アルゴリズム 46−10
（1995. 7. 20）

# Generalized Edge-Rankings of Trees

Xiao Zhou
Education Center for Information Processing, Tohoku University
Md. Abul Kashem and Takao Nishizeki
Graduate School of Information Sciences, Tohoku University
E-mail: zhou@ecip.tohoku.ac.jp, kashem@nishizeki.ecei.tohoku.ac.jp, nishi@ecei.tohoku.ac.jp

We newly define a generalized edge-ranking of a graph $G$ as follows: for a positive integer $c$, a $c$-edge-ranking of $G$ is a labeling (ranking) of the edges of $G$ with integers such that, for any label $i$, deletion of all edges with labels $> i$ leaves connected components, each having at most $c$ edges with label $i$. The problem of finding an optimal $c$-edge-ranking of $G$, that is, a $c$-edge-ranking using the minimum number of ranks, has applications in scheduling the manufacture of complex multi-part products; it is equivalent to finding a $c$-edge-separator tree of $G$ having the minimum height. We present an algorithm to find an optimal $c$-edge-ranking of a given tree $T$ for any positive integer $c$ in time $O(n^2 \log \Delta)$, where $n$ is the number of vertices in $T$ and $\Delta$ is the maximum vertex-degree of $T$.

**Key words:** Algorithm, Edge-ranking, Tree, Separator tree, Visible edges.

# 木の一般化辺ランキング

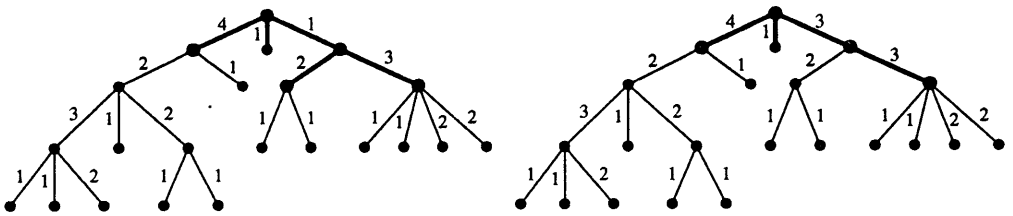
周 暁
東北大学情報処理教育センター
モハメド アブル カシェム　　西関 隆夫
東北大学大学院情報科学研究科

本論文ではグラフの辺ランク付けの一般化として，新たに $c$-辺ランク付けを次のように定義する．グラフ $G$ の $c$-辺ランク付けとは，$G$ の辺に整数のランクを付けて，しかも任意のランク $i$ について，$G$ からランクが $i$ より大きい辺を全て除去すると，どの連結成分にもランク $i$ の辺が高々 $c$ 本しかないようにすることである．明らかに普通の辺ランク付けは 1-辺ランク付けである．本文では与えられた木を，最小のランク数で $c$-辺ランク付けする $O(n^2 \log \Delta)$ 時間のアルゴリズムを与える．ここで，$n$ はグラフ $G$ の点数で，$\Delta$ は最大次数である．

# 1 Introduction

An *edge-ranking* of a graph $G$ is a labeling of edges of $G$ with positive integers such that every path between two edges with the same label $i$ contains an edge with label $j > i$ [1]. Clearly an edge-labeling is an edge-ranking if and only if, for any label $i$, deletion of all edges with labels $> i$ leaves connected components, each having at most one edge with label $i$. The *edge-ranking problem* is to find an edge-ranking of a given graph $G$ using the minimum number of ranks (labels). The problem seems to be NP-complete in general [2]. However, polynomial-time algorithms have been reported for trees. de la Torre *et al.* [2] have given an algorithm to solve the edge-ranking problem for trees $T$ by means of a complicated greedy method in time $O(n^3 \log n)$, where $n$ is the number of vertices in $T$. Very recently a simple and efficient algorithm to solve the edge-ranking problem for trees $T$ in time $O(n^2 \log \Delta)$ has been presented in [4, 5, 6],[1] where $\Delta$ is the maximum vertex-degree of $T$.

