ELSEVIER

Information Processing Letters 56 (1995) 321-328

Information Processing Letters

Generalized vertex-rankings of trees

Xiao Zhou *,[1], Nobuaki Nagai [1], Takao Nishizeki [2]

Department of System Information Sciences, Graduate School of Information Sciences, Tohoku University, Sendai 980-77, Japan

Received 9 November 1994; revised 6 September 1995
Communicated by T. Lengauer

Abstract

We newly define a generalized vertex-ranking of a graph G as follows: for a positive integer c, a c-vertex-ranking of G is a labeling (ranking) of the vertices of G with integers such that, for any label i, every connected component of the graph obtained from G by deleting the vertices with label $> i$ has at most c vertices with label i. Clearly an ordinary vertex-ranking is a 1-vertex-ranking and vice-versa. We present an algorithm to find a c-vertex-ranking of a given tree T using the minimum number of ranks in time $O(cn)$ where n is the number of vertices in T.

Keywords: Algorithms; Generalized ranking; Graphs; Trees; Lexicographical order; Visible vertices

1. Introduction

A *vertex-ranking* of a graph G is a labeling (ranking) of vertices of G with integers such that any path between two vertices with the same label i contains a vertex with label $j > i$. The *vertex-ranking problem* is to find a vertex-ranking of a given graph G using the minimum number of ranks (labels). The vertex-ranking problem is NP-hard in general [1,7]. On the other hand Schäffer has given a linear algorithm to solve the vertex-ranking problem for trees [8]. Very recently Bodlaender et al. have given a polynomial-time algorithm to solve the vertex-ranking problem for graphs with bounded treewidth [1]. The vertex-ranking of a graph G has applications in VLSI layout and in scheduling the manufacture of complex multi-part products [8,5]; it is equivalent to finding the minimum height vertex separator tree of G.

In this paper we newly define a generalization of an ordinary vertex-ranking. For a positive integer c, a *c-vertex-ranking* (or a *c-ranking* for short) of a graph G is a labeling of the vertices of G with integers such that, for any label i, every connected component of the graph obtained from G by deleting the vertices with label $> i$ has at most c vertices with label i. Clearly an ordinary vertex-ranking is a 1-vertex-ranking and vice-versa. The integer label of a vertex is called the *rank* of the vertex. The minimum number of ranks needed for a c-vertex-ranking of G is called the *c-vertex-ranking number* (or the *c-ranking number* for short) and denoted by $r_c(G)$. A c-ranking of G using $r_c(G)$ ranks is called an *optimal c-ranking* of G. The *c-ranking problem* is to find an optimal c-ranking of a given graph G. The problem is NP-hard in general since the ordinary vertex-ranking problem is NP-hard [1,7]. Fig. 1 depicts an optimal 3-ranking of a tree

* Corresponding author.
[1] Email: {zhou,nishi}@ecip.tohoku.ac.jp.
[2] Email: nagai@anakin.nishizeki.ecei.tohoku.ac.jp.

Elsevier Science B.V.
SSDI 0020-0190(95)00172-7

using three ranks, where vertex numbers are drawn in circles and ranks next to circles.

Consider the process of starting with a connected graph and partitioning it recursively by removing at most c vertices and incident edges from each of the remaining connected subgraphs until the graph becomes empty. The tree representing the recursive decomposition is called a *c-vertex separator tree.* Thus a c-vertex separator tree corresponds to a parallel computation scheme based on the process above. The c-vertex-ranking problem is equivalent to finding a c-vertex separator tree of the minimum height. Fig. 2 illustrates a 3-vertex separator tree of the tree depicted in Fig. 1, where deleted vertex numbers are drawn in ovals.

Let M be a sparse symmetric matrix. Let M' be a matrix obtained from M by replacing each non-zero element by 1. Let G be a graph with adjacency matrix M'. Then an optimal c-vertex ranking of G corresponds to a generalized Cholesky factorization of M having the minimum recursive depth [2,4,6].

In this paper we give an algorithm to solve the c-ranking problem on trees T in time $O(cn)$ for any positive integer c where n is the number of vertices in T. Our algorithm uses techniques employed by Schäffer [8] and Iyer et al. [5] for the ordinary vertex-ranking problem as well as new techniques specific to the c-ranking problem.

