Acta Informatica 13, 53–58 (1980)

Applications of Efficient Mergeable Heaps for Optimization Problems on Trees

Zvi Galil*

Department of Mathematical Sciences, Computer Science Division, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel

Summary. It is shown how to use efficient mergeable heaps to improve the running time of two algorithms that solve optimization problems on trees.

0. Introduction

A *tree* is a directed graph with a distinguished vertex - the *root* such that there is exactly one directed path from the root to any other vertex. If there is an edge from a vertex i to a vertex j we say that j is a *son* of i and i is the *father* of j. Given a tree T, a *subtree* of T at a vertex i is a subgraph of T that is a tree with root i. The *full subtree* at i is the largest subtree at i, and is denoted by T_i. Also, let Γ_i be the set of subtrees at i. The vertex 0 is the root of T.

The input to the problems below will consist of a tree T and two (three) vectors $\bar{a}$, $\bar{b}$ (and $\bar{c}$) that assign to every vertex i in T a pair (triple) of positive real numbers a_i, b_i (and c_i). If T' is a subtree of T, then the *ratio* of T', $r(T') \equiv a(T')/b(T')$, where $a(T') = \sum_{i \in T'} a_i$ and $b(T') = \sum_{i \in T'} b_i$. We now describe the problems we are interested in.

Problem 0. Compute $r_0 = \max_{T' \in \Gamma_0} r(T')$, and find a tree $\tilde{T}_0$ in Γ_0 of maximal size that satisfies $r(\tilde{T}_0) = r_0$.

Problem 1. For every vertex i in T compute $r_i = \max_{T' \in \Gamma_i} r(T')$, and find a tree $\tilde{T}_i$ in Γ_i of maximal size that satisfies $r(\tilde{T}_i) = r_i$.

It seems that one needs to solve many instances of Problem 0 in order to solve Problem 1. However, our only solution to Problem 0 is the solution to Problem 1. So we do not mention Problem 0 from now on. Problem 1 arises in a certain sequencing algorithm as will be explained below. We will show that for every vertex i the tree $\tilde{T}_i$ is uniquely defined. The most complicated problem, Problem 2, arises in computing cost allocation as will be explained below.

* The work of the author was supported in part by the Israel Commission for Basic Research and National Science Foundation grant MCS78-25301

Problem 2. Given a tree T and three vectors $\bar{a}$, $\bar{b}$, $\bar{c}$ ($\bar{a}=\{a_i\}_{i\in T}$, etc.), compute a function $f_T^{\bar{a},\bar{b},\bar{c}}$ that assigns a real number $f_T^{\bar{a},\bar{b},\bar{c}}(i)$ to every vertex i in T. The function is defined recursively as follows: Let $r_0=\min_{T'\in\Gamma_0} r(T')$ and $\tilde{T}_0$ the largest tree in Γ_0 with ratio r_0. If i is in $\tilde{T}_0$, $f_T^{\bar{a},\bar{b},\bar{c}}(i)=r_0$. Otherwise $i\in T_j$ for some j whose father is in $\tilde{T}_0$. Then $f_T^{\bar{a},\bar{b},\bar{c}}(i)=f_{T_j}^{\bar{a}',\bar{b}',\bar{c}'}$, where $\bar{a}'$, $\bar{b}'$, and $\bar{c}'$ are the restrictions of $\bar{a}$, $\bar{b}$, $\bar{c}$ to the vertices of T_j with a single exception: $a_j'=a_j+c_j$; i.e., the a of the root is increased by the corresponding c.

In this paper we show how to solve the two problems in time $O(n\log n)$ where n is the number of vertices of T. This is somewhat surprising because it seems that a solution to Problem 2 may require solving many instances of Problem 1. The solution of Problem 1 appears in Section 1, and the solution of Problem 2 in Sect. 2. In Sect. 3 we show how to use the solution of Problem 1 to derive an $O(n\log n)$ time algorithm for single-machine job sequencing with treelike precedence ordering and linear delay penalties, improving Horn's $O(n^2)$ algorithm [4]; then we show how to use the solution of Problem 2 to derive an $O(n\log n)$ time algorithm for computing the nucleolus or cost allocation for trees improving Megiddo's $O(n^3)$ algorithm [5].

The solutions will use efficient mergeable heaps [1]. A *mergeable heap* is a data structure that supports the operations of insertion, deletion, finding the minimum or the maximum and of disjoint union. A mergeable heap is efficient if it takes $O(n\log n)$ time to execute $O(n)$ operations. Various efficient mergeable heaps are given in [2].

