部分 k-木の一般化点ランキング

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

グラフ G の全ての点に正整数のラベルを付ける。各ラベル i について、i より大きなラベルの点を全て G から除去したとき、各連結成分がラベル i の点を高々c 個しかもたないならば、このラベル付けを c-点ランク付けという。ここで c は正整数とする。最小の個数のラベルで部分 k木を c-点ランク付けする多項式時間アルゴリズムを本文は与える。ここで kは定数とする。1-点ランク付けは単に点ランク付けと呼ばれるが、本文のアルゴリズムは、1-点ランク付けを求める既知のアルゴリズムより高速である。

Generalized Vertex-Rankings of Partial k-Trees

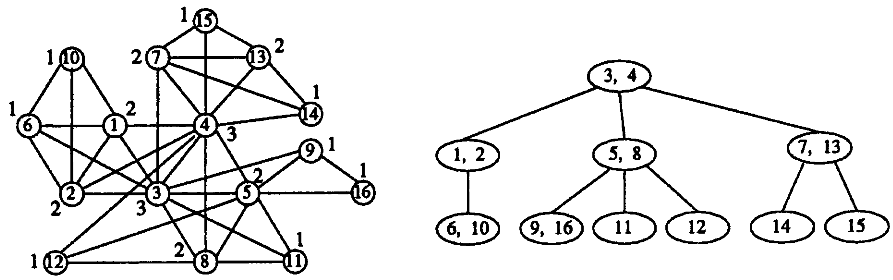
Md. Abul Kashem†, Xiao Zhou‡ and Takao Nishizeki†
†Graduate School of Information Sciences
‡Education Center for Information Processing
Tohoku University, Sendai 980-77, Japan.
E-mail: kashem@nishizeki.ecei.tohoku.ac.jp, zhou@ecip.tohoku.ac.jp, nishi@ecei.tohoku.ac.jp

Abstract. A c-vertex-ranking of a graph G for a positive integer c 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 c vertices with label i. We present a polynomial time algorithm to find a c-vertex-ranking of a partial k-tree using the minimum number of ranks for any positive integer c and any bounded integer k. Our algorithm is faster than the best algorithm known for the ordinary vertex-ranking, that is, 1-vertex-ranking.

Key words: Algorithm, Partial k-tree, Separator tree, Treewidth, Vertex-ranking.

1 Introduction

An ordinary *vertex-ranking* of a graph G is a labeling (ranking) of vertices of G with positive integers such that every path between two vertices with the same label i contains a vertex with label $j > i$ [6]. Clearly a vertex-labeling is a vertex-ranking if and only if, for any label i, deletion of all vertices with labels $> i$ leaves connected components, each having at most one vertex with label i. The integer label of a vertex is called the *rank* of the vertex. The *vertex-ranking problem* is to find a vertex-ranking of a given graph G using the minimum number of ranks. The vertex-ranking problem has applications in VLSI layout and in scheduling the parallel assembly of a complex multi-part product from its components [6]. The vertex-ranking problem is NP-hard in general [3, 10], while Iyer *et al.* presented an $O(n \log n)$ time algorithm to solve the vertex-ranking problem for trees [6]. Then Schäffer obtained a linear-time algorithm by refining their algorithm and its analysis [12]. Recently Deogun *et*



(a) (b)

Figure 1: (a) An optimal 2-ranking of a graph G and (b) a 2-vertex-separator tree of G.

al. gave algorithms to solve the vertex-ranking problem for interval graphs in $O(n^3)$ time and for permutation graphs in $O(n^6)$ time [5]. Very recently Bodlaender *et al.* have given a polynomial-time algorithm to solve the vertex-ranking problem for partial k-trees of bounded treewidth k [3].

A natural generalization of an ordinary vertex-ranking is the c-vertex-ranking [14]. 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, deletion of all vertices with labels $> i$ leaves connected components, each having at most c vertices with label i. Clearly an ordinary vertex-ranking is a 1-vertex-ranking. 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 is 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 [3, 10]. Zhou *et al.* have obtained a linear-time algorithm to solve the c-ranking problem for trees [14]. Fig. 1(a) depicts an optimal 2-ranking of a graph G using three ranks, where vertex numbers are drawn in circles and ranks next to the circles.