2. Preliminaries

In this section we define some terms and present easy observations. Let $T = (V, E)$ denote a tree with vertex set V and edge set E. We often denote by $V(T)$ and $E(T)$ the vertex set and the edge set of T, respectively. We denote by n the number of vertices in T. T is a "free tree", but we regard T as a "rooted tree" for convenience sake: an arbitrary vertex of tree T is designated as the *root* of T. We will use notions as: root, internal vertex, child and leaf in their usual meaning. An edge joining vertices u and v is denoted by (u, v). The maximal subtree of T rooted at vertex v is denoted by $T(v)$. For a c-ranking φ of tree T and a subtree T' of T, we denote by $\varphi|T'$ a restriction of φ to $V(T')$: let $\varphi' = \varphi|T'$, then $\varphi'(v) = \varphi(v)$ for $v \in V(T')$. The definition of a c-ranking immediately implies that a c-ranking of a connected graph labels at most c vertices with the largest rank.

The number of ranks used by a c-ranking φ of tree T is denoted by $\#\varphi$. One may assume without loss of generality that φ uses the consecutive integers $1, 2, \ldots, \#\varphi$ as the ranks. A vertex v of T and its rank $\varphi(v)$ are *visible* (from the root under φ) if all the vertices in the path from the root to v have ranks $\leqslant \varphi(v)$. Thus the root of T and $\#\varphi$ are visible. Denote by $L(\varphi)$ the list of ranks of all visible vertices, and call $L(\varphi)$ the *list of a c-ranking* φ of the rooted tree T. For an integer γ we denote by $count(L(\varphi), \gamma)$ the number of γ's contained in $L(\varphi)$, i.e., the number of visible vertices with rank γ. The ranks in the list $L(\varphi)$ are sorted in non-increasing order. Thus the c-ranking φ in Fig. 1 has the list $L(\varphi) = \{3, 3, 1\}$, and hence $count(L(\varphi), 3) = 2$, $count(L(\varphi), 2) = 0$ and $count(L(\varphi), 1) = 1$. One can easily observe that $count(L(\varphi), \gamma) \leqslant c$ for each rank γ.

We define the *lexicographical order* $\prec$ on the set of non-increasing sequences (lists) of positive integers as follows: let $A = \{a_1, \ldots, a_p\}$ and $B = \{b_1, \ldots, b_q\}$ be two sets (lists) of positive integers such that $a_1 \geqslant \cdots \geqslant a_p$ and $b_1 \geqslant \cdots \geqslant b_q$, then $A \prec B$ if there exists an integer i such that

(a) $a_j = b_j$ for all $1 \leqslant j < i$, and

(b) either $a_i < b_i$ or $p < i \leqslant q$.

We write $A \preceq B$ if $A = B$ or $A \prec B$. A c-ranking φ of T is *critical* if $L(\varphi) \preceq L(\eta)$ for any c-ranking η of T. The optimal c-ranking depicted in Fig. 1 is indeed critical.

For a list A and an integer α, we define a sublist $[\alpha \leqslant A]$ of A as follows:

$$[\alpha \leqslant A] = \{x \in A \mid \alpha \leqslant x\}.$$

Similarly we define sublists $[\alpha < A]$, $[A \leqslant \alpha]$ and $[A < \alpha]$ of A. Obviously if $A \preceq B$ then $[\alpha < A] \preceq [\alpha < B]$ for any $\alpha \geqslant 1$. For lists A and B we use $A \subseteq B$ and $A \cup B$ in their usual meaning in which we regard A, B and $A \cup B$ as multi-sets.

3. Optimal c-ranking

The main result of the paper is the following theorem.

Theorem 1. *An optimal c-ranking of a tree T having n vertices can be found in time* $O(cn)$ *for any positive*

