{F}^*$ in this interval by at least $t - 1 = l - 1 = r - 1$ because of the difference in the height of the root and because of the "edges" (v, x_k) and the subfunctions F_k with $k \leq 2t$, $k \neq i, j$. Therefore, $c_{op} \geq c(\bar{F}^*) + (t - 1)$. From the correctness and monotonicity of AN1, $c(\bar{F}^*) \geq \alpha$, which implies $c_{op} \geq \alpha + (t - 1) = c$.

Suppose now that the two points u_i, u_i' closest to v belong to the same subfunction F_i. Then, $l + r \geq 2t + 1$. The cut of F_{op} at u_i (respectively, u_i') is at least $\gamma_i + l - 1$ (respectively, $\gamma_i + r - 1$). Assume without loss of generality that $l \geq r$; then $l \geq t + 1$. We have $\delta = \gamma_{2t} + b_{2t} + t - 1 \geq \gamma \geq \gamma_i + l - 1 \geq \gamma_i + t$. Since $\gamma_i \geq \gamma_{2t}$ we conclude $\gamma_i = \gamma_{2t}$, $b_{2t} = 1$ and $l = t + 1$. Since $r \leq l$ and $l + r \geq 2t + 1$, it follows that $t \leq r \leq t + 1$. If $r = t + 1$, then both points u_i, u_i' have cut δ, and the mincut between u_i and u_i' is at least $t + 1$; that is, $c_{op} > \langle \delta, t \rangle$. But the disjoint combination has cost $\leq \langle \delta, t \rangle$. Thus, $r = t$ and we conclude that (1) besides F_i no other subfunction among $F_1, \ldots, F_{2t}$ has maxcut points on both sides of v, and (2) each F_k with $k \geq 2t + 1$ is laid out entirely in the interval (u_i, u_i'). Let u_j be the second closest maxcut point (from the first $2t$ subfunctions) to the left of v. The number of subfunctions that have nodes to the left of u_j is at least $l - 1 = t$. There is a cut of at least 1 in the interval (u_j, v) due to the "edge" (v, x_j) and the function F_j. Also, if F_j is balanced, its other maxcut point and its root x_j must be to the left of u_j, and therefore the cut of F_{op} at u_j due to the "edge" (v, x_j) and the function F_j is at least $\gamma_j + b_j \geq \gamma_{2t} + b_{2t} = \delta - (t - 1)$. Let F_i' be the restriction of

F_i to the interval $[u_i, u_i']$; since the cut of F_i at u_i and u_i' is γ_i, the cost of F_i' is c_i. Consider the combination F^* of v with height $h - (t - 1)$ and $F_i', F_{2t+1}, \ldots, F_d$ contained in F_{op}; the support of F^* is $[u_i, u_i']$. Regard the cut of 1 from v to u_j due to F_j and (v, x_j) as an anchor on F^*. The cut of F_{op} in the support $[u_j, u_i']$ of $\bar{F}^*$ exceeds the cut of $\bar{F}^*$ by at least $t - 1$, because of the difference in the height of the root and because of the "edges" (v, x_k) and subfunctions F_k with $k \leq 2t$, $k \neq i, j$. If $\alpha > \langle \delta - (t - 1) \rangle$, then (from the correctness and monotonicity of AN1) also $c(\bar{F}^*) > \langle \delta - (t - 1) \rangle$ and thus $c_{\text{op}} \geq c(\bar{F}^*) + (t - 1) \geq c$.

If $\alpha \leq \langle \delta - (t - 1) \rangle$, then c is either $\langle \delta \rangle$ or $\langle \delta, q \rangle$. Clearly, F_{op} must have cutwidth $\geq \delta$ (Lemma 3). Suppose $c = \langle \delta, q \rangle$; that is, δ occurs in the disjoint combination over F_{2q} $(q < t)$. Thus, $\delta = \gamma_{2q} + b_{2q} + q - 1 = \gamma_i + t$. Since $q < t$, we have $\gamma_{2q} > \gamma_i$ and therefore $2q < i$. From our conclusion (1), no subfunction among the first $2q$ has maxcut points on both sides of v. It follows then that, if we take the $2q$ first maxcut points and let w_i, w_j be the closest ones to v, the cut of F_{op} at w_i and w_j is δ, and the mincut in the interval (w_i, w_j) cannot be less than q. Since the mincut in the interval (u_i, v) is at least $t + 1$, it follows that $\eta_1(c_{\text{op}}) \geq q$ and the mincut q does not occur on both sides of v. Therefore, $c_{\text{op}} \geq \langle \delta, q \rangle \geq c$. □

The algorithm for the anchored case can be thought of as applying OPT to $(h; c_0, c_1, \ldots, c_d)$, where $c_0 = \langle \delta \rangle$ is the cost of a fictitious subfunction.

ANCH$(h; c_1, \ldots, c_d)$

Sort the c_i's and assume $c_1 \geq \cdots \geq c_d$. Computer the disjoint combination and let δ be its cutwidth.

Case 1. If δ occurs only at the root then return $\bar{c}(S) = \langle \delta \rangle$. If δ occurs on the even side on the outside of a subfunction or over more than one subfunction, then return $\bar{c}(S) = \langle \delta + 1 \rangle$.

Case 2. If δ occurs on the even side over only one subfunction on the inside, let F_{2t} be this function. Return $\bar{c}(S) = \text{OP1}(h - t; c_{2t}, c_{2t+1}, \ldots) + t$.

Case 3. δ occurs on the odd but not on the even side.

Let $F_{2t-1} (t \geq 1)$ be the deepest function (max t) on the odd side over which δ occurs. Compute $\alpha = \text{AN1}(h - (t - 1); c_{2t-1}, c_{2t}, \ldots)$.

3a. If $\alpha > \langle \delta - (t - 1) \rangle$, then return $\bar{c}(S) = \alpha + (t - 1)$.

3b. If δ occurs only once on the odd side (over F_{2t-1}), then return $\langle \delta \rangle$. Otherwise, let F_{2q-1} be the deepest function on the odd side before F_{2t-1} where δ occurs; return $\langle \delta, q \rangle$.

LEMMA 9. *ANCH is correct and monotonic assuming that OP*1 *and AN*1 *are correct and monotonic for inputs of equal or smaller size.*

PROOF. Let $S = (h; F_1, \ldots, F_d)$ be an input to ANCH with $c(F_i) = c_i$ for all i. Let F_0 be a new cut-function with cut δ at its root x_0 and 0 everywhere else. Let $\bar{S} = (h; F_0, F_1, \ldots, F_d)$. An anchored combination $\bar{F}$ of S can be viewed also as a combination of $\bar{S}$; thus, $\bar{c}(S) \geq c(\bar{S})$. Conversely, a combination G of $\bar{S}$ contains a combination F of S; if x_0 is outside the support of F, then G can be viewed as an anchored combination of S. The behavior of ANCH(S) is identical with OPT$(\bar{S})$. The disjoint combination $\overline{\text{DC}}$ of $\bar{S}$ is the same as the disjoint combination DC of S, except that F_0 is added as the first subfunction and the even and odd sides are switched. The cutwidth $\bar{\delta}$ of $\overline{\text{DC}}$ is either $\delta + 1$ or δ, depending on whether the cutwidth δ of DC occurs on the even side or not. If Case 1 of ANCH applies to S, then Case 1 of OPT applies to $\bar{S}$, and ANCH returns the disjoint combination $\widehat{\text{DC}}$ whose cost is $\langle \bar{\delta} \rangle$. If Case 2 of ANCH applies to S, then Case 2 of OPT applies to $\bar{S}$, $\bar{\delta} = \delta + 1$, and F_0 is ignored by OPT since the maxcut over it is δ. ANCH returns in Case 2 the same combination as OPT. If Case 3 of ANCH applies to S, then $\bar{\delta} = \delta$ and $\overline{\text{DC}}$ is balanced (because of F_0). Case 3 of OPT applies to $\bar{S}$, F_0 does not enter again in the recursion, and ANCH returns the same combination as OPT. The correctness and monotonicity of ANCH follows from that of OPT, and the fact that in the combination produced by OPT, F_0 will appear outside the rest of the subfunctions. Note that the heavy side of the anchored combination returned by ANCH is the side with the anchor, because the heavy side of OPT($\bar{S}$) is the one containing F_0. □

OP1(h; $c_1, \ldots, c_d$)
Let $c_1 = \langle \gamma_1, \eta_1, \gamma_2, \ldots \rangle$.
If $\eta_1 = \gamma_1$ or $h + \eta_1 > \gamma_1$, then return $\langle \gamma_1 + 1 \rangle$.
Else $\{1 \le \eta_1 < \gamma_1$ and $h + \eta_1 \le \gamma_1\}$.