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

Let M be a sparse symmetric matrix, and let M' be a matrix obtained from M by replacing each nonzero element with 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 [9].

The edge-ranking problem [7] and the c-edge-ranking problem [13] for a graph G are defined similarly. The edge-ranking problem, that is, 1-edge-ranking problem, is NP-hard in general [8], while de la Torre *et al.* obtained an algorithm to solve the edge-ranking problem for trees in $O(n^3 \log n)$ time [4]. On the other hand, Zhou *et al.* gave an $O(n^2 \log \Delta)$ time algorithm to solve the c-edge-ranking problem on trees for any positive integer c, where Δ is the maximum degree of T [13].

In this paper we give a polynomial-time algorithm to solve the c-vertex-ranking problem on partial k-trees of bounded treewidth k for any positive integer c. It is the first polynomial-time algorithm for the generalized vertex-ranking problem on partial k-trees, and is faster than the best algorithm known for the ordinary vertex-ranking problem [3]. The result in this paper implies a polynomial-time algorithm of the generalized vertex-ranking problem for any class of graphs with a uniform upper bound on the treewidth, e.g., series-parallel graphs, outerplanar graphs, k-outerplanar graphs, Halin graphs, graphs with bandwidth $\leq k$, graphs with cutwidth $\leq k$, chordal graphs with maximum clique-size k, graphs that do not contain some fixed planar graph as a minor [1].

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 G. An edge joining vertices u and v is denoted by (u, v). We will use notions as: *leaf, node, child, father* and *root* in their usual meaning.

A natural generalization of ordinary trees is the so-called k-trees. The class of *k-trees* is defined recursively as follows [1]:

(a) A complete graph with k vertices is a k-tree.

(b) If $G = (V, E)$ is a k-tree and k vertices $v_1, v_2, \cdots, v_k$ induce a complete subgraph of G, then $G' = (V \cup \{w\}, E \cup \{(v_i, w) \mid 1 \leq i \leq k\})$ is a k-tree, where w is a new vertex not contained in G.

(c) All k-trees can be formed with rules (a) and (b).

A graph is called a *partial k-tree* if it is a subgraph of a k-tree. Thus a partial k-tree $G = (V, E)$ is a *simple* graph without multiple edges or self-loops, and $|E| < kn$. In this paper we assume that k is a constant. Figure 2(a) illustrates a process of generating a 3-tree, and Fig. 2(b) depicts a partial 3-tree.

A *tree-decomposition* of a graph $G = (V, E)$ is a pair (T, S) with $T = (V_T, E_T)$ a tree and $S = \{X_x \mid x \in V_T\}$ a collection of subsets of V satisfying the following three conditions [11]:

- $\bigcup_{x \in V_T} X_x = V$;
- for every edge $e = (v, w) \in E$, there exits a node $x \in V_T$ with $v, w \in X_x$; and
- for all $x, y, z \in V_T$, if node y lies on the path from node x to node z in T, then $X_x \cap X_z \subseteq X_y$.

Figure 2(c) depicts a tree-decomposition of the partial 3-tree in Fig. 2(b). The *width of a tree-decomposition* (T, S) is $\max_{x \in V_T} |X_x| - 1$. The *treewidth of a graph* G is the minimum width of a tree-decomposition of G, taken over all possible tree-decompositions of G. It is known that every graph with treewidth $\leq k$ is a partial k-tree, and conversely, that every partial k-tree has a tree-decomposition with width $\leq k$. For any fixed k, determining whether the treewidth of G is at most k and finding a corresponding tree-decomposition can be done in $O(n)$ time [2].