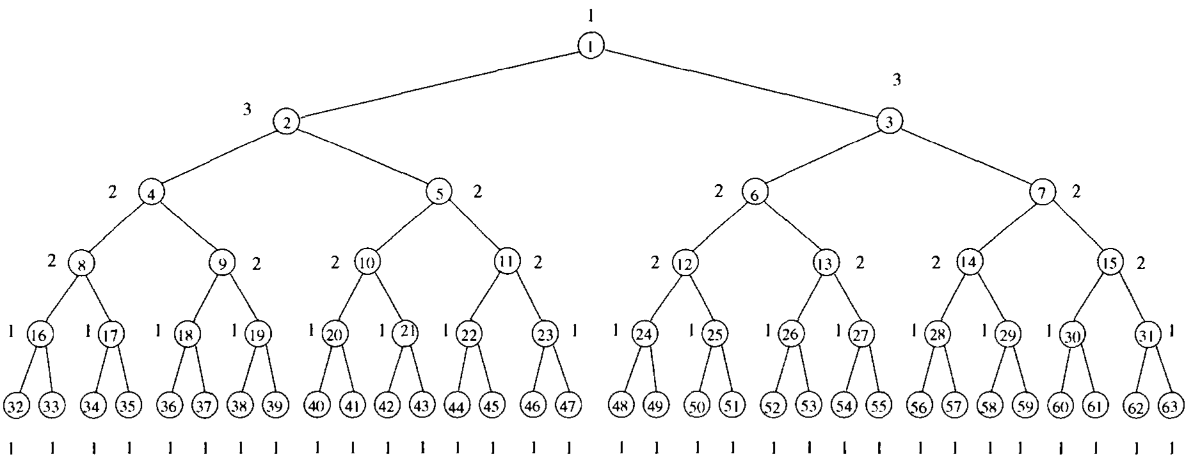

Fig. 1. An optimal 3-vertex-ranking φ of a tree T.

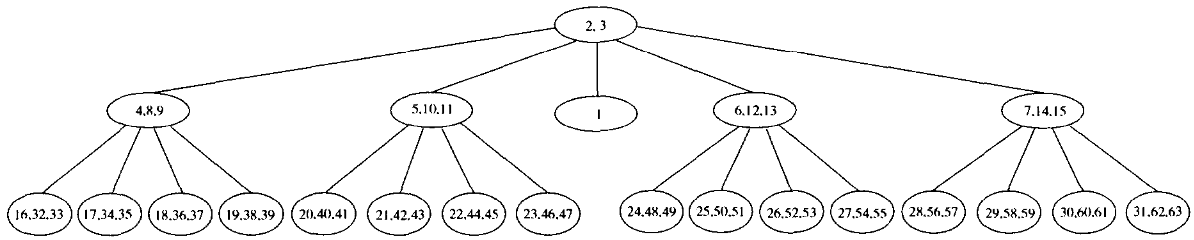

Fig. 2. A 3-vertex separator tree of the tree in Fig. 1.

integer c.

In the remainder of this section we give an algorithm to find a critical c-ranking of a tree T in time $O(cn)$. Our algorithm uses the technique of "bottom-up tree computation". For each internal vertex u of a tree T, we construct a critical c-ranking of $T(u)$ from those of the subtrees rooted at u's children.

One can easily prove the following lemma by induction on n.

Lemma 2. *Every tree T of n vertices has a vertex whose removal leaves subtrees each having no more than $n/2$ vertices.*

Using Lemma 2, we have the following lemma.

Lemma 3. *For any positive integer c, every tree T of n vertices has at most c vertices whose removal leaves subtrees each having no more than n/q vertices, where $q = 2^{\lfloor \log_2(c+3) \rfloor - 1} > (c+3)/4$ and hence $q \geqslant 2$.*

Proof. By Lemma 2 tree T has a vertex whose removal leaves subtrees each having no more than $n/2$ vertices. Clearly the number of subtrees having $n/2^2$ or more vertices does not exceed

$$\left\lfloor \frac{n-1}{n/2^2} \right\rfloor \leqslant 2^2 - 1.$$

By Lemma 2 each of these subtrees has a vertex whose removal leaves subtrees each having no more than $n/2^2$. Therefore T has at most $1 + (2^2 - 1)$ vertices whose removal leaves subtrees each having no more than $n/2^2$. Clearly the number of subtrees having $n/2^3$ or more vertices does not exceed

$$\left\lfloor \frac{n-1}{n/2^3} \right\rfloor \leqslant 2^3 - 1.$$

Repeating this operation p ($\geqslant 1$) times, one can know that T has at most

$$1 + (2^2 - 1) + (2^3 - 1) + \cdots + (2^p - 1) = 2^{p+1} - p - 2$$

vertices whose removal leaves subtrees each having vertices no more than $n/2^p$. Choose $p = \lfloor \log_2(c+3) \rfloor - 1$ so that $2^{p+1} - p - 2 \leqslant c$. Then T has at most c vertices whose removal leaves subtrees each having vertices no more than $n/2^p = n/q$. Note that $q = 2^p > (c+3)/4$ and hence $q \geqslant 2$ for any $c \geqslant 1$. $\square$

Lemma 2 is a special case of Lemma 3 with $c = 1$. By Lemma 3 we have the following lemma.

