SIAM J. COMPUT.
Vol. 6, No. 1, March 1977

# A LINEAR TREE PARTITIONING ALGORITHM*

SUKHAMAY KUNDU† AND JAYADEV MISRA‡

**Abstract.** Given a rooted tree with a positive weight associated with every node, a linear algorithm is presented that will partition the tree into a minimum number of subtrees such that the sum of node weights in no subtree exceed a prespecified value $k$.

**Key words.** tree, partition

**1. Introduction.** Let $T$ be a rooted tree with a positive weight associated with every node. A feasible $k$-partition $C$ of $T$ is a set of edges such that upon removal of these edges from the tree, each of the resulting component *subtrees* has a total node weight (sum of node weights) at most $k$. The problem studied in this paper is to find a feasible $k$-partition of minimum cardinality (an optimal $k$-partition). Note that an optimal $k$-partition partitions the tree into minimum number of components each of whose total weight is less than or equal to $k$. We will use "partition" instead of "$k$-partition" when the context is understood. The weight of each node of $T$ is assumed to be at most $k$.

A more general problem is when the edges are also weighted, and it is required to find a $k$-partition that has minimum sum of edge weights. Lukes [5] has given an $O(k^2n)$ algorithm for this general problem, where $n$ = number of nodes in the tree. We present an $O(n)$ algorithm for the special case of unit edge weights. Our algorithm is also valid for weighted edges if the weights satisfy a certain monotone property, to be defined later. Note that for large $k$, say, order of $\sqrt{n}$, Lukes' algorithm will be less efficient than a linear algorithm.

The tree partition problem arises in partitioning any hierarchical structure into a minimum number of segments when there is a constraint on the size of a segment. For example, distributing a hierarchical data base into a minimum number of pages fits into this model; $k$ denotes the size of a page. Usually the nodes in the data base would be of different sizes reflected by the corresponding weights. Partitioning of logic modules into a minimum number of blocks (when block sizes are fixed) to minimize the total number of interconnections among blocks also fits into this model assuming that the modules are arranged in the form of a tree.

Problems of partitioning arise in several different contexts. For example [3], [4] consider the problem of optimal segmentation of a program into pages so that the average number of interpage branching is minimized under certain assumed probabilities of branching. The problem of optimally partitioning a tree into disjoint *chains* has been considered in [6].

**2. Results.** Let $T_p$ denote the subtree rooted at node $p$, $S(p)$ the set of sons of node $p$, $w(p)$ the weight of node $p$, and $W(p)$ the sum of node weights in $T_p$. The following lemma is fundamental to our algorithm.

* Received by the editors September 24, 1975, and in revised form May 6, 1976.

† Computer Science Department, University of Texas at Austin, Austin, Texas. Now at LOGICON Corporation, San Pedro, California 90733.

‡ Computer Science Department, University of Texas at Austin, Austin, Texas. This research was supported in part by the National Science Foundation under Grant DCR75-09842.

LEMMA 1. *Let $p$ be a node in $T$ such that $W(p) > k$ and $W(r) \leqq k$, $\forall r \in S(p)$. Then there exists an optimal $k$-partition containing the edge $(p, r_0)$, where*

$$W(r_0) = \max_{r \in S(p)} \{W(r)\}.$$

*Proof.* Since $W(p) > k$, any feasible partition $C$ necessarily contains an edge from $T_p$. Let $(u, v)$ be such an edge in $C$ from $T_p$. If $u \neq p$, then $(u, v)$ is in the subtree $T_r$, $r \in S(p)$. Clearly, $C' = C - \{(u, v)\} + \{(p, r)\}$ is a feasible $k$-partition, and is also optimal. We may thus assume that each edge of $C$ from $T_p$ is of the form $(p, r)$. The lemma follows by replacing one of the edges $(p, r)$ in $C$ by $(p, r_0)$.

$r_0$ is called a *heaviest* son of $p$. Note that the sum of the node weights in a subtree (not the weight of the node) determines the heaviest son. $T - T_r$ will denote the rooted tree obtained upon deletion of $T_r$ from $T$.

LEMMA 2. *Let $(p, r)$ be in some optimal partition of $T$. If $C_1$, $C_2$ are optimal partitions of $T - T_r$ and $T_r$ respectively, then $C = C_1 + C_2 + \{(p, r)\}$ is an optimal partition for $T$.*

*Proof.* $C$ is a feasible partition. Let $C'$ be any optimal partition containing $(p, r)$, and let $C_1'$, $C_2'$ denote the set of edges in $C'$ from $T - T_r$ and $T_r$ respectively. Obviously, $|C_1| \leqq |C_1'|$ and $|C_2| \leqq |C_2'|$. Hence $|C| \leqq |C'|$.

Lemmas 1 and 2 lead to the following algorithm. Find a node $p$ such that $W(p) > k$ and $W(r) \leqq k$, $\forall r \in S(p)$. Let $r_0$ be a heaviest son of $p$. Then construct an optimal $k$-partition $C_1$ of $T - T_{r_0}$ and let $C = C_1 + \{(p, r_0)\}$. (Since $W(r_0) \leqq k$, the optimal partition of $T_{r_0}$ is null.)

A node $p$ as above can be located by proceeding along the tree level by level, beginning at the highest level (distance from root) and going down to the root level (level 1). At any stage of the algorithm, we have a single tree which is modified by deletion of a subtree. We let $W^*(p)$ denote the weight of subtree rooted at node $p$ in the modified tree.