Consider a tree-decomposition (T, S) of G with width $\leq k$. We transform it to a *binary tree-decomposition* as follows [1]: regard T as a rooted tree by choosing an arbitrary node as the root, and replace every internal node x with d children, say $y_1, y_2, \cdots, y_d$, by $d + 1$ new nodes $x_1, x_2, \cdots, x_{d+1}$ such that $X_x = X_{x_1} = X_{x_2} = \cdots = X_{x_{d+1}}$, where x_1 has the same father as x, x_i is the father of x_{i+1} and the ith child y_i, $1 \leq i \leq d$, of x, and x_{d+1} is a leaf of the tree. This transformation can be done in $O(n)$ time. The resulted tree-decomposition (T, S) of $G = (V, E)$ has the following characteristics:

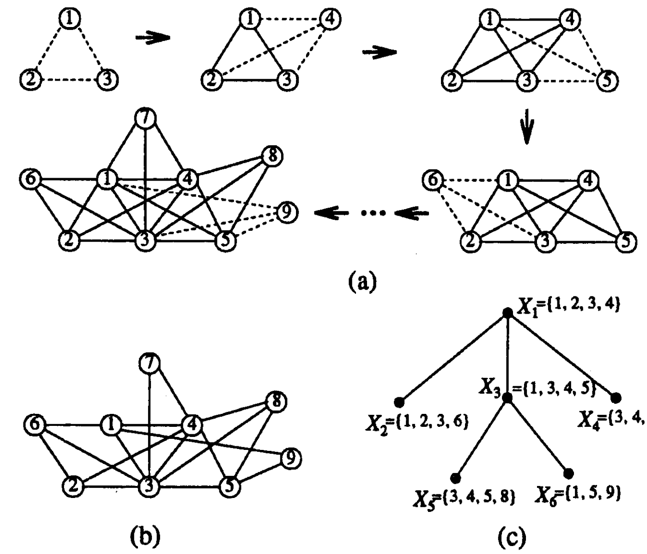


Figure 2: (a) 3-trees, (b) a partial 3-tree and (c) a tree-decomposition of the partial 3-tree.

- the width of (T, S) is $\leq k$, and the number of nodes in T is $O(n)$;
- each internal node x of T has exactly two children, say y and z, and either $X_x = X_y$ or $X_x = X_z$; and
- for each edge $e = (v, w) \in E$, there is at least one leaf y in T such that $v, w \in X_y$.

Consider a rooted binary tree-decomposition (T, S) of a partial k-tree G. To each node x of the rooted tree T, we associate a subgraph $G_x = (V_x, E_x)$ of G, where $V_x = \bigcup\{X_y \mid y = x$ or y is a descendant of x in $T\}$ and $E_x = \{(v, w) \in E \mid v, w \in V_x\}$. Thus the root of T corresponds to G.

Let φ be a vertex-labeling of a partial k-tree G with positive integers. The label (rank) of a vertex $v \in V$ is denoted by $\varphi(v)$. The number of ranks used by a vertex-labeling φ is denoted by $\#\varphi$. One may assume without loss of generality that φ uses consecutive integers $1, 2, \cdots, \#\varphi$ as the ranks. For a subgraph $G_x = (V_x, E_x)$ of G, $x \in V_T$, we denote by $\varphi|G_x$ or simply by φ_x a restriction of φ to G_x: $\varphi_x(v) = \varphi(v)$ for $v \in V_x$. A vertex $w \in V_x$ is said to be *visible* from a vertex $v \in X_x$ under φ in G_x if G_x has a path from v to w every vertex of which has a rank $\leq \varphi(w)$. Then the rank $\varphi(w)$ of w is also said to be *visible* from v under φ in G_x. Thus the smallest rank visible from the vertex v under φ in G_x is equal to $\varphi(v)$. We then have the following lemma which characterizes the c-ranking of a partial k-tree by the number of visible vertices.

Lemma 2.1 *Let (T, S) be a rooted binary tree-decomposition of a partial k-tree G. Let an internal node x in T have two children y and z. Then a vertex-labeling φ of G_x is a c-ranking of G_x if and only if*

(a) *$\varphi|G_y$ and $\varphi|G_z$ are c-rankings of G_y and G_z, respectively; and*

(b) *at most c vertices of the same rank are visible from any vertex $v \in X_x$ under φ in G_x.* □

We next show that the number of ranks needed for an optimal c-ranking of a partial k-tree is at most $\left\lceil\frac{k+1}{c}\right\rceil(1+\log n)$. We first cite the following lemma from [11].