Case 1. If c_1 is of length ≥ 3, let $c_1' = \langle \gamma_2, \eta_2, \ldots \rangle$ be the suffix of c_1 from the third entry on.

Let c_1^* be $\langle 0, 0 \rangle$ if $\eta_1 = \gamma_2$, and $c_1' - \eta_1$ if $\eta_1 < \gamma_2$. Compute $\alpha = \mathrm{OPT}(h; c_1^*, c_2, \ldots, c_d)$.

1a. If $\alpha + \eta_1 \ge \langle \gamma_1 + 1 \rangle$, then $c(S) := \langle \gamma_1 + 1 \rangle$;
1b. If $\langle \gamma_1 + 1 \rangle > \alpha + \eta_1 > \langle \gamma_1 \rangle$, then $c(S) := \alpha + \eta_1$;
1c. If $\langle \gamma_1 \rangle = \alpha + \eta_1$, then $c(S) := \langle \gamma_1, \eta_1 \rangle$;
1d. Else $\{\langle \gamma_1 \rangle > \alpha + \eta_1\} c(S) := \langle \gamma_1, \eta_1, \alpha + \eta_1 \rangle$.

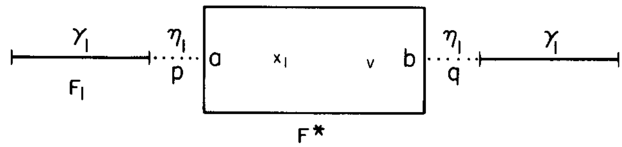
Case 2. $c_1 = \langle \gamma_1, \eta_1 \rangle$
If $d = 1$, then $c(S) := \langle \gamma_1, \eta_1 \rangle$
Else $\{d \ge 2\}$ compute $\alpha = \mathrm{ANCH}(h; c_2, \ldots, c_d)$

2a. If $\alpha + \eta_1 \ge \langle \gamma_1 + 1 \rangle$, then $c(S) := \langle \gamma_1 + 1 \rangle$
2b. If $\langle \gamma_1 + 1 \rangle > \alpha + \eta_1 > \langle \gamma_1 \rangle$, then $c(S) := \alpha + \eta_1$
2c. Else $\{\alpha + \eta_1 \le \langle \gamma_1 \rangle\} c(S) := \langle \gamma_1, \eta_1 \rangle$.

Let $S = (h; F_1, \ldots, F_d)$ be an input in the domain of OP1 with $c(F_i) = c_i$ for all i. Recall that F_1 is balanced and in the disjoint combination the maxcut δ occurs only over F_1. If γ_1 is the cutwidth of F_1 ($= \delta - 1$), we want to determine whether the optimal cost of S is indeed $\langle \delta \rangle$ (as in the disjoint combination) or whether the cutwidth is $\delta - 1$, in which case the optimal combination of S will be balanced. One way to determining whether the cutwidth is δ or $\delta - 1$ would be to compute OPT(h; $c_2, \ldots, c_d$) and check the necessary and sufficient condition given in Lemma 6. If the cutwidth is δ, then the disjoint combination is optimal with cost $\langle \delta \rangle$. However, if the cutwidth is $\delta - 1$, then OPT(h; $c_2, \ldots, c_d$) does not contain enough information to compute the whole cost sequence (cf. the comments after Lemma 6). For this reason, OP1 employs a different (recursive) way to compute the cutwidth and the rst of the cost sequence at the same time.

We describe now how to construct a combination F achieving the cost c returned by the algorithm. It suffices to argue in each case that $c(F) \le c$, since we argue later that the opposite inequality has to hold for any combination, that is, that $c(S) \ge c$. If $\gamma_1 = \eta_1$ or $h + \eta_1 > \gamma_1$, then OP1 returns the disjoint combination. In Case 1, where η_1 occurs on both sides in F_1, OP1 finds the two points p, q of F_1 closest to x_1 on each side where η_1 occurs. These points are next to two nodes a and b of F_1. If $\eta_1 = \gamma_2$, then $a = b = x_1$ (i.e., p and q are in the two gaps bordering x_1) and $F_1(x_1) \le \eta_1$. In this case (where $\eta_1 = \gamma_2$), let F_1^* be the cut-function with a single node x_1 and value $F_1^*(x_1) = F_1(x_1) - \eta_1$; clearly, $c(F_1^*) = \langle 0, 0 \rangle = c_1^*$. If $\eta_1 < \gamma_2$, then the restriction F_1' of F_1 to the interval $[a, b]$ has cost c_1'. Let F_1^* be obtained from F_1' by subtracting a cut of η_1. In this case also the cost of F_1^* is $c_1' - \eta_1 = c_1^*$. Now OP1 calls OPT to find an optimal combination F^* of the root v with height h and $F_1^*, F_2, \ldots, F_d$. Note that, even if $d = 1$ and $c_1^* = \langle 0, 0 \rangle$, F^* is not a degenerate cut-function because of the "edge" (v, x_1); that is, $\alpha \ne \langle 0, 0 \rangle$. Without loss of generality assume that F^* is oriented so that it preserves the order of the

FIGURE 9



nodes of F_1^*. (We can reverse F^* if necessary to ensure this.) Let F be the combination of S obtained by inserting F^* between the points p and q of F_1. That is, F orders the nodes as follows: first come the nodes of F_1 strictly to the left of a, then the nodes of F^*, and finally the nodes of F_1 to the right of b. (See Figure 9; of course, F^* may have more nodes besides the ones shown in the figure. Nodes a, x_1, and b are ordered as in the figure. However, the position shown for v is arbitrary; node v could be to the left of a or to the right of b or anywhere in between.) Clearly, the ordering of all the nodes in F is proper.

OP1 checks to see if F has cutwidth less than δ $(= \gamma_1 + 1)$. The cut of F outside the support of F^* is equal to the cut of F_1. The cut of F in the support of F^* is η_1 more than the cut of F^*, because of the cut η_1 of F_1 that we subtracted. If $\alpha + \eta_1 \geq \langle \gamma_1 + 1 \rangle$ (Case 1a), then the cutwidth of F is at least $\gamma_1 + 1$, and OP1 returns the disjoint combination. If $\alpha + \eta_1 < \langle \gamma_1 + 1 \rangle$ (Case 1b, 1c, 1d), then the cutwidth of F is γ_1, and OP1 returns F. In case 1b ($\alpha + \eta_1 > \langle \gamma_1 \rangle$), there are two points within the support of F^* on each side of v where F has cut γ_1; therefore, $c(F) = \alpha + \eta_1$. In Case 1c ($\alpha + \eta_1 = \langle \gamma_1 \rangle$), one side of F^*, say the right side has cut less than γ_1. Then, the first maxcut γ_1 of F on the right side occurs outside the support of F^*, and the mincut before this point is η_1. Thus, $c(F) \leq \langle \gamma_1, \eta_1 \rangle$. In Case 1d ($\alpha + \eta_1 < \langle \gamma_1 \rangle$), the two closest maxcut points of F on each side are the same as the maxcut points of F_1, the two mincut points before them are p, q and the restriction of F to $[p, q]$ has cost $\alpha + \eta_1$. The cutwidth of $\alpha + \eta_1$ is greater than η_1 since $\alpha \neq \langle 0, 0 \rangle$. Therefore, $c(F) = \langle \gamma_1, \eta_1, \alpha + \eta_1 \rangle$. In all cases, the heavy side of F coincides with the heavy side of F^*.

In Case 2, we argue that the cost of the combination we construct is at most c without using the fact that c_1 has length 2, that is, that either the mincut η_1 of F_1 occurs only on one side, or the cut at x_1 is γ_1. Let p be the point of F_1 closest to x_1 (on the light side) where the mincut η_1 occurs. Since $\eta_1 < \gamma_1$, the maximum cut of F_1 between x_1 and p is strictly less than γ_1. Assume without loss of generality that p is to the right of x_1. If $d = 1$, that is, there are no other subfunctions, form a combination F by placing the root v at point p. The maximum cut to the right of v is γ_1 (at the point where F_1 has this cut), and the mincut before this point is η_1 (right next to v). Since the maximum cut of F_1 in the interval (x_1, p) is less than γ_1, the maximum cut of F to the left of v is γ_1. Also the cut at v is at most γ_1 because $h + \eta_1 \leq \gamma_1$. Therefore, $c(F) \leq \langle \gamma_1, \eta_1 \rangle$.

If $d \geq 2$, OP1 calls ANCH to find the best anchored combination $\bar{F}'$ of v and the rest of the F_i's. Let F be the combination of S obtained by inserting F' into F_1 immediately to the right of p (i.e., in the gap of F_1 that contains p); F' is oriented so that the side of the anchor faces x_1 (see Figure 10). Clearly, this is a proper ordering of the nodes.