In this paper we newly define a generalization of an ordinary edge-ranking. For a positive integer $c$, a *c-edge-ranking* ( or a *c-ranking* for short) of a graph $G$ is a labeling of the edges of $G$ with integers such that, for any label $i$, deletion of all edges with labels $> i$ leaves connected components, each having at most $c$ edges with label $i$. Clearly an ordinary edge-ranking is a 1-edge-ranking. The integer label of an edge is called the *rank* of the edge. A $c$-ranking of $G$ using the minimum number of ranks is called an *optimal c-ranking* of $G$. The *c-ranking problem* is to find an optimal $c$-ranking of a given graph $G$. Fig. 1 depicts two optimal 2-rankings $\varphi$ and $\eta$ of a tree $T$ using four ranks, where the ranks are drawn next to the edges. Connected components obtained from $T$ by deleting all edges with labels $> i$ for the 2-ranking $\varphi$ of Fig. 1(a) are drawn in ovals in Fig. 2.



(a) Supercritical 2-ranking $\varphi$ (b) Subcritical 2-ranking $\eta$

Figure 1: Two optimal 2-rankings of a tree $T$.

The problem of finding an optimal $c$-ranking of a graph $G$ has applications in scheduling the parallel assembly of a complex multi-part product from its components, where the vertices of $G$ correspond to the parts and the edges correspond to assembly operations. Let us consider a robot with $c+1$ hands which can connect at most $c+1$ components at a time. If we have as many robots as we need, then the problem of minimizing the number of steps required for the parallel assembly of a product using the robots is equivalent to finding an optimal $c$-ranking of the graph $G$. Fig. 2 shows that one can assemble in parallel a product of Fig. 1(a) in four steps using seven robots of three hands. Note that there are seven connected components which are not an isolated vertex in Step 1. In each step robots simultaneously connect at most three connected components of the previous step.

The $c$-edge-ranking problem is also equivalent to finding a $c$-edge-separator tree of minimum height. Consider the process of starting with a connected graph and partitioning it recursively

[1] The algorithms in [4] and [5] contain flaws, and the algorithm in [6] corrected them.

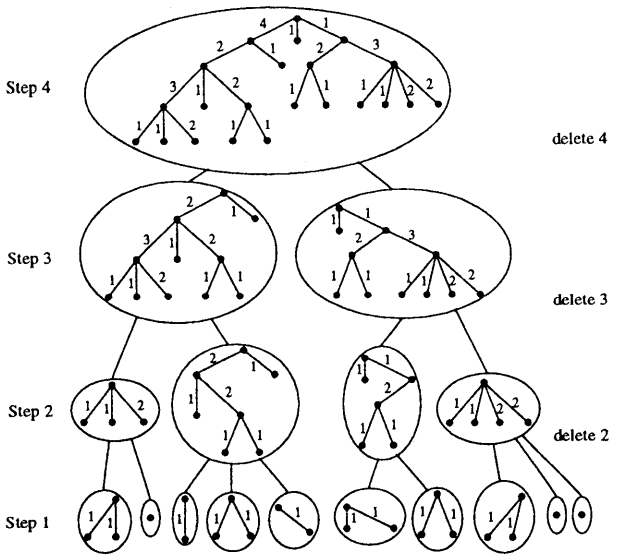



Figure 2: A 2-edge-separator tree of the tree in Fig. 1(a).

by removing at most c edges from each of the remaining connected components until the graph has no edge. The tree representing the recursive decomposition is called a *c-edge separator tree*. Thus a $c$-edge-separator tree corresponds to a parallel computation scheme based on the process above. Fig. 2 illustrates a 2-edge-separator tree of the tree $T$ depicted in Fig. 1(a).

In this paper we give an algorithm to solve the $c$-edge-ranking problem on trees. It is the first polynomial-time algorithm, and takes time $O(n^2 \log \Delta)$ for any positive integer $c$. Thus it is as efficient and simple as the best algorithm known for the ordinary edge-ranking problem [6].

## 2 Preliminaries

In this section we define some terms and present easy observations.