ALGORITHM FOR OPTIMAL $k$-PARTITION

**begin**
  $C := \varnothing$; assign $W^*(q) := w(q)$ to all leaf nodes in $T$;
  **for** $i :=$ maximum-level-in-$T$ **downto** 1 **do**
  **begin** process $i$th level:
    **while** there is an unprocessed node $p$ in level $i$ **do**
    **begin** process node $p$:
      remove heaviest sons of $p$ one by one from $S(p)$ until
      $W^*(p) = \sum_{q \in S(p)} W^*(q) + w(p) \leqq k$;
      For every such son $r$ removed, add the edge $(p, r)$ to $C$
    **end** process node
  **end** process level
**end** algorithm;

A simple technique to process node $p$ is to rank order all the sons of $p$ based on their $W^*(\cdot)$ value. The heaviest sons can then be removed one by one until the total weight is less than or equal to $k - w(p)$. This step however, requires $O(|S(p)| \log |S(p)|)$ time to process node $p$ and hence the overall running time of the algorithm becomes $O(n \log n)$.

The following technique for processing node $p$ was suggested by an anonymous referee, resulting in a reduced running time for this step to $O(|S(p)|)$ only. "Process node $p$" step essentially partitions $S(p)$ into two subsets $S_L(p)$ and $S_H(p)$ (light and heavy) such that

$$q \in S_L(p) \quad \text{and} \quad r \in S_H(p) \Rightarrow W^*(q) \leqq W^*(r), \tag{1}$$

$$\sum_{q \in S_L(p)} W^*(q) + w(p) \leqq k, \tag{2}$$

$$\sum_{q \in S_L(p)} W^*(q) + w(p) + W^*(r) > k, \quad \forall r \in S_H(p). \tag{3}$$

This partitioning can be performed by successively splitting $S(p)$ using a linear median finding algorithm [1]. First $S(p)$ is split into two parts $S_L(p)$, $S_H(p)$ satisfying (1) and $|S_H(p)| \leqq S_L(p)| \leqq |S_H(p)| + 1$. Then conditions (2), (3) are checked in $O(|S(p)|)$ time. If both conditions hold, we have located the desired partition. If (2) holds but (3) does not hold, then $S_H(p)$ is split into "lower" and "upper" parts and the algorithm is repeated. If (3) holds but (2) does not hold, then $S_L(p)$ is split.

More formally, the following routine returns the set $S_H$ as value given the inputs $S$ and $k - w(p)$. Medianfind-and-halve$(S, S_1)$ returns the "upper" half $S_1$ of $S$.

```
split(S, k):
begin
    if |S| = 1 then [split := if W*(q), q ∈ S ≦ k then ∅
                                                  else S]
    else begin
        medianfind-and-halve(S, S1);
        t := Σ_{q∉S1} W*(q);
        case t of
            t = k: split := S1;
            t < k: split := split(S1, k − t);
            t > k: split := split(S − S1, k) + S1
        end
end split;
```

Medianfind-and-halve takes linear time. Since we examine a set of approximately half the size in every succeeding step, split$(S(p), k)$ requires

$$O\left(|S(p)| + \frac{|S(p)|}{2} + \frac{|S(p)|}{4} + \cdots\right) = O(|S(p)|)$$

time. Hence the running time of the optimal $k$-partition algorithm is $O(n)$. For small values of $k$, good running time can be obtained by distributing $S(p)$ into $k$ buckets—node $q$ goes to the $i$th bucket for $W^*(q) = i, q \in S(p)$. Then lighter sons' weights are successively added to obtain a value as close as possible, but not exceeding $k - w(p)$.

*Remarks.* The algorithm presented above can be applied to a few other similar problems. First, assume that the edges are also weighted and the weights satisfy the following *monotone* property: on each path from the root to a node of

the tree, the edges closer to the root have smaller weights than those that are further away, and all edges directed away from a node have equal weights. To obtain a $k$-partition with minimum sum of edge weights, the partition algorithm can be used successfully since at each iteration step an edge is added to C which is as close to the root as possible.

Next, consider the partition problem where it is required that each rooted subtree of the partition be a simple chain with total weight not exceeding $k$. We process the nodes $p$ from higher to lower levels as before. In the step "process node $p$," we remove all but the lightest son of $p$, i.e., we find $r \in S(p)$, where

$$W^*(r) = \min_{q \in S[p]} \{W^*(q)\}$$

and add every $(p, q)$, $q \neq r$, $q \in S(p)$ to $C$. If furthermore $w(p) + W^*(r) > k$, then $(p, r)$ is also added to $C$. The running time of this algorithm is also $O(n)$.

**Acknowledgment.** The authors are indebted to an anonymous referee who suggested the use of median finding algorithm in processing a node. This has lowered the running time to $O(n)$ from the previous bound of $O(n \log n)$.

## REFERENCES

[1] M. BLUM, R. FLOYD, V. PRATT, R. RIVEST AND R. TARJAN, *Time bounds for selection*, J. Comput. Systems Sci., 7 (1973), pp. 448–461.

[2] M. R. GAREY, D. S. JOHNSON AND L. J. STOCKMEYER, *Some simplified NP-complete problems*, 6th ACM Symp. on Theory of Computing, Seattle, WA, 1974, pp. 47–68.

[3] B. W. KERNIGHAN, *Optimal sequential partitions of graphs*, J. Assoc. Comput. Mach., 18 (1971), pp. 34–40.

[4] J. KRAL, *To the problem of segmentation of a program*, Information Processing Machines, (1965), pp. 140–149.

[5] J. A. LUKES, *Efficient algorithm for partitioning of trees*, IBM J. Res. Develop., 18 (1974), no. 3, p. 217.

[6] J. MISRA AND R. E. TARJAN, *Optimal chain partitions of trees*, Information Processing Lett., 4 (1975), pp. 24–26.

[7] W. H. HOSKEN, *Optimum partitions of tree addressing structures*, this Journal, 4 (1975), pp. 341–347.