Now OP1 checks the cutwidth of F. In Case 2a, OP1 returns the disjoint combination. In Cases 2b and 2c, it returns the combination F. The cut of F in the support of F' exceeds the cut of $\bar{F}'$ by η_1: the "edge" (v, x_1) plays the role of the anchor. The maximum cut of F outside the support of F' is γ_1 as in the $d = 1$ case. Thus, $\alpha + \eta_1 < \langle \gamma_1 + 1 \rangle$ implies that the cutwidth of F is γ_1. If $\alpha + \eta_1 \leq \langle \gamma_1 \rangle$ (Case 2c), then the maximum cut of $\bar{F}'$ on the side without the anchor is less than

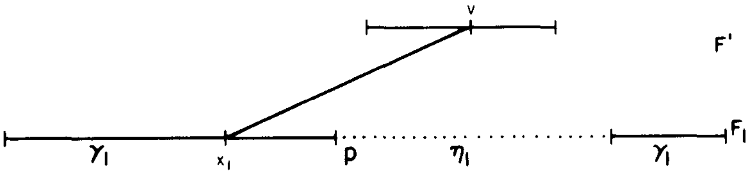


FIGURE 10

$\gamma_1 - \eta_1$. (Recall that the heavy side is the one with the anchor.) Thus, the maximum cut of F to the right of v occurs outside the support of F' and the mincut before this point is η_1. Therefore, $c(F) \le \langle \gamma_1, \eta_1 \rangle$.

In Case 2b, α has cutwidth $\gamma_1 - \eta_1$ and is balanced. Therefore, F has a cut γ_1 to the right of v at a point within the support of F'. If F has a cut γ_1 also to the left of v within the support of F', then clearly $c(F) = \alpha + \eta_1$. Suppose that the closest point u_1 to the left of v where F has a cut γ_1 is outside the support of F', that is, to the left of p. (This corresponds to the case where the maximum cut $\gamma_1 - \eta_1$ of $\bar{F}'$ occurs on the side with the anchor only at the end of the anchor.) If F has a mincut of η_1 between v and u_1, then $c(F) \le \langle \gamma_1, \eta_1 \rangle < \alpha + \eta_1$. (We will see later that this is impossible if $c_1 = \langle \gamma_1, \eta_1 \rangle$.) Otherwise, the mincut between v and u_1 is $\eta_1 + 1$ and occurs at p. It follows that the points between u_1 and p do not play any role in the cost of F; that is, we can treat the cut of F in the interval (u_1, p) as if it was $\eta_1 + 1$ without affecting the cost. Therefore, $c(F) = \alpha + \eta_1$. In both Cases 2b and 2c, the heavy side of F is the same as the heavy side of $\bar{F}'$; that is, the left side in the figure.

LEMMA 10. *OP1 is correct and monotonic assuming that OPT and ANCH are correct and monotonic for inputs of strictly smaller size.*

PROOF. Let $S = (h; F_1, \ldots, F_d)$ be an input in the domain of OP1 with $c(F_i) = c_i$ for all i. Let c be the cost returned by OP1 on input S. We prove first the monotonicity and then the correctness of OP1.

(a) *Monotonicity.* Let $\hat{S} = (\hat{h}; \hat{F}_1, \ldots, \hat{F}_d)$ where $\hat{S} \le S$. After sorting the costs of the $\hat{F}_i$'s, we have again $\hat{c}_1 \ge \cdots \ge \hat{c}_d$ with $\hat{c}_i \le c_i$. We have to exhibit a combination $\hat{F}$ of $\hat{S}$ with cost at most c. We do not assume that $\hat{S}$ is in the domain of OP1. If $c = \langle \gamma_1 + 1 \rangle$, then the disjoint combination of $\hat{S}$ will do. Thus, we may assume that c has cutwidth γ_1. Let $G = \text{OPT}(h; F_2, \ldots, F_d)$ and $\hat{G} = \text{OPT}(\hat{h}; \hat{F}_2, \ldots, \hat{F}_d)$. From the correctness and monotonicity of OPT, $c(\hat{G}) \le c(G)$. From Lemma 6, since S has a combination with cutwidth γ_1, G has cutwidth at most $\gamma_1 - \eta_1 \le \gamma_1 - 1$. (This can be deduced easily also from OP1). Thus, the same is true of $\hat{G}$. If $\hat{c}_1 \le \langle \gamma_1 \rangle$, then form a combination $\hat{F}$ by placing $\hat{F}_1$ and $\hat{G}$ in disjoint intervals and facing each other with their light sides. The cost of $\hat{F}$ is at most $\langle \gamma_1 \rangle$ which is smaller than c. Thus, we may assume that F_1 has cutwidth γ_1 and is balanced. Note that in this case $\hat{S}$ is in the domain of OP1.

Let $\hat{c}_1 = \langle \gamma_1, \theta_1, \ldots \rangle$. We have $\theta_1 \le \eta_1$; furthermore, if $\theta_1 = \eta_1$ and c_1 has length at least 3, then so does $\hat{c}_1$ and its suffix from the third entry on is at most equal to the suffix of c_1. Since c has cutwidth γ_1, we have $\theta_1 \le \eta_1 < \gamma_1$ and $\hat{h} + \theta_1 \le h + \eta_1 \le \gamma_1$. Assume first that Case 1 applies to both S and $\hat{S}$. Form a combination $\hat{F}$ of $\hat{S}$ as in Case 1. Note that the subcases of Case 1 are listed in order of decreasing $\alpha + \eta_1$ and also in order of decreasing cost $c(S)$. Also, in every subcase, $c(S)$ depends monotonically on $\alpha + \eta_1$. Therefore, the cost returned in Case 1 is a monotonic function of $\alpha + \eta_1$. It follows from the monotonicity and correctness of OPT that $c(\hat{F}) \le c$.

Assume now that $\theta_1 < \eta_1$, or $\theta_1 = \eta_1$ but Case 2 applies to S. Form a combination $\hat{F}$ as in Case 2, regardless of whether $\hat{S}$ falls in Case 1 or 2; that is, we insert an optimal anchored combination of v with height h and the rest of the subfunctions into a gap of $\hat{F}_1$ where θ_1 occurs. If $d = 1$, $c(\hat{F}) \le \langle \gamma_1, \theta_1 \rangle \le c$. So assume $d \ge 2$. Note that the cost returned by OP1 in Case 2 is a monotonic function of $\alpha + \eta_1$. It follows, therefore, that if Case 2 applies to S, then $c(\hat{F}) \le c$, because of the correctness and monotonicity of ANCH.

Finally, suppose that Case 1 applies to S but not $\hat{S}$; then $\theta_1 \le \eta_1 - 1$. Let $\alpha =$ OPT$(h; c_1^*, c_2, \ldots, c_d)$ and $\beta =$ANCH$(\hat{h}; \hat{c}_2, \ldots, \hat{c}_d)$. From the correctness of OPT, $c(G) \le \alpha$: removing F_1^* from the combination achieving the cost α cannot increase the cost. From Lemma 7, $\beta < c(\hat{G}) + 1$. Therefore, $\beta + \theta_1 < c(\hat{G}) + 1 + \theta_1 \le c(G) + \eta_1 \le \alpha + \eta_1$. Since c has cutwidth γ_1, it follows that $\beta + \theta_1 < \alpha + \eta_1 < \langle \gamma_1 + 1 \rangle$. If $\beta + \theta_1 > \langle \gamma_1 \rangle$, then also $\alpha + \eta_1 > \langle \gamma_1 \rangle$ and $c(\hat{F}) \le \beta + \theta_1 < \alpha + \eta_1 \le c$. If $\beta + \theta_1 \le \langle \gamma_1 \rangle$, then $c(\hat{F}) \le \langle \gamma_1, \theta_1 \rangle < c$.

(b) *Correctness.* Let F_{op} be an optimal combination with cost c_{op} and let c be the cost returned by OP1.

If $\eta_1 = \gamma_1$, then the cut of F_1 next to the root x_1 on each side is γ_1; addition of the edge (x_1, v) will create a cut of $\gamma_1 + 1$. Thus we may assume that $1 \le \eta_1 < \gamma_1$.

If the cutwidth of the optimal combination F_{op} is $\gamma_1 + 1(= \delta)$, then $c_{op} \ge c$. So we assume that the cutwidth of F_{op} is γ_1. Let u_1, u_1' be the closest points to x_1 on each side where the cut γ_1 of F_1 occurs. Since $\gamma(F_{op}) = \gamma_1$, the root v and all other subfunctions have to lie in the interval (u_1, u_1'). The mincut of F_1 in this interval is η_1. The cut at the root v is at least $h + \eta_1$. Since the cutwidth of F_{op} is γ_1, it follows that $h + \eta_1 \le \gamma_1$.