Lemma 2.2 *Let $G=(V,E)$ be a partial k-tree of n vertices. Then there exists $X \subseteq V$ with $|X| \leq k+1$ such that every connected component of $G-X$ contains at most $\frac{1}{2}|V-X| \leq \frac{n}{2}$ vertices.* □

We then have the following lemma.

Lemma 2.3 *Every partial k-tree G of n vertices satisfies $r_c(G) \leq \left\lceil\frac{k+1}{c}\right\rceil(1+\log n)$.* □

3 Optimal c-ranking

The main result of this paper is the following theorem.

Theorem 3.1 *For any positive integer c and any bounded integer k, an optimal c-ranking of a partial k-tree G can be found in time $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil\log(c+1)+2}\cdot\log^{k(k+1)+1} n\cdot\log\log n)$, where n is the number of vertices in G.*

In the remaining of this section we give an algorithm to find an optimal c-ranking of a partial k-tree G in time $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil\log(c+1)+2}\cdot\log^{k(k+1)+1} n\cdot\log\log n)$. Let (T,S) be a binary tree-decomposition of G. The first step of our algorithm is to decide, for a positive integer m, whether G has a c-ranking φ with $\#\varphi \leq m$ by means of dynamic programming and bottom-up tree computation on the binary tree T: for each node x of T from leaves to the root, we construct all (equivalence classes of) c-rankings of G_x from those of two subgraphs G_y and G_z associated with the children y and z of x. Then, by using a binary search over the range of m, $1 \leq m \leq \left\lceil\frac{k+1}{c}\right\rceil(1+\log n)$, we determine the minimum value of m such that G has a c-ranking φ with $m=\#\varphi=r_c(G)$, and find an optimal c-ranking of G.

Many algorithms on partial k-trees use dynamic programming. For each node x of T, a table of all possible partial solutions of the problem is computed, where each entry in the table represents an equivalence class. The time-complexity of an algorithm mainly depends on the size of the table. Therefore, we find a suitable equivalence class for which the table has a polynomial size. For the ordinary vertex-ranking problem, Bodlaender *et al.* [3] defined an equivalence class for which the size of the table on each node is $O(n^{(k+1)^2(k+2)}\log^{(k+1)(k+2)/2} n)$, while the size of the table of our algorithm for the c-ranking problem is $O(n^{(k+1)\lceil\frac{k+1}{c}\rceil\log(c+1)}\log^{k(k+1)/2} n)$. This size of equivalence classes is one of the key ideas behind the speed-up of our algorithm over the best algorithm known for the ordinary vertex-ranking [3]. Before defining the equivalence class, we need to define some terms.

Let $R=\{1,2,\cdots,m\}$ be the set of ranks. Then a mapping (vertex-labeling) $\varphi: V \rightarrow R$ is a c-ranking of a graph G if and only if for any rank $i \in R$, deletion of all vertices with ranks $> i$ leaves connected components, each having at most c vertices with rank i. A c-ranking of G_x, $x \in V_T$, is defined to be *extensible* if it can be extended to a c-ranking of G.

Let $\varphi: V_x \rightarrow R$ be a vertex-labeling of $G_x=(V_x,E_x)$. Then more than one vertices with the same rank may be visible from a vertex $v \in X_x$ under φ in G_x. For an integer i, we denote by $count(\varphi,v,i)$ the number of vertices ranked by i and visible from v under φ in G_x. If φ is a c-ranking of G_x, then by Lemma 2.1 $count(\varphi,v,i) \leq c$ for any vertex $v \in X_x$ and any integer $i \in R$. Iyer *et al.* introduced an idea of a "critical list" to solve the ordinary vertex-ranking problem for trees [6], while we define a *count-list* $L(\varphi,v)$ and a *list-set* $\mathcal{L}(\varphi)$ as follows:

$$\begin{aligned} L(\varphi,v) &= \{(i, count(\varphi,v,i)) \mid \text{ rank } i \text{ is visible from } v \text{ under } \varphi \text{ in } G_x\}; \text{ and} \\ \mathcal{L}(\varphi) &= \{L(\varphi,v) \mid v \in X_x\}. \end{aligned}$$