Let $G = (V, E)$ denote a graph with vertex set $V$ and edge set $E$. We often denote by $V(G)$ and $E(G)$ the vertex set and the edge set of $G$, respectively. We denote by $n$ the number of vertices in a tree $T = (V, E)$ and by $\Delta$ the maximum vertex-degree of $T$. $T$ is a "free tree," but we regard $T$ as a "rooted tree" for convenience sake: an arbitrary vertex of $T$ is designated as the *root* of $T$. The maximal subtree of $T$ rooted at a vertex $w \in V$ is denoted by $T(w)$. Let $e = (v, w)$ be an edge in $T$ such that $w$ is a child of $v$. Then the tree obtained from $T(w)$ by adding $e$ is denoted by $T(e)$. We denote by $T - T(e)$ the tree obtained from $T$ by deleting all edges and all vertices of $T(e)$ except $v$. We will use notions as: root, internal vertex, child and leaf in their usual meaning.

Let $\varphi$ be an edge-labeling of a tree $T$ with positive integers. The label (rank) of an edge $e \in E$ is represented by $\varphi(e)$. An edge $e$ of $T$ is *visible from vertex $w$ under $\varphi$* if $e$ is contained in $T(w)$ and every edge in the path from $w$ to $e$ has a rank $\leq \varphi(e)$. An edge visible from the root of $T$ is often called a *visible edge* for short. In Fig. 1 all visible edges are drawn in thick lines. For a subtree $T'$ of $T$, we denote by $\varphi|T'$ a restriction of $\varphi$ to $E(T')$: let $\varphi' = \varphi|T'$, then $\varphi'(e) = \varphi(e)$ for $e \in E(T')$. Then one can easily observe that the following lemma holds.

**Lemma 2.1** *An edge-labeling $\varphi$ of a tree $T$ is a $c$-ranking of $T$ if and only if*

(a) *$\varphi|T(w)$ is a $c$-ranking of $T(w)$ for every child $w$ of the root of $T$; and*

(b) *no more than $c$ edges of the same rank are visible from the root under $\varphi$.* □

Since it is not so simple to find an optimal $c$-ranking, we focus on specific types of optimal $c$-rankings. Before formally defining them, we need to define some terms.

One may assume without loss of generality that a $c$-ranking $\varphi$ of tree $T$ uses consecutive positive integers starting from 1 as the ranks. Thus the largest rank is equal to the number of ranks used by $\varphi$. The *list $L(\varphi)$ of a $c$-ranking $\varphi$ of tree $T$* is a list containing the ranks of all edges visible from the root, that is,

$$L(\varphi) = \{\varphi(e) \mid e \in E \text{ is visible from the root}\}.$$

The ranks in the list $L(\varphi)$ are sorted in non-increasing order. The list $L(\varphi)$ may contain same ranks with repetition $\leq c$. For an integer $l$ we denote by $count(L(\varphi), l)$ the number of $l$'s contained in $L(\varphi)$, that is, the number of visible edges with rank $l$. By Lemma 2.1 $count(L(\varphi), l) \leq c$ for each rank $l$. The $c$-ranking $\varphi$ in Fig. 1(a) has the list $L(\varphi) = \{4, 3, 2, 1, 1\}$, and hence $count(L(\varphi), 4) = count(L(\varphi), 3) = count(L(\varphi), 2) = 1$ and $count(L(\varphi), 1) = 2$. On the other hand, the $c$-ranking $\eta$ in Fig. 1(b) has the list $L(\eta) = \{4, 3, 3, 1\}$.

For a list $L$ and an integer $i$, we define a sublist $[i \leq L]$ of $L$ as follows:

$$[i \leq L] = \{l \in L \mid i \leq l\}.$$

Similarly we define sublists $[i < L]$, $[L \leq i]$ and $[L < i]$ of $L$. For lists $L$ and $L'$ we use $L \subseteq L'$ and $L \cup L'$ in their usual meaning in which we regard $L$, $L'$ and $L \cup L'$ as multi-sets. We define the *lexicographical order* $\prec$ on $L$ and $L'$ as usual, and write $L \preceq L'$ if either $L = L'$ or $L \prec L'$.