Case 1. Suppose at first that c_1 has length at least 3. Let F_1^* be the cut-function with cost c_1^* that we defined when we described the combination returned by OP1. That is, we take the two nodes a and b on each side of x_1, which are next to the gaps where η_1 occurs. We restrict F_1 to the interval $[a, b]$ and then subtract a cut of η_1. Note that a and b are in the interval (u_1, u_1') both in F_1 and F_{op}. Let F^* be the combination of v (with height h) and $F_1^*, F_2, \ldots, F_d$ contained in F_{op}. From the correctness of OPT, $c(F^*) \ge \alpha$. The support of F^* is contained in the interval (u_1, u_1'). The cut of F_{op} in the support of F^* exceeds the cut of F^* by at least η_1. Therefore, $c_{op} \ge \alpha + \eta_1$. Since F_{op} has cutwidth γ_1, we have $\alpha + \eta_1 < \langle \gamma_1 + 1 \rangle$. In Case 1b, $c_{op} \ge \alpha + \eta_1 = c$. In Case 1c ($\alpha + \eta_1 = \langle \gamma_1 \rangle$), there is a point, say y, in the support of F^* at which F_{op} has cut γ_1. The mincut of F_{op} ($\eta_1(c_{op})$) is clearly at least η_1. Either y is the root v itself, or the mincut of F_{op} between v and y is at least $\eta_1 + 1$ (because v and y are in the support of F^*). In either case, $c_{op} \ge \langle \gamma_1, \eta_1 \rangle = c$. Finally, in Case 1d, the best that F_{op} can do is to have mincut η_1 on both sides of the root v. In this case, these two mincut points must be outside and on different sides of the support of F^*. The restriction of F_{op} between these two mincut points has cost at least $\alpha + \eta_1$. Therefore, $c_{op} \ge \langle \gamma_1, \eta_1, \alpha + \eta_1 \rangle$.

Case 2. Now, either F_1 has cut γ_1 at its root x_1 or the mincut η_1 occurs only on one side. As we argued before, the root v and the rest of the subfunctions lie in the interval (u_1, u_1'). If F_1 has cut γ_1 at x_1, then they all lie either in the interval (u_1, x_1) or in the interval (x_1, u_1'). Assume without loss of generality that v is in the interval (x_1, u_1'). If F_1 has cut γ_1 at x_1, redefine u_1 to be x_1; otherwise, we let u_1 to be as before the first maxcut point of F_1 to the left of x_1. In either case, v and $F_2, \ldots, F_d$ lie in the interval (u_1, u_1'), the mincut of F_1 in this interval is at least η_1, and F_{op} has cut γ_1 at u_1 and u_1'.

We show now that $c_{op} \geq \langle \gamma_1, \eta_1 \rangle$. If $u_1 = x_1$, then the mincut of F_{op} between v and u_1 is at least $\eta_1 + 1$ because of the "edge" (v, x_1). If $u_1 \neq x_1$ is on the heavy side of F_1, the mincut of F_{op} between v and u_1 is again at least $\eta_1 + 1$ because F_1 has at least this mincut in the interval (u_1, x_1), and because of the "edge" (v, x_1) in the interval (x_1, v). If $u_1 \neq x_1$ is on the light side of F_1, then F_1 (and F_{op}) has mincut at least $\eta_1 + 1$ in the interval (x_1, u_1') (and therefore also between v and u_i'). Thus, in all cases the mincut of F_{op} between v and one maxcut point is at least $\eta_1 + 1$. Therefore, $c_{op} \geq \langle \gamma_1, \eta_1 \rangle$. This takes care of the case $d = 1$ and Case 2c. Thus, we may assume $d \geq 2$, and that c is computed in Case 2a or 2b.

Let F' be the combination of v (with height h) and $F_2, \ldots, F_d$ contained in F_{op}. The support of F' is contained in (u_i, u_i'). The cut of F_{op} exceeds the cut of F' in this interval by at least $\eta_1 + 1$ on one side of v and η_1 on the other side (because of F_1 and the "edge" (v, x_1)). From the correctness of ANCH and Lemma 7, if $\alpha \geq \langle \gamma_1 + 1 - \eta_1 \rangle$ (Case 2a) then, either F' has cutwidth $\gamma_1 + 1 - \eta_1$ (at least) or it has cutwidth $\gamma_1 - \eta_1$ and is balanced. Either case implies a cutwidth of at least $\gamma_1 + 1$ for F_{op}. Suppose now that α is balanced with cutwidth $\gamma_1 - \eta_1$ (Case 2b). From Lemma 7 (Case 2 of the lemma), the fictitious cut placed at the end of the anchor (the cutwidth of the disjoint combination of v and $F_2, \ldots, F_d$) is $\gamma_1 - \eta_1$. Assume without loss of generality that the mincut of F_{op} between v and u_1 is (at least) $\eta_1 + 1$. Regard a cut of 1 from v to u_1 as an anchor on F' and a cut of $\gamma_1 - \eta_1$ at u_1 as the fictitious cut at the end of the anchor. Let $\bar{F}'$ be the resulting anchored combination. Since F_{op} has an additional cut of η_1 throughout the support of $\bar{F}'$, we have $c_{op} \geq c(\bar{F}') + \eta_1 \geq \alpha + \eta_1 = c$. $\square$

AN1(h; $c_1, \ldots, c_d$)
Let δ be the cutwidth of the disjoint combination.
If $\delta = 1$, then return $\langle 1, 1 \rangle$;
Compute $\alpha = \mathrm{OPT}(h - 1; c_2, \ldots, c_d)$.

Case 1. If $\alpha \geq \langle \delta - 1 \rangle$, then return $\langle \delta, 1 \rangle$.

Case 2. If $\alpha < \langle \delta - 1 \rangle$ and F_1 is not balanced (i.e., $c_1 = \langle \delta \rangle$), then return $\langle \delta, 1, \alpha + 1 \rangle$ if $\alpha \neq \langle 0, 0 \rangle$, and $\langle \delta, 1, 1 \rangle$ if $\alpha = \langle 0, 0 \rangle$.

Case 3. Else $\{\alpha < \langle \delta - 1 \rangle$ and F_1 is balanced with cutwidth $\delta - 1\}$.

3a. If $\alpha > (\delta - 1) - c_1$, or $\alpha = (\delta - 1) - c_1$ and they are completely balanced, then return $\langle \delta, 1, \alpha + 1 \rangle$ if $\alpha \neq \langle 0, 0 \rangle$, and $\langle \delta, 1, 1 \rangle$ if $\alpha = \langle 0, 0 \rangle$.
3b. Otherwise, return $\langle \delta \rangle$.

Let $S = (h; F_1, \ldots, F_d)$ be an input in the domain of AN1, and let c be the cost returned by AN1 on input S. Recall that the maximum cut in the disjoint combination occurs only over the first subfunction F_1 and possibly at the root. Thus, if $\delta = 1$, there is only one subfunction (F_1) and the obvious (the disjoint) combination with the anchor has cost $\langle 1, 1 \rangle$. We can take the heavy side to be the one with the anchor.

Suppose $\delta > 1$. From Lemma 7, since the disjoint combination is not balanced, either $\bar{c}(S)$ is balanced with cutwidth δ or $\bar{c}(S) = \langle \delta \rangle$. The second case happens iff there is a combination of S with cutwidth $\delta - 1$ or there is a combination in which δ occurs only at the root. Suppose that we decrease the height of the root by 1. The effect of this on any combination is to decrease the cut at the root by 1. Therefore, $\bar{c}(S) = \langle \delta \rangle$ iff there is a combination of v with height $h - 1$ and the F_i's with cutwidth $\delta - 1$. In order to test if this is the case, AN1 uses the condition of Lemma 6. It calls OPT on input $S' = (h - 1; F_2, \ldots, F_d)$. OPT returns an optimal combination F' of S'. If there is a combination of v with height $h - 1$ and $F_1, \ldots, F_d$ whose cutwidth is $\delta - 1$, F_1 must be balanced (because otherwise

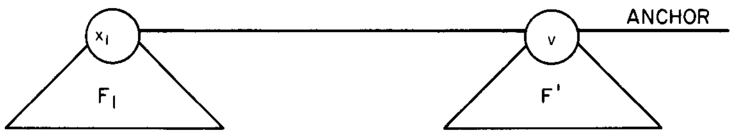


FIGURE 11

$\gamma(F_1) = \delta$), and the condition of Lemma 6 must be satisfied for c_1 and $c(F')$. If the condition is satisfied (Case 3b), AN1 constructs the join of F_1 and F' that has cutwidth $\delta - 1$ as in the proof of Lemma 6, increases the cut at the root by 1, and adds the anchor to form an anchored combination $\bar{F}$ with cost $\langle \delta \rangle$. The heavy side of $\bar{F}$ is, of course, the side with the anchor.