For the vertex-labeling φ of G_x, define a function $\lambda_\varphi : X_x \times X_x \rightarrow R \cup \{0, \infty\}$ as follows:

$$\lambda_\varphi(v, w) = \min\{\lambda \mid G_x \text{ has a path from } v \in X_x \text{ to } w \in X_x \text{ such that } \varphi(u) \leq \lambda \text{ for each internal vertex } u \text{ of the path }\}.$$

Let $\lambda_\varphi(v, w) = 0$ if $(v, w) \in E_x$ or $v = w$, and let $\lambda_\varphi(v, w) = \infty$ if G_x has no path from v to w. Clearly $\lambda_\varphi(v, w) = \lambda_\varphi(w, v)$. We next define a pair $\mathcal{R}(\varphi)$ as follows:

$$\mathcal{R}(\varphi) = (\mathcal{L}(\varphi), \lambda_\varphi).$$

We call such a pair $\mathcal{R}(\varphi)$ the *vector* of φ on node x. The vector $\mathcal{R}(\varphi)$ is called *feasible* if the vertex-labeling φ is a c-ranking of G_x. We then have the following lemma.

Lemma 3.2 *Let φ and η be two c-rankings of G_x such that $\mathcal{R}(\varphi) = \mathcal{R}(\eta)$. Then φ is extensible if and only if η is extensible.* □

Thus a feasible vector $\mathcal{R}(\varphi)$ of φ on x can be seen as an equivalence class of extensible c-rankings of G_x. Since $|R| = m$ and $0 \leq count(\varphi, v, i) \leq c$ for a c-ranking φ and a rank $i \in R$, the number of distinct count-lists $L(\varphi, v)$ is at most $(c+1)^m$ for each vertex $v \in X_x$. Furthermore $|X_x| \leq k+1$. Therefore the number of distinct list-sets $\mathcal{L}(\varphi)$ is at most $(c+1)^{(k+1)m}$. On the other hand, the number of distinct functions $\lambda_\varphi : X_x \times X_x \rightarrow R \cup \{0, \infty\}$ is at most $(m+2)^{k(k+1)/2}$, since $\lambda_\varphi(v, v) = 0$ and $\lambda_\varphi(v, w) = \lambda_\varphi(w, v)$ for any $v, w \in X_x$. Therefore the total number of different feasible vectors on node x is at most $(c+1)^{(k+1)m} \cdot (m+2)^{k(k+1)/2}$. By Lemma 2.3, one may assume that $m \leq \lceil \frac{k+1}{c} \rceil (1 + \log n)$. Thus the total number of different feasible vectors on x is $O(n^{(k+1)\lceil \frac{k+1}{c} \rceil \log(c+1)} \cdot \log^{k(k+1)/2} n)$ for fixed k.

The main step of our algorithm is to compute a table of all feasible vectors on the root of T by means of dynamic programming and bottom-up tree computation on T. If the table has at least one feasible vector, then the partial k-tree G corresponding to the root of T has a c-ranking φ such that $\#\varphi \leq m$.

For each leaf x of T, the table of all feasible vectors $\mathcal{R}(\varphi) = (\mathcal{L}(\varphi), \lambda_\varphi)$ on x can be computed in time $O(\log^{k+1} n)$ as follows:

(1) enumerate all vertex-labelings $\varphi : V_x \rightarrow R$; and

(2) compute all feasible vectors $\mathcal{R}(\varphi)$ on x from the vertex-labelings φ of G_x.

Since $|V_x| \leq k+1$ and $|R| = m$, the number of vertex-labelings $\varphi : V_x \rightarrow R$ is at most m^{k+1}. For each vertex-labeling φ, λ_φ can be computed in time $O(1)$. Futhermore, the count-lists $L(\varphi, v)$, $v \in X_x = V_x$, can be computed in time $O(1)$. Then checking whether a vertex-labeling φ is a c-ranking of G_x, and if so, computing $\mathcal{L}(\varphi)$ can be done in time $O(1)$. Therefore, steps (1) and (2) can be executed for a leaf in time $O(m^{k+1}) = O(\log^{k+1} n)$.