1. The Solution of Problem 1

Given a subtree of T, T', at the root of T, let $P(T')$ be the set of vertices of T, that are not in T' but their fathers are in T'. Horn suggested the following solution that computes r_i and $\tilde{T}_i$ from the leaves of the tree to the root.

Algorithm 0

1. If i is a leaf of T (i.e. it has no sons), then $\tilde{T}_i\leftarrow T_i$ (a tree which consists of a single vertex) and $r_i\leftarrow a_i/b_i$.

2. If $i_1,\ldots,i_k$ are the sons of i and $\tilde{T}_l$, r_l have been computed for every l in $T_{i_1}, T_{i_2},\ldots,T_{i_k}$, then $\tilde{T}_i$ and r_i are computed as follows.

a) Set $\hat{T}_i$ to the vertex i and $r_i\leftarrow a_i/b_i$. ($\hat{T}_i$ is a tree variable whose final value will be $\tilde{T}_i$.)

b) If $P(\hat{T}_i)=\emptyset$, then $\tilde{T}_i\leftarrow\hat{T}_i$ and stop; otherwise find j in $P(\hat{T}_i)$ such that $r_j=\max_{l\in P(\hat{T}_i)} r_l$.

c) If $r_j\geq r_i$, then (i) change $\hat{T}_i$ by adding to it $\tilde{T}_j$ and the edge that enters j; (ii) change r_i respectively; and (iii) go to step b.

d) Otherwise ($r_j<r_i$), $\tilde{T}_i\leftarrow\hat{T}_i$ and stop.

We now prove the correctness of A0. (Horn's proof is quite long.) Let $\hat{\hat{T}}_k$ be the final value of $\hat{T}_k$ in A0. We show by induction on the number of vertices in T_i that r_i is computed correctly, and $\tilde{T}_i = \hat{\hat{T}}_i$ (so $\tilde{T}_i$ is unique). The base (T_i consists of one vertex) is immediate. We assume that it holds for all vertices in $T_{i_1}, \ldots, T_{i_k}$ (as in step 2) and prove it for i. By definition of $\tilde{T}_k$, the following claims hold.

Claim 1. If $l \in P(\tilde{T}_m)$, then $r_l < r_m$.

Claim 2. If $l \in \tilde{T}_i$, then $\tilde{T}_l$ is contained in $\tilde{T}_i$.

Consider the loop that computes $\tilde{T}_i$ at a certain time. Let Γ_i' be the family of trees T that can be obtained from $\hat{T}_i$ by starting with $T = \hat{T}_i$ and repeating the following step a number (possibly zero) of times: Choose some $l \in P(T)$ and add to T $\tilde{T}_l$ and the edge that enters l. An easy induction on the number of times step 2c is executed shows that $\tilde{T}_i \in \Gamma_i'$. (The base is immediate and the induction step follows from Claims 1 and 2.) Now, consider the final Γ_i' (that corresponds to $\hat{\hat{T}}_i$). $\tilde{T}_i \in \Gamma_i'$ and thus $\hat{\hat{T}}_i$ is contained in $\tilde{T}_i$. Note that for every $l \in P(\hat{\hat{T}}_i)$ $r_l < r_i$. This together with Claim 1 imply that any tree in Γ_i that strictly contains $\hat{\hat{T}}_i$ must have a ratio smaller than that of $\hat{\hat{T}}_i$. Thus $\hat{\hat{T}}_i = \tilde{T}_i$.

Horn did not analyze the running time of his algorithm. But a straightforward implementation of the algorithm takes $O(n^2)$ time and this bound is also given in [3] when Horn's algorithm is mentioned.

The algorithm above can be sped up by using efficient mergeable heaps. During the construction of $\tilde{T}_i$ we will maintain the mergeable heaps MH_l for some (not all) vertices l in $\tilde{T}_i$. The elements stored at MH_l will be all the pairs (p, r_p) for p in $P(\hat{T}_l)$. This will enable fast execution of step 2 in A0. A1 – the improved algorithm – is given below.