In all other cases (1, 2, and 3a), AN1 forms an anchored combination $\bar{F}$ by placing F_1 and F' in disjoint intervals and facing each other with their light sides; the anchor is placed over the heavy side of F' (see Figure 11). In the disjoint combination of v with height $h - 1$ and $F_1, \ldots, F_d$, the cutwidth δ occurs only over F_1 (and not at the root). Therefore, the cutwidth of the disjoint combination of S' is at most $\delta - 1$, which can occur only on one side. From the correctness of OPT, $c(F') = \alpha \leq \langle \delta - 1 \rangle$. The cut of $\bar{F}$ at any point within F' is one more than the cut of F': on the one side because of the anchor, on the other side because of the "edge" (v, x_1), and at the root because we subtracted 1 from its height. Therefore, the cutwidth of $\bar{F}$ is δ. The cutwidth δ occurs over F_1; furthermore, it does not occur over F' unless $\alpha = \langle \delta - 1 \rangle$ (in which case it occurs under the anchor or at the root).

In Case 1, $\alpha = \langle \delta - 1 \rangle$. The maximum cut of $\bar{F}$ to the left of v occurs for the first time within F_1, and the mincut before this point is 1. Thus, $c(F) \leq \langle \delta, 1 \rangle$. The heavy side is the one with the anchor. In Cases 2 and 3a ($\alpha < \langle \delta - 1 \rangle$), the closest maxcut points of $\bar{F}$ on each side of v are in F_1 and at the end of the anchor. The mincut on each side before these maxcut points is 1, and the cost of the restriction of $\bar{F}$ in the interval between the two mincut points is $\alpha + 1$. Therefore, if the cutwidth of $\alpha + 1$ is 1 (which means $\alpha = \langle 0, 0 \rangle$), then $c(\bar{F}) = \langle \delta, 1, 1 \rangle$; otherwise $c(\bar{F}) = \langle \delta, 1, \alpha + 1 \rangle$. The heavy side of $\bar{F}$ is the same as the heavy side of F', that is, the side with the anchor.

LEMMA 11. *AN1 is correct and monotonic assuming that OPT is correct and monotonic for inputs of smaller size.*

PROOF. Let $S = (h; F_1, \ldots, F_d)$ be an input in the domain of AN1 with $c(F_i) = c_i$ for all i. Let c be the cost returned by AN1 on input S.

(a) *Monotonicity.* Let $\hat{S} = (\hat{h}; \hat{F}_1, \ldots, \hat{F}_d)$ with $\hat{S} \leq S$. After sorting the costs of the $\hat{F}_i$'s we have again $\hat{c}_1 \geq \cdots \geq \hat{c}_d$ with $\hat{c}_i \leq c_i$ for all i. If the cutwidth of the disjoint combination of $\hat{S}$ is less than δ, or if it is δ but occurs only at the root, then $\bar{c}(\hat{S})$ is at most $\langle \delta \rangle \leq c$ (Lemma 7). Thus, we may assume that the cutwidth of the disjoint combination of $\hat{S}$ is δ and occurs over some subfunction. This subfunction can be only the first one ($\hat{F}_1$) because this is the case with the disjoint combination of S. Therefore, $\hat{S}$ is in the domain of AN1. Observe now that the cost returned by AN1 is a monotonic function of α and c_1. Since $\hat{c}_1 \leq c_1$, and from the monotonicity of OPT, it follows that $\bar{c}(\hat{S}) \leq c$.

(b) *Correctness.* Let $\bar{F}_{op}$ be the optimal anchored combination of S and c_{op} its cost. We have to show that $c \leq c_{op}$. If $\delta = 1$, this is obvious. So assume $\delta > 1$. From Lemma 7, $\bar{F}_{op}$ has cutwidth δ and it is balanced, unless F_{op} has cutwidth $\delta - 1$ or cutwidth δ occurring only at the root. As we argued above, the latter case can happen only if F_1 is balanced and the condition of Lemma 6 is met. Since the disjoint combination of v with height $h - 1$ and $F_2, \ldots, F_d$ has cost at most $\langle \delta - 1 \rangle$, we have $c_1 > \alpha$. Clearly, the first entry of $(\delta - 1) - c_1$ is at most $\delta - 2$. Therefore, if $\alpha \geq \langle \delta - 1 \rangle$ (Case 1), the condition of Lemma 6 is not satisfied. If $\alpha < \langle \delta - 1 \rangle$, the condition is explicitly checked in Case 3. Therefore, if $\bar{F}_{\text{op}}$ is unbalanced, the algorithm correctly returns $c = \langle \delta \rangle = c_{\text{op}}$.

Assume now that $\bar{F}_{\text{op}}$ is balanced. The largest cost that AN1 may return is $\langle \delta, 1 \rangle$. Therefore, for $c > c_{\text{op}}$ to hold, the minimum cut in each side of $\bar{F}_{\text{op}}$ before the first cut of δ occurs must be 1, and the cut at the root must be less than δ. Suppose this is the case. On one side, the one edge is the anchor. Therefore, besides the fictitious cut of δ at the end of the anchor, there is no maxcut under the anchor; that is, the maximum cut of F_{op} on the side of the anchor is at most $\delta - 2$. For concreteness, let's say the anchor goes to the right of the root v. Let u_1 be a maxcut point of F_1 and u_1' a second such point on the other side of x_1 if F_1 is balanced. The cut of F_1 at u_1 (and u_1') is at least $\delta - 1$. Therefore, u_1 (and u_1' if F_1 is balanced) is to the left of v. Let u_1 be the point that is closest to v. If F_1 is unbalanced, the cut of F_1 at u_1 is δ. If F_1 is balanced, then x_1 is to the left of u_1, and therefore the cut at u_1 due to F_1 and the "edge" (v, x_1) is also δ. Since $\bar{F}_{\text{op}}$ has cutwidth δ, the rest of the subfunctions must lie entirely to the right of u_1. Let F' be the combination of v with height $h - 1$ and $F_2, \ldots, F_d$ contained in F_{op}. From the correctness of OPT, $c(F') \geq \alpha$. At each point within F', the cut of $\bar{F}_{\text{op}}$ is 1 more than the cut of F: on the one side, the anchor contributes 1; on the other side, F_1 or the "edge" (v, x_1) contributes 1, and at the root because we subtracted 1 from its height. For the mincut of 1 to occur on both sides of v and the cut at v to be less than δ, we must have $\gamma(F') < \delta - 1$. Therefore $\alpha < \langle \delta - 1 \rangle$. In this case F' lies entirely in the interval between the two mincut points, and the cost of the restriction of $\bar{F}_{\text{op}}$ to this interval is at least $c(F') + 1 \geq \alpha + 1$. Thus, if $\alpha = \langle 0, 0 \rangle$, then $c_{\text{op}} \geq \langle \delta, 1, 1 \rangle = c$; if $\alpha \neq \langle 0, 0 \rangle$, then $c_{\text{op}} \geq \langle \delta, 1, \alpha + 1 \rangle = c$. □

The optimal layout of a rooted tree T can be computed by the following algorithm CUT(T).

CUT(T)

Let v be the root of T, $h(v)$ its height, $x_1, \ldots, x_d$ its children, and $T_1, \ldots, T_d$ the subtrees of T rooted at the x_i's.

CUT(T) := OPT($h(v)$; CUT(T_1), . . . , CUT(T_d))

THEOREM 2. *CUT computes (the cut-function of) an optimal layout of T.*

PROOF. This follows from Proposition 1 and the monotonicity and correctness of OPT. □

Example. Let us consider the application of the algorithm on a complete binary tree with all node heights 0. The cost of a leaf (the trivial tree) is $\langle 0, 0 \rangle$. The cost of a subtree of height 1 is easily seen to be $\langle 1, 1 \rangle$, and of a subtree of height 2 is $\langle 2, 1, 1 \rangle$ (the disjoint layout achieves these costs). The problem becomes interesting from the third level up. Consider a node v of height 3 in the tree with 2 subtrees of cost $\langle 2, 1, 1 \rangle$. The disjoint layout has cutwidth $\delta = 3$ and is balanced. Thus, OPT goes to Case 3 and calls AN1 with the one subtree as argument. In AN1, α is set to $\langle 0, 0 \rangle$ (OPT of the trivial tree); Case 3 applies and $\alpha < \delta - 1 - \langle 2, 1, 1 \rangle = 2 - \langle 2, 1, 1 \rangle = \langle 1, 1 \rangle$. Thus, AN1 returns $\langle 3 \rangle$ and OPT gives the cost of the tree as $\langle 3 \rangle$ (Case 3b).