We are now ready to define notions of optimality. A *critical* $c$-ranking $\varphi$ of tree $T$ is defined to be a $c$-ranking with the lexicographically smallest list $L(\varphi)$. Every critical $c$-ranking $\varphi$ is optimal, because all edges of the largest rank are visible and hence the topmost rank in $L(\varphi)$ is equal to the number of ranks used by $\varphi$. The optimal $c$-ranking $\varphi$ depicted in Fig. 1(a) is indeed critical, but the optimal $c$-ranking $\eta$ in Fig. 1(b) is not critical since $L(\varphi) \prec L(\eta)$. A $c$-ranking $\varphi$ of tree $T$ is *supercritical* if the restriction $\varphi|T(v)$ is critical for every vertex $v$ of $T$. Thus a supercritical $c$-ranking is critical and hence optimal. The $c$-ranking $\varphi$ depicted in Fig. 1(a) is indeed supercritical. A $c$-ranking $\varphi$ of tree $T$ is *subcritical* if $\varphi|T(v)$ is critical for every vertex $v$ of $T$ except the root. Although a supercritical $c$-ranking is subcritical, a subcritical $c$-ranking is not always critical and is not always optimal. For example, the optimal $c$-ranking $\eta$ in Fig. 1(b) is subcritical, but is not critical.

# 3 Optimal $c$-ranking

The main result of this paper is the following theorem.

**Theorem 3.1** *For any positive integer $c$ an optimal $c$-ranking of a tree $T$ can be found in time $O(n^2 \log_2 \Delta)$, where $n$ is the number of vertices in $T$ and $\Delta$ is the maximum vertex-degree of $T$.* □

In the remaining of this section we give an algorithm to find a supercritical $c$-ranking of a tree $T$ in time $O(n^2 \log \Delta)$. Our algorithm uses the technique of "bottom-up tree computation." That is, it repeats the following operation for each internal vertex $v$ of a tree $T$ from leaves to the root: constructs a supercritical $c$-ranking $\varphi$ of subtree $T(v)$ from those $\varphi_1, \varphi_2, \cdots, \varphi_d$ of the subtrees $T(w_1), T(w_2), \cdots, T(w_d)$. For such a strategy to succeed, it is favorable that

a supercritical $c$-ranking $\varphi$ of $T(v)$ could be extended from $\varphi_1, \varphi_2, \cdots, \varphi_d$ by appropriately labeling the edges $e_i = (v, w_i)$, $1 \leq i \leq d$, without changing the labeling of the subtrees.

For notational convenience we may assume that $v$ is the root of tree $T$ and hence $T(v) = T$. We first have the following lemma.

**Lemma 3.2** *Every tree $T$ has a supercritical $c$-ranking $\varphi$. Furthermore for any supercritical $c$-rankings $\varphi_i$ of $T(w_i)$, $1 \leq i \leq d$, $T$ has a supercritical $c$-ranking $\varphi$ which is an extension of $\varphi_i$, that is, $\varphi|T(w_i) = \varphi_i$ for every $i$, $1 \leq i \leq d$.* □

In the remaining of this section we present a sequence of lemmas and corollaries which help us to decide the $d$ ranks in time $O(dn\log(d+1))$. Repeating the same operation for all internal vertices $v$ of $T$ from bottom to top, we can find a supercritical $c$-ranking of $T$ in time

$$O\left(\sum_{v \in V} d(v)n\log(d(v)+1)\right) = O(n^2 \log \Delta),$$

where $d(v)$ is the number of children of $v$ in $T$. Note that $\sum_{v \in V} d(v) = n - 1$.

The key idea of our algorithm is to decide the $d$ ranks in non-increasing order. Let $E(v) = \{e_i = (v, w_i) \mid 1 \leq i \leq d\}$. For a $c$-ranking $\varphi$ of tree $T$, let $m_\varphi$ be the maximum rank of edges in $E(v)$, that is, $m_\varphi = \max\{\varphi(e_i) \mid e_i \in E(v)\}$. Let $n_\varphi$ be the number of edges in $E(v)$ labeled by $m_\varphi$. We then have the following lemma.

**Lemma 3.3** *Let $\varphi$ and $\eta$ be two subcritical $c$-rankings of tree $T$. If $(m_\varphi, n_\varphi) \prec (m_\eta, n_\eta)$, then $L(\varphi) \prec L(\eta)$.* □

The following corollary is an immediate consequence of Lemma 3.3.

**Corollary 3.4** *The following* (a) *and* (b) *hold:*

(a) *if $\varphi$ is a supercritical $c$-ranking of tree $T$ and $\eta$ is a subcritical $c$-ranking of $T$, then $(m_\varphi, n_\varphi) \preceq (m_\eta, n_\eta)$; and*

(b) *every supercritical $c$-ranking $\varphi$ of $T$ has the same pair $(m_\varphi, n_\varphi)$.* □

We thus denote by $\beta_{\text{sup}}$ the same value $m_\varphi$ for all supercritical $c$-rankings $\varphi$ of $T$, and call $\beta_{\text{sup}}$ *the super rank of tree $T$*. Then Corollary 3.4 immediately implies the following.