Lemma 4. *Every tree T of n vertices satisfies $r_c(T) \leqslant 1 + \operatorname{rank} n$.*

Proof. Recursively applying Lemma 3, one can construct a c-vertex separator tree of height $h(n)$ satisfying the following recurrence relation

$$h(n) \leqslant 1 + h\left(\left\lfloor \frac{n}{q} \right\rfloor\right).$$

Solving the recurrence, we have $h(n) \leqslant \operatorname{rank} n$. Note that $h(1) = 0$. Hence $r_c(T) \leqslant 1 + h(n) \leqslant 1 + \operatorname{rank} n$. $\square$

Let $d(u)$ be the number of children of vertex u in T, and let $v_1, v_2, \ldots, v_{d(u)}$ be the children of u. Our idea is to construct a critical c-ranking of $T(u)$ from critical c-rankings φ_i of $T(v_i)$, $i = 1, 2, \ldots, d(u)$. One can easily observe the following lemma.

Lemma 5. *A vertex-labeling η of $T(u)$ is a c-ranking of $T(u)$ if and only if $\eta | T(v_i)$ is a c-ranking of $T(v_i)$ for every i, $1 \leqslant i \leqslant d(u)$, and there are no more than c visible vertices of the same rank under η, that is, $count(L(\eta), \gamma) \leqslant c$ for every rank $\gamma \in L(\eta)$.*

We then have the following lemma.

Lemma 6. *Let φ_i be an arbitrary critical c-ranking of $T(v_i)$, $i = 1, 2, \ldots, d(u)$. Then $T(u)$ has a critical c-ranking η such that $\eta | T(v_i) = \varphi_i$ for every i, $1 \leqslant i \leqslant d(u)$.*

Proof. Let ψ be an arbitrary critical c-ranking of $T(u)$. Since φ_i is critical but $\psi | T(v_i)$ is not always critical, we have $L(\varphi_i) \preceq L(\psi | T(v_i))$ for each i, $1 \leqslant i \leqslant d(u)$. If $L(\varphi_i) \prec L(\psi | T(v_i))$, then let γ_i be an integer such that

(a) $[\gamma_i < L(\varphi_i)] = [\gamma_i < L(\psi | T(v_i))]$, and

(b) $count(L(\varphi_i), \gamma_i) < count(L(\psi | T(v_i)), \gamma_i)$.

Otherwise let $\gamma_i = 0$. Let $\gamma_{\max} = \max\{\gamma_i \mid 1 \leqslant i \leqslant d(u)\}$. Construct a vertex-labeling η of $T(u)$ from ψ and φ_i as follows:

$$\eta(v) = \begin{cases} \max\{\psi(u), \gamma_{\max}\} & \text{if } v = u, \\ \varphi_i(v) & \text{if } v \in V(T(v_i)), \\ & \quad 1 \leqslant i \leqslant d(u). \end{cases}$$

Then $\eta | T(v_i) = \varphi_i$ for all i, $1 \leqslant i \leqslant d(u)$.

We now claim that $L(\eta) \subseteq L(\psi)$. Clearly we have

$$L(\eta) = \{\eta(u)\} \cup \left[\eta(u) \leqslant \bigcup_{i=1}^{d(u)} L(\varphi_i)\right] \tag{1}$$

and

$$L(\psi) = \{\psi(u)\} \cup \left[\psi(u) \leqslant \bigcup_{i=1}^{d(u)} L(\psi | T(v_i))\right]. \tag{2}$$

Consider first the case $\psi(u) > \gamma_{\max}$. In this case we have $\eta(u) = \psi(u)$. Since $\gamma_i < \eta(u)$, we have

$$[\eta(u) \leqslant L(\varphi_i)] = [\eta(u) \leqslant L(\psi | T(v_i))] \tag{3}$$

for every i, $1 \leqslant i \leqslant d(u)$. By (1)–(3) we have $L(\eta) = L(\psi)$. Consider next the case $\psi(u) \leqslant \gamma_{\max}$. In this case $\eta(u) = \gamma_{\max} \geqslant \psi(u)$. For every i, $1 \leqslant i \leqslant d(u)$, such that $\gamma_i = \gamma_{\max}$, by (a) and (b) we have

$$\begin{aligned} &\{\eta(u)\} \cup [\eta(u) \leqslant L(\varphi_i)] \\ &\quad \subseteq [\eta(u) \leqslant L(\psi | T(v_i))] \\ &\quad \subseteq [\psi(u) \leqslant L(\psi | T(v_i))]. \end{aligned} \tag{4}$$

For every i such that $\gamma_i < \gamma_{\max} = \eta(u)$, by (a) we have

$$\begin{aligned} [\eta(u) \leqslant L(\varphi_i)] &= [\eta(u) \leqslant L(\psi | T(v_i))] \\ &\subseteq [\psi(u) \leqslant L(\psi | T(v_i))]. \end{aligned} \tag{5}$$

By (1), (2), (4) and (5) we have $L(\eta) \subseteq L(\psi)$ as desired.