Consider now a node v of height 4 in the tree with two subtrees of cost $\langle 3 \rangle$. The disjoint layout has cutwidth $\delta = 3$ and is balanced; its cost is $\langle 3, 1, 1 \rangle$. Case 3 of OPT applies again, and AN1 is called with one of the subtrees as the argument. Now Case 2 of AN1 applies and AN1 returns $\langle 3, 1, 1 \rangle$. In OPT, $\alpha = \langle 3, 1, 1 \rangle > \langle \delta - (t-1) \rangle = \langle 3 - (1-1) \rangle = \langle 3 \rangle$; thus, OPT returns $\langle 3, 1, 1 \rangle$.

In general, if OPT takes two subtrees of cost $\langle k, 1, 1 \rangle$ as arguments, it will return $\langle k+1 \rangle$, and with two subtrees of cost $\langle k+1 \rangle$ as inputs it returns $\langle k+1, 1, 1 \rangle$. Therefore, the cost of a complete binary tree of height $2h$ ($h \geq 1$) is $\langle h+1, 1, 1 \rangle$, and of a complete binary tree of height $2h-1$ ($h \geq 2$) is $\langle h+1 \rangle$. □

7. *Time Complexity*

First we prove that the algorithm can be implemented to run in time $O(n^2)$ (for the whole tree). Then we show that, if the node heights are small, then with a slight modification the running time can be improved to $O(n \log n)$.

Let us consider the time complexity of OPT. First we observe that if several subfunctions have the same cutwidth and the same type (balanced or not) then, eventually, only one will be examined in the computation. The only times when more components of a cost sequence besides the cutwidth and the type are needed are in procedures OP1 and AN1, for F_1, the first argument. When OP1 or AN1 is called, the maxcut of the disjoint combination occurs only over one subfunction, F_1. Therefore, the rest of the arguments ($F_2, \ldots$) have a strictly smaller cutwidth-type combination. Thus, if several subfunctions (cost sequences) have the same cutwidth-type combination, then the only one among them that might be the first argument of OP1 or AN1 (and therefore whose cost sequence beyond the second entry might be needed) is the one with the smallest cost.

A second observation is that when a procedure returns a cost, it does not have to return the whole sequence. Rather the returned sequence can be described using pointers. For example, in step 3a, AN1 will return δ, 1, and a pointer to α with a number (1) to be added to α. More formally, a cost sequence is represented as a linked list with a number on each link; the number has to be added to all subsequent entries. With this representation all returns from procedure calls take constant time. Also, c_1^* in Case 1 of OP1 can be generated from c_1 in constant time. Note that the first k entries of a sequence can be generated in time proportional to k.

At the beginning, we perform some initialization steps. First we sort the cost sequences $c_1, \ldots, c_d$ of the subtrees on the basis of cutwidth and type; this takes time $O(d \log d)$. Let s be the number of different cutwidth-type combinations. For each such combination, we find the minimum cost sequence. This takes time proportional to the sum of the lengths of the cost sequences, $\sum_{i=1}^{d} |c_i|$. From here on, we keep only the smallest cost sequence for each combination, and the number of cost sequences that have this cutwidth-type combination, in sorted order. In addition we compute the following information. For $j = 1, \ldots, s$, let DC_j be the disjoint combination of the cost sequences with cutwidth-type combination equal to or smaller than the jth one. For each j we compute the maximum cut of DC_j on each side and the two deepest subfunctions over which this cut occurs. It is easy to see that, given this information for DC_{j+1}, we can compute it for DC_j in constant time. Therefore, this step takes $O(s)$ time. From this information, we can compute in constant time all we need to know to decide what action to take in OPT (or ANCH), that is, the maxcut of the disjoint combination, where it occurs, and so forth.

After these initialization steps, in subsequent recursive calls, the arguments will be already sorted, and the information about the cuts of the disjoint combination will be already available, except in Case 1 of OP1 where c_1^* is out of place. If c_1^* does not agree with any of $c_2, \ldots, c_d$ in cutwidth and type, then we can insert it and update the information in time d (sequential search). If c_1^* agrees with some c_i in cutwidth and type, then $\min\{|c_1^*|, |c_i|\}$ additional time is spent in updating the minimum cost sequence for this combination. We measure the size l of the input to a procedure by the sum of the lengths of the smallest cost sequences for the different cutwidth-type combinations; that is, if several c_i's have the same combination, then only the length $|c_i|$ of the smallest c_i with this cutwidth-type combination is counted in l. Thus, when OP1 calls OPT (Case 1), the time spent in updating the information is bounded from above by $d \cdot \Delta l$, where Δl is the difference of the sizes of the inputs.

Let *opt*, *anch*, *op*1, and *an*1 be the running times. The recursive equations are as follows (ignoring constant factors).

$$opt(l) \le \max\{op1(l), an1(l)\} + \text{constant}.$$

$$anch(l) \le \max\{op1(l), an1(l)\} + \text{constant}.$$

$$op1(l) \le \max\{opt(l - k) + dk, anch(l - 1)\} + \text{constant}.$$

Here k is the difference of the size of the input to OP1 from that of the input to OPT.

$$an1(l) \le opt(l - k) + k + \text{constant}.$$

Here, k is the length of the first argument c_1 of AN1; note that the time to check the condition in step 3 of AN1 is proportional to k.

With the obvious initial conditions we can deduce that all running times are $O(dl)$. Therefore, the time complexity of OPT is $O(d \cdot \sum_{i=1}^{d} |c_i|)$. If F_i is the cut-function of an optimal layout for the subtree T_i, then the length of c_i is at most equal to the number of nodes of T_i plus 1. Therefore, $\sum_{i=1}^{d} |c_i| \le n + d$, where n is the number of nodes in the whole tree T. Since $\sum d$ over all the nodes of the tree is $n - 1$ (the number of edges), the total running time of CUT over the whole tree is $O(n^2)$.

Suppose now that the height of every node u of the tree is bounded from above by a constant multiple of its degree; that is, there is a constant r such that for every node u, $h(u) \le r \cdot d(u)$, where $d(u)$ is the degree of u. Note that, in the standard cutwidth problem, this is the case since all nodes have height 0. We show that this case covers also the black and white pebble game. Since the nodes have "small" heights, the cutwidth of every rooted subtree T_i with n_i nodes is at most $(r + 1)n_i$. We do our accounting now using the cutwidths of the cost sequences rather than their lengths. From Section 5, we know that the length of a cost sequence is at most equal to its cutwidth + 2. We now measure the size l of an input to a procedure (OPT, ...) by the number of different cutwidth-type combinations s + the sum of the cutwidths of the different such combinations; that is, if several c_i's have the same cutwidth and type, their cutwidth is counted only once in l. The recurrence relations for *opt*, *anch*, and *an*1 stay the same. However, we can get a better bound now for *op*1. Suppose we are in Case 1 of OP1 where c_1^* is out of place. If c_1^* does not agree with any of $c_2, \ldots, c_d$ in cutwidth and type, then we can insert it and update the information in time proportional to $\gamma_1 - \gamma(c_1^*)$ (sequential search). If c_1^* agrees with some c_i in cutwidth and type, then $\gamma(c_1^*)$

additional time is spent in updating the minimum cost sequence for this combination. Thus, when OP1 calls OPT (Case 1), the time spent in updating the information is proportional to the difference of the sizes of the inputs. Therefore, we have now:

$$op1(l) \le \max\{opt(l-k)+k, anch(l-1)\} + \text{constant}.$$

With the obvious initial conditions, we can deduce now that all running times are proportional to the size of the input. Including also the initialization steps, the time complexity of the one level problem is $O(d \log d + \sum_{i=1}^{d} \gamma_i)$. It has been shown in [5] that the cutwidth of a fixed degree tree with all node heights 0 is $O(\log n)$; addition of "small" heights at the nodes increases the cutwidth by a constant. Therefore, in the case of fixed degree trees, this bound implies that the time complexity of the algorithm for the whole tree is $O(n \log n)$.