Algorithm 1

```
1. If $i$ is a leaf of $T$, then $\tilde{T}_i \leftarrow T_i$, $r_i \leftarrow a_i/b_i$, $MH_i \leftarrow \emptyset$, and $\hat{a}_i \leftarrow a_i$, $\hat{b}_i \leftarrow b_i$.
2. If $i_1, \ldots, i_k$ are the sons of $i$ and $\tilde{T}_l$, $r_l$ have been computed for every $l$ in $T_{i_1}$,
   $T_{i_2}, \ldots, T_{i_k}$, then $\tilde{T}_i$ and $r_i$ are computed as follows.
   a) Set $\hat{T}_i$ to the vertex $i$, $r_i \leftarrow a_i/b_i$, $\hat{a}_i \leftarrow a_i$, $\hat{b}_i \leftarrow b_i$ and $MH_i \leftarrow \{(i_1, r_{i_1}) \ldots (i_k, r_{i_k})\}$.
   b) If $MH_i = \emptyset$, then $\tilde{T}_i \leftarrow \hat{T}_i$ and stop; otherwise find a pair $(j, r_j)$ in $MH_i$ with
      maximal second component.
   c) If $r_j \geqq r_i$ then
      (i) $MH_i \leftarrow MH_i \cup MH_j$,
      (ii) $\hat{a}_i \leftarrow \hat{a}_i + \hat{a}_j$, $\hat{b}_i \leftarrow \hat{b}_i + \hat{b}_j$, $r_i \leftarrow \hat{a}_i/\hat{b}_i$, and
      (iii) delete $(j, r_j)$ from $MH_i$, and go to step $b$.
   d) Otherwise $(r_j < r_i)$ $\tilde{T}_i \leftarrow \hat{T}_i$ and stop.
```

The number of insertions is bounded by n. The number of maximum findings is bounded by twice the number of unions, and the number of deletions equals the number of unions. These unions are of disjoint sets (in 2c(i) $(j, r_j) \in MH_i$ and hence $j \in P(\hat{T}_i)$ and $MH_i \cap MH_j = \emptyset$), hence their number is bounded by n. Thus, the number of operations on the mergeable heaps is smaller than $5n$ and hence the running time is $O(n \log n)$.

2. The Solution of Problem 2

One can solve Problem 2 by repeatedly solving Problem 1 (with minimum instead of maximum). Since at most n instances of Problem 1 are solved, by using Horn's algorithm A0 one gets an $O(n^3)$ algorithm. Megiddo solved it differently also in time $O(n^3)$. Obviously, using A1, our improved solution to Problem 1, we get an $O(n^2 \log n)$ algorithm. However, the following simple observation yields an $O(n \log n)$ algorithm. Assume we have solved Problem 1 once and found $r_i = \min_{T' \in \Gamma_i} r(T')$ for every vertex i of T. Let $MH_0 = \{(l, r_l)\}$. Using MH_0 we can locate the vertices of $\tilde{T}_0$ and assign them the value r_0. It remains to compute f for vertices in the T_l's such that $(l, r_l) \in MH_0$. Let (l, r_l) be a pair in MH_0, and let $j_1, \ldots, j_r$ be the sons of l. We now need to solve Problem 1 for T_l with the change in a_l, but we do not need to start the algorithm from the beginning because the change affects only $\tilde{T}_l$. If p is in T_{j_s}, $1 \leqq s \leqq r$, then $\tilde{T}_p$ is the same after the change in a_l. We even do not have to start step 2 in A1 from the beginning (with $\hat{T}_i =$ the vertex i) but with the $\hat{T}_i$ obtained while solving Problem 1 for T. A2, the solution for Problem 2, now follows.

Algorithm 2

1. Put 0 in a queue, set $l=0$. (The algorithm always solves Problem 1 for the tree T_l.) Consider vertices i of T from the leaves up. If i is a leaf of T, then $\hat{T}_i \leftarrow T_i$, $r_i \leftarrow a_i/b_i$, and $\hat{a}_i \leftarrow a_i$, $\hat{b}_i \leftarrow b_i$.

2. Compute r_i and $\hat{T}_i$ for every vertex of T_l that is not a leaf as follows. If $i_1, \ldots, i_k$ are the sons of i and r_p, $\hat{T}_p$ have been computed for every p in $T_{i_1}, \ldots, T_{i_k}$, then set $\hat{T}_i \leftarrow \{i\}$, $r_i \leftarrow a_i/b_i$, $\hat{a}_i \leftarrow a_i$, $\hat{b}_i \leftarrow b_i$ and $MH_i = \{(i_1, r_{i_1}), \ldots, (i_k, r_k)\}$ and execute the procedure $BUILD(i)$.

```
Procedure $BUILD(i)$
a) If $MH_i = \emptyset$ return.
b) Find a pair $(j, r_j)$ in $MH_i$ with minimal second component.
c) If $r_j \leqq r_i$ then
    (i) $MH_i \leftarrow MH_i \cup MH_j$
    (ii) $\hat{a}_i \leftarrow \hat{a}_i + \hat{a}_j$, $\hat{b}_i \leftarrow \hat{b}_i + \hat{b}_j$, $r_i \leftarrow \hat{a}_i/\hat{b}_i$ and
    (iii) delete $(j, r_j)$ from $MH_i$ and go to step $a$.
d) Otherwise $(r_j > r_i)$ return.
```