Since $L(\eta) \subseteq L(\psi)$ and ψ is a c-ranking, by Lemma 5 η is a c-ranking. Since $L(\eta) \subseteq L(\psi)$, $L(\eta) \preceq L(\psi)$. Therefore η is critical since ψ is critical. $\square$

Let $m = \max\{\#\varphi_i \mid 1 \leqslant i \leqslant d(u)\}$. Then we have the following lemma.

Lemma 7. $r_c(T(u)) = m$ *or* $m + 1$.

Proof. Clearly $m \leqslant r_c(T(u))$. Therefore it suffices to prove that $r_c(T(u)) \leqslant m+1$. One can extend φ_i, $1 \leqslant i \leqslant d(u)$, to a c-ranking η of $T(u)$ as follows:

$$\eta(v) = \begin{cases} m+1 & \text{if } v = u, \\ \varphi_i(v) & \text{if } v \in V(T(v_i)),\ 1 \leqslant i \leqslant d(u). \end{cases}$$

Thus $r_c(T(u)) \leqslant \#\eta = m+1$. □

The following lemma gives a necessary and sufficient condition for $r_c(T(u)) = m$.

Lemma 8. *$r_c(T(u)) = m$ if and only if there is a rank α, $1 \leqslant \alpha \leqslant m$, such that*
(a) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \alpha) \leqslant c-1$ *and*
(b) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \gamma) \leqslant c$ *for every rank γ, $\alpha+1 \leqslant \gamma \leqslant m$.*

Proof. ($\Longleftarrow$) One can easily extend the critical c-rankings φ_i to a c-ranking η of $T(u)$ with $\#\eta = m$ as follows:

$$\eta(v) = \begin{cases} \alpha & \text{if } v = u, \\ \varphi_i(v) & \text{if } v \in V(T(v_i)),\ 1 \leqslant i \leqslant d(u). \end{cases}$$

Therefore $r_c(T(u)) \leqslant \#\eta = m$, and hence by Lemma 7 $r_c(T(u)) = m$.

($\Longrightarrow$) Suppose that $r_c(T(u)) = m$. By Lemma 6 there is a c-ranking η of $T(u)$ such that $\#\eta = m$ and $\eta|T(v_i) = \varphi_i$ for each i, $1 \leqslant i \leqslant d(u)$. Let $\alpha = \eta(u)$, then (a) and (b) above hold since η is a c-ranking of $T(u)$. □

In order to find a critical c-ranking η of $T(u)$ from φ_i, $i = 1, 2, \ldots, d(u)$, we need the following two lemmas.

Lemma 9. *If $r_c(T(u)) = m+1$, then*

$$\eta(v) = \begin{cases} m+1 & \text{if } v = u, \\ \varphi_i(v) & \text{if } v \in V(T(v_i)),\ 1 \leqslant i \leqslant d(u) \end{cases}$$

is a critical c-ranking of $T(u)$ and $L(\eta) = \{m+1\}$.

Proof. immediate. □

Lemma 10. *If $r_c(T(u)) = m$, then*

$$\eta(v) = \begin{cases} \alpha & \text{if } v = u, \\ \varphi_i(v) & \text{if } v \in V(T(v_i)),\ 1 \leqslant i \leqslant d(u) \end{cases}$$

```
  Procedure Ranking(T(u));
  begin
1 if u is a leaf
    then return a trivial c-ranking: u → 1
2   else
3     begin
4       let v1, v2, ···, vd(u) be the children of u;
5       for i := 1 to d(u) do Ranking(T(vi));
6       find a critical c-ranking of T(u) from critical
          c-rankings of T(vi), i = 1,2,...,d(u), by
          Lemmas 9 and 10;
7     return a critical c-ranking of T(u)
8     end
  end.
```

Fig. 3.

is a critical c-ranking of $T(u)$, where α is the minimum rank such that
(a) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \alpha) \leqslant c-1$ *and*
(b) $\sum_{i=1}^{d(u)} count(L(\varphi_i), \gamma) \leqslant c$ *for every rank γ, $\alpha+1 \leqslant \gamma \leqslant m$.*
Furthermore, $L(\eta) = \{\alpha\} \cup [\alpha \leqslant \bigcup_{i=1}^{d(u)} L(\varphi_i)]$.