In general, however, this bound still gives an overall time $O(n^2)$ for the whole tree. The reason is that we might have many $(\Omega(n))$ nodes such that the subtree rooted at each of them has large $(\Omega(n))$ cutwidth. This means that there are many nodes v such that one of the subtrees (T_1) below v has cutwidth much larger than the rest of the subtrees $(T_2, \ldots, T_d)$. Let us see what happens in this case. Let $c_1 = \langle \zeta_1, \eta_1, \zeta_2, \ldots \rangle$ be the cost of T_1, and suppose that the length of c_1 (and the cutwidth) is much larger than the cutwidth of the disjoint combination of the rest of the subtrees. Since the maximum cut of the disjoint combination occurs only over T_1, OPT calls OP1. OP1 replaces c_1 by $c_1^* = \langle \zeta_2 - \eta_1, \eta_2 - \eta_1, \ldots \rangle$ and calls back OPT. If the maximum cut of the disjoint combination occurs still over the first subfunction, OPT will call again OP1. OP1 will replace c_1^* by $c_2^* = \langle (\zeta_3 - \eta_1) - (\eta_2 - \eta_1), \ldots \rangle = \langle \zeta_3 - \eta_2, \eta_3 - \eta_2, \ldots \rangle$ and will call back OPT. This looping will continue until a j is found such that in the disjoint combination of $c_j^* = \langle \zeta_{j+1} - \eta_j, \eta_{j+1} - \eta_j, \ldots \rangle$ and $c_2, \ldots, c_d$, the maximum cut occurs over another subfunction (or at the root). At this point, OPT will compute $\alpha = \mathrm{OPT}(h(v);$ $c_j^*, c_2, \ldots, c_d)$. Afterward, the changes will not propagate before ζ_j; that is, the computed cost c will agree with c_1 in the first $2(j-1)$ entries. This can be seen by examining what OP1 will do in the first j calls, and verifying that in the first $j-2$ calls Case 1d will apply, whereas in the $(j-1)$st call Case 1c or 1d will apply. A more direct way to see it is as follows. Let F_{j-1}^* be the cut-function obtained by restricting the first function F_1 to the interval between the two η_{j-1} mincut points and then subtracting a cut of η_{j-1}. The cost of F_{j-1}^* is $c_{j-1}^* = \langle \zeta_j - \eta_{j-1}, \eta_j - \eta_{j-1}, \ldots \rangle$. Since the cutwidth of the disjoint combination of $F_{j-1}^*, F_2, \ldots, F_d$ occurs only over the first subfunction, their optimal combination F^* has cost at most $\langle \zeta_j - \eta_{j-1} + 1 \rangle$. Inserting F^* between the two points of F_1 where η_{j-1} occurs will create at most a cut $\zeta_j + 1 \le \zeta_{j-1}$ on the one side. Therefore, the maxcuts and mincuts $\zeta_1, \eta_1, \ldots, \zeta_{j-1}, \eta_{j-1}$ will not be affected. In the worst case, the optimal cost c will be $\langle \zeta_1, \eta_1, \ldots, \zeta_{j-1}, \eta_{j-1} \rangle$. Clearly, the maximum cut of the disjoint combination occurs over the first subfunction as long as $\zeta_i - \eta_{i-1} > \gamma_2 + d, h(v)$. If $|c_1| \gg \gamma_2 + r \cdot d$, then most of the time is spent reading an initial portion of c_1 that is useless.

Therefore, what we could let CUT do to avoid this unnecessary work is the following. After the initialization steps, we check if $\gamma(T_1) > \gamma(T_2) + r \cdot d$. If this is the case, we read $c(T_1)$ backwards until the appropriate ζ_{j+1}, η_j are found—this will take at most $\gamma_2 + r \cdot d$ time. After this, we can continue with the algorithm as before. To be able to read the cost sequence backward, all we need is the sum of the numbers on the links; clearly, this sum can be updated in constant time when the cost sequence changes.

With this slight modification, the one level problem is solved in time $O(d \log d + \sum_{i=2}^{d} \gamma_i)$. Let $n_1 \geq n_2 \geq \cdots \geq n_d$ be the numbers of nodes in the subtrees (n_i is not necessarily the number of nodes of T_i). For each i, at most $i - 1$ subtrees can have cutwidth greater than $(r + 1)n_i$, because the cutwidth is smaller than $r + 1$ times the size of a tree. Therefore, $(r + 1)n_i \geq \gamma_i$, and an upper bound for one level is $O(d \log d + \sum_{i=2}^{d} n_i)$. Since $\sum d$ over all the nodes is $n - 1$, the first term contributes $O(n \log n)$ to the total running time. An easy induction can show that the contribution of the second term is also $O(n \log n)$: Suppose that there are d subtrees $T_1, \ldots, T_d$ hanging from the sons of the root of tree T. Let $n, n_1 \geq \cdots \geq n_d$ be the numbers of nodes of $T, T_1, \ldots, T_d$, respectively. Assume by induction that summing the second term over all nodes of a subtree T_i gives at most $n_i \log n_i$. Then, the sum of the second term over all nodes of T is bounded by $\sum_{i=2}^{d} n_i + \sum_{i=1}^{d} n_i \log n_i = n_1 \log n_1 + \sum_{i=2}^{d} n_i(1 + \log n_i) \leq n_1 \log n + \sum_{i=2}^{d} n_i \log n = n \log n$, because $n_i \leq n/2$ for $i \neq 1$.

The optimal layout of the tree can be computed within the same time bounds as the cost. Cut-functions and layouts are represented by doubly linked lists of nodes. For each η_i in the cost sequence of a cut-function, we have one or two pointers at the point(s) where the mincut η_i occurs. It is easy to see now how to cut and paste together the subfunctions to form an optimal combination (layout) at the same time that the cost is computed.

8. *Laying Out Tree Circuits on a Line*

Suppose that the tree models a circuit that we want to embed on a line (see Figure 2) and that the active elements have different heights. If we set the height of a node in our algorithm to be the height of the corresponding active element, the cutwidth computed by the algorithm will not be equal to the height of the embedded circuit, if the wires come into an element from the top as in Figure 2. The reason for this is that we defined the cut of a layout L at a point p where a node v is embedded to be the sum of $h(v)$ and the number of edges that cross over v; that is, only the edges that are not incident to v (and pass over it) contribute to the cut. However, when we lay out the circuit, at the left side of the element that corresponds to v, we have also to take into account the edges that come to v from the left, and on the right side we have to count the edges that come to v from the right. Thus, setting the height of a node equal to the height of the corresponding element, models the situation where the wires come into the elements from the sides: wires coming from the left enter the left side of the active element, and wires coming from the right enter the right side. If this is the case, the algorithm will return the minimum height of a layout for the circuit. In order to achieve this height, it might be necessary to move the active elements sufficiently apart from each other so that the wires will have enough space to change tracks in order to pass above the elements.

Even if the wires enter from the top (as in Figure 2) we can use the algorithm (with different heights) to find a minimum height layout for the circuit, assuming that there is enough horizontal space for the wires to change tracks. Consider a layout L of the tree T and the corresponding layout of the circuit. Let the height $h(u)$ of a node u be the height of the corresponding active element + $\lceil d(u)/2 \rceil$, where $d(u)$ is the degree of u. Clearly, the height of the layout of the circuit is at least as large as the cutwidth of L. Therefore, the minimum height of a layout for the circuit is at least as large as $\gamma(T)$. If an optimal layout for T has the property that at each node half of the incident edges go left and half of them go to the right, then, clearly, the height of the corresponding layout of the circuit is $\gamma(T)$. The layout constructed by the algorithm has this property.

LEMMA 12. *Let T be a tree rooted at node v and L the layout of T constructed by the algorithm. For every node u, $\lceil d(u)/2 \rceil$ edges incident to u have one direction, and $\lfloor d(u)/2 \rfloor$ have the opposite direction. For the root v, if $d(v)$ is odd, the majority of the edges go to the heavy side of L.*

PROOF. The proof is by induction. The claim clearly holds for the trivial tree. So, it suffices to consider the one level problem. Consider a root v with sons $x_1, \ldots, x_d$. Let $T_1, \ldots, T_d$ be the subtrees rooted at the sons with optimal layouts $L_1, \ldots, L_d$. Assume that the claim holds for the L_i's and let L be the combined optimal layout. Since the layout of a subtree does not change, the claim holds for all nodes u other than v and the x_i's. To prove the claim for v and its children, it suffices to show the following. Let $F_1, \ldots, F_d$ be cut-functions with roots $x_1, \ldots, x_d$, respectively, and let F be the optimal combination of v and the F_i's constructed by the algorithm. (1) For each i the light side of F_i faces v in F, and (2) $\lceil d/2 \rceil$ of the x_i's are on the heavy side of F and $\lfloor d/2 \rfloor$ on the light side. Notice that the disjoint combination has these two properties.

The two properties can be shown recursively going through the different cases in the algorithms. In the case of ANCH and AN1, we count also the anchor as an edge. Thus, for OPT, in Case 1, the claim follows from the fact that the disjoint combination has the properties. In Case 2, the claim reduces to OP1 having the properties. In Case 3, it reduces to AN1 having the properties where the heavy side of the anchored combination is the side with the anchor. Similarly, the properties can be verified for the other procedures. In the case of AN1, the observation is that when we join in Lemma 6 two functions, they face each other's root with their light sides. □

9. *The Black and White Pebble Game*

The pebble game is played on directed acyclic graphs. The rules of the black and white pebble game are as follows:

(1) A white pebble can be placed on any node and a black pebble removed from any node at any time.
(2) A black pebble can be placed on a node u, a white pebble removed from a node u, or a white pebble on u changed into black, only if all children of u have pebbles.
(3) The graph has no pebbles at the beginning and the end of the game, and every node receives (and loses) a pebble at least once.