**Corollary 3.5** *The super rank $\beta_{\text{sup}}$ of tree $T$ is equal to the minimum integer $\beta$ for which $T$ has a subcritical $c$-ranking $\varphi$ of $m_\varphi = \beta$.* □

Let $\beta$ be a positive integer, and let $L_i = L(\varphi_i)$. Let $j$, $1 \leq j \leq d$, be an index such that sublist $[L_j < \beta]$ is the lexicographically largest among all $[L_i < \beta]$, $1 \leq i \leq d$, that is, if $e_j$ was labeled by $\beta$ then the sublist of ranks in $L_j$ hidden by $\beta$ would be lexicographically largest. Then the following lemma holds.

**Lemma 3.6** *Let $j$ be the index defined for $\beta$ as above, and assume that tree $T$ has a subcritical $c$-ranking $\eta$ of $m_\eta = \beta$. Then $T$ has a subcritical $c$-ranking $\varphi$ such that $\varphi(e_j) = m_\varphi = \beta$ and $L(\varphi) \preceq L(\eta)$.* □

The following corollary is an immediate consequence of Lemma 3.6.

**Corollary 3.7** *Let $j$ be the index defined for $\beta = \beta_{\text{sup}}$, that is, let $[L_j < \beta_{\text{sup}}]$ be the lexicographically largest one among all sublists $[L_i < \beta_{\text{sup}}]$, $1 \leq i \leq d$. Then tree $T$ has a supercritical $c$-ranking $\varphi$ such that $\varphi(e_j) = \beta_{\text{sup}}$.* □

Thus, once $\beta_{\sup}$ is decided, one can easily decide an edge $e_j$ to be labeled by $\beta_{\sup}$. Furthermore we have the following lemma.

**Lemma 3.8** *Let $j$ be the index defined for $\beta = \beta_{\sup}$, and let $T' = T - T(e_j)$. Then any supercritical c-ranking $\varphi'$ of $T'$ can be extended to a supercritical c-ranking $\varphi$ of $T$ as follows:*

$$\varphi(e) = \begin{cases} \beta_{\sup} & \text{if } e = e_j; \\ \varphi_j(e) & \text{if } e \in E(T(w_j)); \text{ and} \\ \varphi'(e) & \text{if } e \in E(T'). \end{cases}$$

Using Corollary 3.7 and Lemma 3.8 one can easily verify that the following algorithm UPDATE correctly decides the ranks of $e_1, e_2, \cdots, e_d$ if algorithm SUPER-RANK, given later, could correctly find the super rank of trees.

```
Procedure UPDATE(v);
begin
1  T' := T(v);
2  for d := d(v) downto 1 do        { decide the d(v) ranks in non-increasing order,
                                      where d(v) is the number of children of v in T }
     begin
3      let w_{i_1}, w_{i_2}, ..., w_{i_d} be the children of v in T';
4      find the super rank β_sup of tree T' by the algorithm SUPER-RANK;
                                                  { SUPER-RANK will be given later }
5      find an index i_j, 1 ≤ j ≤ d, such that [L_{i_j} < β_sup] is the lexicographically
         largest one among the d lists [L_{i_k} < β_sup], 1 ≤ k ≤ d;
6      label edge e_{i_j} = (v, w_{i_j}) with β_sup;          { cf. Corollary 3.7 }
7      T' := T' - T'(e_{i_j});                                { cf. Lemma 3.8 }
     end
end;
```

Clearly line 1 can be done in time $O(1)$. Lines 3–7 are executed $d = d(v)$ times. One execution of lines 3 and 5–7 can be done in time $O(n)$. Therefore if one execution of line 4, i.e., SUPER-RANK, takes time $O(n \log(d+1))$, then the algorithm UPDATE runs in time $O(dn \log(d+1))$ as we claim. Thus it suffices to give the algorithm SUPER-RANK for finding $\beta_{\sup}$ of a tree in time $O(n \log(d+1))$.