Proof. By Lemma 6 there is a critical c-ranking ψ of $T(u)$ such that $L(\psi|T(v_i)) = L(\varphi_i)$ for every i, $1 \leqslant i \leqslant d(u)$. Since $\alpha = \eta(u)$ is the minimum rank satisfying (a) and (b) above, $L(\eta) \preceq L(\psi)$ and hence η is a critical c-ranking of $T(u)$. Clearly

$$L(\eta) = \{\alpha\} \cup \left[\alpha \leqslant \bigcup_{i=1}^{d(u)} L(\varphi_i)\right]. \qquad \square$$

By Lemmas 8, 9 and 10 above we have the recursive algorithm in Fig. 3 to find a critical c-ranking of $T(u)$.

Clearly one can correctly find a critical c-ranking of a tree T by calling Procedure Ranking$(T(r))$ for the root r of T. Therefore it suffices to verify the time-complexity of the algorithm. Let φ_i, $i = 1, 2, \ldots, d(u)$, be a critical c-ranking of $T(v_i)$. Assume without loss of generality that $\#\varphi_1$ and $\#\varphi_2$ are the two largest, possibly equal, numbers among $\#\varphi_i$, $i = 1, 2, \cdots, d(u)$, and that $\#\varphi_1 \geqslant \#\varphi_2$. Let $\#\varphi_2 = 0$ if $d(u) = 1$. Let η be a critical c-ranking of $T(u)$ obtained from φ_i, $i = 1, 2, \ldots, d(u)$, at line 6. Then the following lemma holds, which will be proved later.

Lemma 11. *One execution of line 6 can be done in time $O(x_u + d(u) + c \cdot \#\varphi_2)$, where x_u is the number of vertices which were visible in $T(v_i)$ under φ_i, $1 \leqslant$*

$i \leqslant d(u)$, but are not visible in $T(u)$ under η.

Once a vertex becomes invisible, it will never become visible again. Furthermore, $\sum d(u) \leqslant n$ where the summation is taken over all internal vertices. Therefore the total time counted by the first term x_u and the second term $d(u)$ is $O(n)$ when Procedure Ranking is recursively called for all vertices. Let n_{u_2} be the number of vertices in the second largest tree $T(v_{u_2})$ among $T(v_i), i = 1, 2, \cdots, d(u)$, if $d(u) \geqslant 2$. Then by Lemma 4 we have $\#\varphi_2 \leqslant 1 + \text{rank}\, n_{u_2}$. Note that $\#\varphi_2$ is not always the c-ranking number of $T(v_{u_2})$. The following lemma implies that the total time counted by the third term $c \cdot \#\varphi_2$ is also $O(cn)$. Thus the total running time of Ranking is $O(cn)$. This completes the proof of Theorem 1.

Lemma 12. *Let $V_2 = \{u \in V \mid d(u) \geqslant 2\}$, then*

$$\sum_{u \in V_2} \left(1 + \log_q n_{u_2}\right) = O(n).$$

Proof. For a tree T, let

$$S(T) = \sum_{u \in V_2} \left(1 + \log_q n_{u_2}\right).$$

We now prove by induction on n that

$$S(T) \leqslant 2n - (1 + \log_q n). \tag{6}$$

Trivially (6) holds when $n = 1$. Now assume that $n \geqslant 2$ and (6) holds for any tree having at most $n - 1$ vertices.

Let T be a tree with n vertices rooted at vertex u. One may assume that $d(u) \geqslant 2$. Let $v_1, v_2, \ldots, v_{d(u)}$ be the children of u, and let n_i, $i = 1, 2, \ldots, d(u)$, be the number of vertices of $T(v_i)$, respectively. Then

$$\sum_{i=1}^{d(u)} n_i = n - 1. \tag{7}$$

Assume without loss of generality that $n_1 \geqslant n_2 \geqslant \cdots \geqslant n_{d(u)}$, then $n_{u_2} = n_2$. By (7), the definition of $S(T)$ and the inductive hypothesis we have

$$S(T(u)) = 1 + \log_q n_2 + \sum_{i=1}^{d(u)} S(T(v_i))$$

$$\leqslant 1 + \log_q n_2 + \sum_{i=1}^{d(u)} \{2n_i - (1 + \log_q n_i)\}$$

$$= 2n - \Big\{1 + d(u) + \log_q n_1 + \sum_{i=3}^{d(u)} \log_q n_i\Big\}. \tag{8}$$