Let us denote $p(D)$ the number of pebbles needed to play the black and white game on a dag D. Determining the number of pebbles needed to play the black version of the game (where there are no white pebbles) is known to be PSPACE-complete [13a], and the same is believed for the black and white game. When no repebbling is allowed in the black and white game (i.e., in rule 3, every node receives and loses a pebble exactly once), the problem is NP-complete [17]. In the case of rooted trees, repebbling does not help. Determining $p(T)$ for a rooted tree T turns out to be a special case of the cutwidth problem with heights.

THEOREM 3. *Let T be a rooted tree, and denote by $od(u)$ the out-degree (number of children) of a node u. Define the height $h(u)$ of a node u to be $od(u) + 1$. Then, $p(T) = \gamma(T)$.*

PROOF

(1) $p(T) \geq \gamma(T)$. Fix a strategy that pebbles T with $p(T)$ pebbles, and assume without loss of generality that each node is pebbled exactly once. Thus, each node either receives only a black pebble, or only a white pebble, or it receives a white pebble whose color is changed later to black. Define the *pebbling time* $t(u)$ of a node u as follows. In the first case, $t(u)$ is the time that u receives the black pebble $+ \epsilon$; in the second case, $t(u)$ is the time that u loses the white pebble $- \epsilon$; and in the third case, $t(u)$ is the time that the white pebble on u is changed into black, where ϵ is a sufficiently small amount of time. (Think of $t \pm \epsilon$ as right after or right before t.) Clearly, the nodes of T have distinct pebbling times.

Consider now the layout L that orders the nodes according to their pebbling times. We show that the cutwidth of L is at most $p(T)$. Since the height of a node u is at least equal to the number of edges incident to u, the cut of L at a node u is at least as large as the cut of L in the two intervals bordering u. Thus, it suffices to show that the cut of L at a node u is at most equal to the number of pebbles present at time $t(u)$.

From rule 2 and the definition of pebbling time, it follows that, at time $t(u)$, node u and all its children have pebbles. The number of these pebbles is $od(u) + 1 = h(u)$. It remains to show that for every edge (x, y) that passes over u there is one pebble on a distinct node of T. Consider such an edge (x, y) where x is the father of y. There are two cases: (a) $L(y) < L(u) < L(x)$, and (b) $L(x) < L(u) < L(y)$. In either case, y has a pebble at $t(y)$ and at $t(x)$ (the pebbling time of its father). Since every node is pebbled exactly once, y has a pebble also at $t(u)$. Since every node has a unique father, this pebble on y is not counted in $h(u)$ (y is not a child of u), nor for any other edge passing over u. Therefore, at time $t(u)$, there are at least $cut_L(u)$ pebbles on T, and the cutwidth of L is at most $p(T)$.

(2) $p(T) \leq \gamma(T)$. Let L be a layout with cutwidth $\gamma(T)$. Pebble the tree as follows. Let u be a node and $f(u)$ its father. If u comes before $f(u)$ in the layout L, then a black pebble is placed on u at time $L(u)$ and removed at time $L(f(u)) + \epsilon$; if $f(u)$ comes before u in L, then a white pebble is placed on u at time $L(f(u)) - \epsilon$ and removed at time $L(u)$. For the root r, a black pebble is placed on r at time $L(r)$ and removed immediately. Clearly, for every node x, at time $L(x)$ when a black pebble is placed on x or a white pebble removed from x, all children of x have pebbles. Therefore, the rules of the pebble game are observed. It is easy to verify that the number of pebbles at time $L(u)$ is equal to the cut of L at u. Thus, this pebbling strategy uses no more than $\gamma(T)$ pebbles. □

Since all nodes have small heights in Theorem 3, the number of pebbles needed to play the black and white pebble game on a tree (and an optimal strategy) can be determined in time $O(n \log n)$.

ACKNOWLEDGMENT. I wish to thank the referee for many insightful comments and suggestions that helped improve the presentation of this material.

REFERENCES

1. BRENT, R.P., AND KUNG, H.T. On the area of binary tree layouts. *Inf. Proc. Lett. 11* (1980), 44–46.
2. CHUNG, F. R. K. Some problems and results in labelings of graphs. In *The Theory and Applications of Graphs*, G. Chartrand, ed. Wiley, New York, 1981, pp. 255–263.
3. CHUNG, F.R.K. On the cutwidth and the topological bandwidth of a tree. *SIAM J. Alg. Disc. Meth.*, to appear.

4. CHUNG, F. R. K. On optimal linear arrangements of trees. *Comput. Math. Applic. 10* (1984), 43–60.
5. CHUNG, M., MAKEDON, F., SUDBOROUGH, I. H., AND TURNER, J. Polynomial tile algorithms for the min cut problem on degree restricted trees. In *Proceedings of the 23rd Annual Symposium on the Foundations of Computer Science*. IEEE, New York, 1982, pp. 262–271.
6. COOK, S. A., AND SETHI, R. Storage requirements for deterministic polynomial finite recognizable languages. *J. Comput. Syst. Sci. 13* (1976), 25–37.
7. DOLEV, D., AND TRICKEY, H. Embedding a tree on a line. IBM tech. rep. RJ3368, IBM, San Jose, Calif., 1982.
8. FELLER, A. Automatic layout of low-cost quick turnaround random-logic custom LSI devices. In *Proceedings of the 13th Annual Design Automation Conference*. IEEE, New York, 1976, pp. 79–85.
9. FISHER, M. J., AND PATERSON, M. Optimal tree layout. In *Proceedings of the 12th Annual ACM Symposium on the Theory of Computing* (Los Angeles, Calif., Apr. 28–30), ACM, New York, 1980, pp. 177–189.
10. FOSTER, M. J., AND KUNG, H. T. Recognizing regular languages with programmable building blocks. In *VLSI-81*, J. P. Gray, ed. Academic Press, Orlando, Fla., 1981, pp. 75–84.
11. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, Calif., 1979.
12. GAREY, M. R., GRAHAM, R. L., JOHNSON, D. S., AND KNUTH, D. Complexity results for bandwidth minimization. *SIAM J. Appl. Math. 34* (1978), 477–495.
13. GAVRIL, F. Some NP-complete problems on graphs. In *Proceedings of the 11th Conference on Information Sciences and Systems*. Johns Hopkins University, Baltimore, Md., 1977, pp. 91–95.
13a. GILBERT, J. R., LENGAUER, T., AND TARJAN, R. E. The pebbling problem is complete in polynomial space. *SIAM J. Comput. 9*, 3 (1980), 513–525.
14. GOLBERG, M., AND KLIPKER, I. An algorithm of a minimal placing of a tree on the line. *Sakharth. SSR Mech. Acad. Moambe 83* (1976), 553–556.
15. LEISERSON, C. E. Area-efficient graph layouts (for VLSI). In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*. IEEE, New York, 1980, pp. 270–281.
16. LENGAUER, T. Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees. *SIAM J. Alg. Disc. Meth. 3* (1982), 99–113.
17. LENGAUER, T. Black-white pebbles and graph separation. AT&T Bell Laboratory Memorandum. AT&T Bell Laboratories, Murray Hill, N.J.
18. LENGAUER, T., AND TARJAN, R. E. The space complexity of pebble games on trees. *Inf. Proc. Lett. 10* (1980), 184–188.
19. LOUI, M. C. The space complexity of two pebble games on trees. LCS rep. No. 133, M.I.T., Cambridge, Mass., 1979.
20. MAKEDON, F., AND SUDBOROUGH, I. H. Minimizing width in linear layouts. In *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming*. Springer Verlag, New York, 1983, pp. 478–490.
21. MAKEDON, F., PAPADIMITRIOU, C. H., AND SUDBOROUGH, I. H. Topological bandwidth. *SIAM J. Alg. Disc. Meth.* to appear.
22. MEGGIDO, N., HAKIMI, S. L., GAREY, M. R., JOHNSON, D. S., PAPADIMITRIOU, C. H. The complexity of searching a graph. In *Proceedings of the 12th Annual Symposium on Foundations of Computer Science*. IEEE, New York, 1981, pp. 376–385.
23. MEYER AUF DER HEIDE, F. A comparison between two variations of a pebble game on graphs. *Theor. Comput. Sci, 13* (1981), 315–322.
24. PERSKY, G., DEUTSCH, D., AND SCHWEIKERT, D. LTX-A minicomputer-based system for automated LSI layout. *J. Des. Autom. Fault Tolerant Comput. 1* (1977) 217–255.
25. SHILOACH, Y. A minimum linear arrangement algorithm for undirected trees. *SIAM J. Comput. 8* (1979), 15–32.
26. VALIANT, L. G. Universality considerations in VLSI circuits. *IEEE Trans. Comput. C-30*, 2 (1981), 135–140.

RECEIVED OCTOBER 1983; REVISED FEBRUARY 1985; ACCEPTED MARCH 1985