3. Using MH_l separate $\hat{T}_l$ from T_l and for every vertex p and $\hat{T}_l$ set $f(p) = r_l$. For every vertex q in $P(\hat{T}_l)$ change $\hat{a}_q \leftarrow \hat{a}_q + c_q$, $r_q \leftarrow \hat{a}_q/\hat{b}_q$, and add q to the queue. Delete l from the queue.

4. If the queue is empty stop. Otherwise, let l be the first in the queue, execute $BUILD(l)$ and go to step 3.

Let $\tilde{T}_l$ be the largest optimal subtree at l and let $\tilde{T}_l'$ be the largest optimal subtree at l after a_l is incremented. By the correctness of A1, $\tilde{T}_l'$ can be obtained by starting with $\tilde{T}_l$ and applying steps b and c of A1 (or alternatively $BUILD(l)$ of A2). In fact if we construct $\tilde{T}_l'$ directly using A1 we will get $\tilde{T}_l$ in an intermediate stage. Thus A2 is correct, and the number of operations is still $O(n)$, since all the bounds for the various operations are the same as for A1, except that the bound on the number of minimum findings can be at most $3n$. As a result, the time of A2 is $O(n \log n)$.

3. Applications

The following algorithm is Horn's algorithm [4] for finding optimal single-machine job sequencing with treelike precedence ordering and linear delay penalties.

1. Let $l=1$ and $R_1 \leftarrow \{0\}$. (0 is the root of T.)
2. Compute $r_i = \max\limits_{T' \in \Gamma_i} r(T')$ for every vertex i of T.
3. Let i_l be an element j in R_l such that $r_j = \max\limits_{p \in R_l} r_p$.

Let $R_{l+1} \leftarrow R_l \cup \{\text{sons of } i_l\} - \{i_l\}$, and increase l by 1.

4. If $R_l = \emptyset$ stop; otherwise go to step 3.

A *priority queue* [1] is a data structure that supports the operations of finding maximum, insertions and deletions. Various efficient priority queues (that perform $O(n)$ operations in time $O(n \log n)$) are described in [2]. Steps 1 and 2 above can be implemented in time $O(n \log n)$ by using A1. Steps 3 and 4 above are implemented in time $O(n \log n)$ by using an efficient priority queue to represent R_l. The algorithm above can also be applied to forests (finite collections of trees). The only necessary change is (in step 1) to set $R_1 \leftarrow \{\text{the roots of the given forest}\}$. The same time bound holds.

Megiddo [5] reduced the problem of computing cost allocation for trees to a special case of Problem 2. The collection of c_i's (the appropriate increments) is not given as input. Instead the increment is r_0, the r of the root that has just been computed. ($a_j' = a_j' + r_0$ in the statement of Problem 2.) By modifying in step 3 of A2 to read '... change $\hat{a}_q \leftarrow \hat{a}_q + r_l \ldots$', we derive an $O(n \log n)$ algorithm for solving the problem in [5]. Megiddo assigned the a_i's and b_i's to the edges (e_i's) of the tree and not to its vertices. But if we choose e_i to be the edge that enters i we get equivalent problems.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Reading, Mass: Addison-Wesley 1974
2. Brown, M.R.: The analysis of a practical and nearly optimal priority queue. Stanford Ph.D. Dissertation. STAN-CS-77-600, Stanford University, 1977

3. Bruno, J.L.: Mean weighted flow-time criterion: In Computer and Job-Shop Scheduling Theory, E.G. Coffman, ed., pp. 101–138. New York: John Wiley 1976
4. Horn, W.A.: Single-machine job sequencing with treelike precedence ordering and linear delay panalties. SIAM J. Appl. Math. **23**, 189–202 (1972)
5. Megiddo, N.: Computational complexity of the game theory approach to cost allocation for a tree. MOR **3**, 189–196 (1978)

Note added in proof. It has been pointed out to the author that on $O(n \log n)$ algorithm for finding optimal single-machine job sequencing with treelike precedence ordering and linear delay penalties is not new and can be found in:
Adolphson, D. and the T.C.: Optimal linear ordering. SIAM J. Appl. Math. **25**, 403–423 (1973)

Received December 8, 1977 / July 24, 1979