Since $q \geqslant 2$ and $2^{d(u)} \geqslant d(u) + 1$, we have

$$\begin{aligned} 1 + d(u) + \log_q n_1 + \sum_{i=3}^{d(u)} \log_q n_i &\geqslant 1 + \log_q 2^{d(u)} + \log_q \{n_1 n_3 n_4 \cdots n_{d(u)}\} \\ &= 1 + \log_q \{2^{d(u)} n_1 n_3 n_4 \cdots n_{d(u)}\} \\ &\geqslant 1 + \log_q \{(d(u) + 1) n_1\} \\ &\geqslant 1 + \log_q \Big\{\sum_{i=1}^{d(u)} n_i + 1\Big\} \\ &= 1 + \log_q n. \end{aligned} \tag{9}$$

Substituting (9) to (8), we have $S(T(u)) \leqslant 2n - (1 + \log_q n)$. □

We finally give in Fig. 4 an implementation of line 6 of Procedure Ranking, which finds a critical c-ranking η of $T(u)$ from the critical c-rankings φ_i, $i = 1, 2, \ldots, d(u)$.

We are now ready to prove Lemma 11.

Proof of Lemma 11. As a data-structure to represent a list $L(\varphi)$ of a c-ranking φ, we use a linked list L_φ consisting of records. Each record contains two items of data: rank γ, $1 \leqslant \gamma \leqslant \#\varphi$ and $count(L(\varphi), \gamma)$ such that $count(L(\varphi), \gamma) \geqslant 1$.

If $d(u) = 1$, then using linked list L_{φ_1} one can easily find α at line 4 in $O(x_u)$ time where $x_u = |[L(\varphi_1) < \alpha]|$. It should be noted that all the x_u vertices of ranks in $[L(\varphi_1) < \alpha]$ were visible but they become invisible after lines 5 and 6 are executed. Thus lines 3–7 can be done total in time $O(x_u)$. Similarly, if lines 20–23 are executed, then at line 21 one can easily find α in $O(x_u)$ time, and hence lines 20–23 can be done in time $O(x_u)$ time.

We now claim that if $d(u) \geqslant 2$ then lines 10–12 and 15–16 can be done total in time $O(d(u) + c \cdot \#\varphi_2)$. We construct a linked list L_s as follows. First set L_s as an empty list. For each i, $1 \leqslant i \leqslant d(u)$, add

```
   Procedure Line-6(φ_1, ..., φ_d(u), η);
   begin
1  η|T(v_i) := φ_i for each i, i := 1,2,...,d(u);   { determine the rank of u as follows. }
2    if d(u) = 1 then
3      begin
4        find a smallest integer α ⩾ 1 such that count(L(φ_1),α) ⩽ c − 1;
5        η(u) := α;
6        L(η) := {α} ∪ [α ⩽ L(φ_1)]
7      end
8    else { d(u) ⩾ 2 }
9      begin
10       find the two largest, possibly equal, numbers among #φ_i, i := 1,2,...,d(u);
           { assume w.l.o.g. that #φ_1 and #φ_2 are these largest numbers and #φ_1 ⩾ #φ_2. }
11       let L_s := [L(φ_1) ⩽ #φ_2] ∪ (⋃_{i=2}^{d(u)} L(φ_i));
12       find a smallest rank α such that 1 ⩽ α ⩽ #φ_2, count(L_s,α) ⩽ c − 1 and
           count(L_s,γ) ⩽ c for all ranks γ, α + 1 ⩽ γ ⩽ #φ_2;
13       if such a rank α exists then
14         begin
15           η(u) := α;
16           L(η) := {α} ∪ [α ⩽ L_s] ∪ [#φ_2 < L(φ_1)]
17         end
18       else
19         begin
20           L_s := L_s ∪ [#φ_2 < L(φ_1)];   { L_s = ⋃_{i=1}^{d(u)} L(φ_i) }
21           find a smallest integer α such that #φ_2 + 1 ⩽ α ⩽ #φ_1 + 1 and count(L_s,α) ⩽ c − 1;
22           η(u) := α;
23           L(η) := {α} ∪ [α ⩽ L_s];
24         end
25     end
   end;
```

Fig. 4.

to L_s all ranks γ ($\leqslant \#\varphi_2$) in L_{φ_i} in the decreasing order of γ until either $count(L_s,\gamma) > c$ or all such ranks γ have been added. Thus line 11 can be done in time $O(c \cdot \#\varphi_2)$. Clearly line 10 can be done in time $O(d(u))$ and lines 12, 15 and 16 in time $O(\#\varphi_2)$. Therefore lines 10–12 and 15–16 can be done total in time $O(d(u) + c \cdot \#\varphi_2)$.