By Corollary 3.5, in order to find $\beta_{\sup}$ we need to check the existence of a subcritical c-ranking $\varphi$ with $m_\varphi = \beta$ for some integers $\beta$. A necessary and sufficient condition for the existence is given in Lemma 3.9 below.

**Lemma 3.9** *Let $1 \le c' \le c$, and let $j$ be the index defined for $\beta$ as before. Tree $T$ has a subcritical c-ranking $\varphi$ such that $\varphi(e_j) = m_\varphi = \beta$ and $c_\varphi \le c'$ if and only if the following three conditions hold:*

(a) $count(\cup_{i=1}^{d} L_i, \beta) \le c' - 1$;

(b) $count(\cup_{i=1}^{d} L_i, \gamma) \le c$ *for all ranks* $\gamma \in [\beta < L_j]$; *and*

(c) *if $d \ge 2$, then tree $T' = T - T(e_j)$ has a subcritical c-ranking $\varphi'$ such that*

(i) *if $count(\cup_{i=1}^{d} L_i, \beta) \le c' - 2$, then $m_{\varphi'} = \beta$ and $c_{\varphi'} \le c''$ where $c'' = c' - n_j - 1$ and $n_j = count(L_j, \beta)$; and*

(ii) *if $count(\cup_{i=1}^{d} L_i, \beta) = c' - 1$, then there is an integer $\beta'$, $1 \le \beta' \le \beta - 1$, such that $count(\cup_{\substack{i=1 \\ i \ne j}}^{d} L_i, \beta') \le c - 1$, and furthermore $m_{\varphi'} = \beta'$ for the largest $\beta'$ among all such integers.* □

Let $B = \{\gamma \mid 1 \leq \gamma \leq n-1 \text{ and } count(\cup_{i=1}^{d} L_i, \gamma) \leq c-1\}$. Then $T$ necessarily has a subcritical $c$-ranking $\varphi$ such that $m_\varphi = \beta$ for some integer $\beta \in B$. Thus $B$ is a set of integers eligible for $\beta_{\sup}$. From Lemma 3.9 one can easily derive the following recursive algorithm CHECK to decide whether $T$ has a subcritical $c$-ranking $\varphi$ of $m_\varphi = \beta$ and $c_\varphi \leq c'$ for a given integer $\beta \in B$ by checking the conditions (a), (b) and (c) in Lemma 3.9.

```
   Procedure CHECK(T, β, c');
   begin
 8  if count(∪_{i=1}^{d} L_i, β) > c' − 1
      then return false                                  { the condition (a) does not hold}
      else begin                                              {the condition (a) holds}
 9      let w_1, w_2, ···, w_d be the children of v in T;                          {d ≥ 1}
10      let j, 1 ≤ j ≤ d, be an index for which [L_j < β] is the lexicographically
            largest among all [L_k < β], 1 ≤ k ≤ d;
11      if count(∪_{i=1}^{d} L_i, γ) > c for a rank γ ∈ [β < L_j]
          then return false                              {the condition (b) does not hold}
          else                                                {the condition (b) holds}
12          if the root v of T has exactly one child     {obviously the condition (c) holds}
              then return true
              else begin                                                      {d ≥ 2}
13              T' := T − T(e_j);
14              if count(∪_{i=1}^{d} L_i, β) ≤ c' − 2 then begin
15                n_j := count(L_j, β);
16                c'' := c' − n_j − 1;
                  CHECK(T', β, c'')   {check the condition (c) by recursively calling CHECK}
                end else                                    {count(∪_{i=1}^{d} L_i, β) = c' − 1}
17                if there is no integer β', 1 ≤ β' ≤ β − 1, such that count(∪_{i=1, i≠j}^{d} L_i, β') ≤ c − 1
                  then return false                    {the condition (c) does not hold}
                  else begin
18                  let β', 1 ≤ β' ≤ β − 1, be the largest one among all integers
                        such that count(∪_{i=1, i≠j}^{d} L_i, β') ≤ c − 1;
                    CHECK(T', β', c)  {check the condition (c) by recursively calling CHECK}
                  end
              end
      end
   end;
```

By Corollary 3.5 and Lemma 3.9, $\beta_{\sup}$ is the smallest integer $\beta$ satisfying the conditions (a), (b) and (c) for $c' = c$. Therefore we have the following algorithm SUPER-RANK to find $\beta_{\sup}$ of tree $T$.

```
   Procedure SUPER-RANK(T);
   begin
19  B := {γ | 1 ≤ γ ≤ n − 1 and count(∪_{i=1}^{d} L_i, γ) ≤ c − 1};
20  choose the smallest integer β ∈ B satisfying the conditions (a), (b) and (c)
      by executing CHECK(T, β, c) for all β ∈ B;
21  β_sup := β
   end;
```

We can prove the following lemma.