Thus Procedure Line-6 can be done total in time $O(x_u + d(u) + c \cdot \#\varphi_2)$. □

4. Conclusion

We newly define a generalized vertex-ranking of a graph, called a c-ranking, and give an efficient algorithm to find an optimal c-ranking of a given tree T in time $O(cn)$ for any $c \geqslant 1$ where n is the number of vertices in T. If c is a bounded integer, then our algorithm takes linear time. If c is not bounded, our algorithm takes time $O(cn)$. However, if c is large, say $c = n^\varepsilon$ for some $\varepsilon > 0$, then by Lemma 4 $r_c(T)$ is bounded and hence one execution of line 6 can be done in time $O(d(u))$ and consequently our algorithm takes linear time.

We may replace the positive integer c by a function $f : \mathbb{N} \to \mathbb{N}$ to define a more generalized vertex-ranking of a graph as follows: an *f-vertex-ranking* [3] of a graph G is a labeling of the vertices of G with integers such that, for any label i, deletion of all vertices with labels $> i$ leaves connected components, each having at most $f(i)$ vertices with label $i \in \mathbb{N}$. By some trivial modifications of our algorithm for the c-vertex-ranking of a tree, we can find an optimal f-vertex-ranking of a given tree in time complexity of $O(n \max_i f(i))$, where the maximum is taken over all labels i used by the algorithm.

[3] We wish to thank Professor A. Nakayama of Fukushima University for suggesting the f-vertex-ranking of a graph.

A generalized edge-ranking can be defined similarly, and the algorithms for the ordinary edge-ranking of trees [3,10–12] can be extended to find an optimal c-edge-ranking [9].

References

[1] H. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller and Zs. Tuza, Ranking of graphs, in: *Proc. Internat. Workshop on Graph-Theoretic Concepts in Computer Science*, Herrsching, Bavaria, Germany (1994).

[2] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson and T. Kloks, Approximating treewidth, pathwidth and minimum elimination tree height, *J. Algorithms* **18** (1995) 238–255.

[3] P. de la Torre, R. Greenlaw and A. A. Schäffer, Optimal ranking of trees in polynomial time, in: *Proc. 4th Ann. ACM–SIAM Symp. on Discrete Algorithms*, Austin, Texas (1993) 138–144; also in: *Algorithmica*, to appear.

[4] I.S. Duff and J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, *ACM Trans. Math. Software* **9** (1983) 302–325.

[5] A.V. Iyer, H.D. Ratliff and G. Vijayan, Optimal node ranking of trees, *Inform. Process. Lett.* **28** (1988) 225–229.

[6] J.W.H. Liu, The role of elimination trees in sparse factorization, *SIAM J. Matrix Analysis and Applications* **11** (1990) 134–172.

[7] A. Pothen, The complexity of optimal elimination trees, Tech. Rept. CS-88-13, Pennsylvania State University, 1988.

[8] A.A. Schäffer, Optimal node ranking of trees in linear time, *Inform. Process. Lett.* **33** (1989) 91–99.

[9] X. Zhou, M.A. Kashem and T. Nishizeki, Generalized edge-rankings of trees, Tech. Rept. SIGAL, 95-AL-46-10, 73-80, Inf. Proc. Soc. of Japan, July 1995.

[10] X. Zhou and T. Nishizeki, An efficient algorithm for edge-ranking trees, in: *Proc. 2nd European Symp. on Algorithms*, Lecture Notes in Computer Science **885** (Springer, Berlin, 1994) 118–129.

[11] X. Zhou and T. Nishizeki, Finding optimal edge-rankings of trees, in: *Proc. 6th Ann. ACM–SIAM Symp. on Discrete Algorithms* (1995) 122–131.

[12] X. Zhou and T. Nishizeki, Finding optimal edge-rankings of trees – A correct algorithm, Tech. Rept. 9501, Dept. of Inf. Eng., Tohoku University, 1995.