**Lemma 3.10** *The algorithm CHECK takes $O(n)$ time.* □

By Lemma 3.10 the straightforward implementation of SUPER-RANK takes $O(n^2)$ time since $|B| \leq n-1$. We will improve the time-complexity $O(n^2)$ to $O(n\log(d+1))$. Our idea is to use the binary search.

If $count(\cup_{i=1}^{d} L_i, \gamma) \leq c$ for all ranks $\gamma \in \cup_{i=1}^{d} L_i$, let $\alpha = 0$. Otherwise, let $\alpha$ be the largest rank $\gamma$ such that $count(\cup_{i=1}^{d} L_i, \gamma) \geq c+1$. Then $\beta_{\sup} \geq \alpha + 1$. Let $\beta_1 < \beta_2 < \cdots < \beta_d$ be the smallest $d$ integers such that $\beta_i \geq \alpha + 1$ and $count(\cup_{i=1}^{d} L_i, \beta_i) \leq c-1$. We then have the following lemma by Corollary 3.5 and Lemma 3.9.

**Lemma 3.11** *The following* (a) *and* (b) *hold:*

(a) $\beta_{\sup} \in \{\beta_1, \beta_2, \cdots, \beta_d\}$; *and*

(b) *let $i$ be an integer such that $1 \leq i \leq d$, then $\beta_{\sup} \in \{\beta_1, \beta_2, \cdots, \beta_i\}$ if and only if $T$ has a subcritical $c$-ranking $\eta$ of $m_\eta = \beta_i$.* □

Replace line 19 of Procedure SUPER-RANK with the following:

$$B := \{\beta_1, \beta_2, \cdots, \beta_d\}.$$

Then, using the binary search technique, one can find smallest integer $\beta$ at line 20 by calling CHECK at most $\lceil \log(d+1) \rceil$ times. Clearly line 19 can be done in time $O(n)$, and line 21 can be done in time $O(1)$. Therefore the algorithm SUPER-RANK can be done in time $O(n\log(d+1))$.

# 4 Conclusion

We newly define a generalized edge-ranking of a graph, called a $c$-ranking, and give an algorithm for finding an optimal $c$-ranking of a given tree. This is the first polynomial-time algorithm for the generalized edge-ranking problem, and is as simple and efficient as the best algorithm known for the ordinary edge-ranking problem [6].

# References


[1] A. V. Iyer, H. D. Ratliff, and G. Vijayan, On an edge-ranking problem of trees and graphs, *Discrete Applied Mathematics*, **30** (1991), pp. 43–52.

[2] P. de la Torre, R. Greenlaw, and A. A. Schäffer, Optimal ranking of trees in polynomial time, *Proc. of the 4th Annual ACM-SIAM Symp. on Discrete Algorithms*, 1993, pp. 138–144, also to appear in *Algorithmica.*

[3] X. Zhou, H. Nagai, and T. Nishizeki, Generalized rankings of trees, *Technical Report of SIGAL 94-AL-42, Information Processing Society of Japan*, 1994, pp. 79–86.

[4] X. Zhou and T. Nishizeki, An efficient algorithm for edge-ranking trees, *Proc. of the 2nd. European Symp. on Algorithms, Lecture Notes in Computer Science, Springer-Verlag*, **855** (1994), pp. 118–129.

[5] X. Zhou and T. Nishizeki, Finding optimal edge-rankings of trees, *Proc. of the 6th Annual ACM-SIAM Symp. on Discrete Algorithms*, 1995, pp. 122–131.

[6] X. Zhou and T. Nishizeki, Finding optimal edge-rankings of trees – A correct algorithm, *Technical Report 9501, Dept. of Information Engg., Tohoku Univ.*, 1995.