Next we show how to compute all feasible vectors $\mathcal{R}(\varphi) = (\mathcal{L}(\varphi), \lambda_\varphi)$ on an internal node x of T from those on two children y and z of x. One may assume that $X_x = X_y$. Therefore $V_x = V_y \cup V_z$. Let η and ψ be c-rankings of G_y and G_z such that $\eta(v) = \psi(v)$ for any vertex $v \in X_y \cap X_z$, and let φ be the vertex-labeling of G_x extended from η and ψ. Then $\varphi | G_y = \eta$ and $\varphi | G_z = \psi$.

We first compute λ_φ. Construct a graph $G(\eta)$ from λ_η and $\eta(v)$, $v \in X_y$, as follows: construct a complete graph of the vertices in X_y; assign a weight of $\eta(v)$ to each vertex $v \in X_y$ of the graph; place a dummy vertex on each edge (v, w) of the graph and assign a weight of $\lambda_\eta(v, w)$ to the dummy vertex. Then the total number of vertices in $G(\eta)$ is $\leq k+1+k(k+1)/2 = (k+1)(k+2)/2$ and the total number of edges is $\leq k(k+1)$. Similarly construct a graph $G(\psi)$. We now construct a graph $G'(\varphi)$ from $G(\eta)$ and $G(\psi)$ by identifying the vertices $v \in X_y \cap X_z$. Then the total number of vertices in $G'(\varphi)$ is $\leq (k+1)(k+2)$ and the total number of edges is $\leq 2k(k+1)$. Define a function $\lambda_{\eta\psi} : X_x \times X_x \to R \cup \{0, \infty\}$ as follows:

$$\lambda_{\eta\psi}(v, w) = \min\{\lambda \mid G'(\varphi) \text{ has a path from } v \in X_x \text{ to } w \in X_x \text{ such that all internal vertices of the path have weights } \leq \lambda\}.$$

Then by the construction of $G'(\varphi)$, $\lambda_{\varphi}(v, w) = \lambda_{\eta\psi}(v, w)$ for $v, w \in X_x$. Since $G'(\varphi)$ has a constant number of vertices and a constant number of edges, λ_{φ} can be computed by the help of $G'(\varphi)$ in time $O(1)$ for each vector $\mathcal{R}(\varphi)$.

We next show how to compute $\mathcal{L}(\varphi)$. Let v be any vertex in X_x. No vertex with a rank $i < \varphi(v)$ is visible from v under φ in G_x. Let $i \in R$ be any rank such that $i \geq \varphi(v)$. Delete all vertices with ranks $> i$ from G_x. Among the connected components of the resulting graph, let H be the one containing the vertex v. Clearly the number of vertices with rank i in H is equal to $count(\varphi, v, i)$. Since $|E_x| = O(n)$ and $|R| = m$, the count-lists $L(\varphi, v)$, $v \in X_x$, can be computed in time $O(n \cdot m) = O(n \log n)$. Then checking whether a vertex-labeling φ is a c-ranking of G_x, and if so, computing $\mathcal{L}(\varphi)$ can be done in time $O(n \log n)$.

Therefore each vector on an internal node can be computed in time $O(n \log n)$. The table of all feasible vectors on an internal node x can be obtained from the tables of all feasible vectors on the two children of x, and the number of different feasible vectors on any node of T is $O(n^{(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)} \cdot \log^{k(k+1)/2} n)$. Therefore the table on x can be computed in time $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)+1} \cdot \log^{k(k+1)+1} n)$.

A binary tree-decomposition (T, S) of a given partial k-tree G can be found in $O(n)$ time [2]. We then have an algorithm CHECK to determine whether G has a c-ranking φ with $\#\varphi \leq m$ for a positive integer m.

Algorithm CHECK;
begin
1 compute a table of all feasible vectors on each leaf x of T, and keep a c-ranking φ of G_x arbitrarily chosen from the c-rankings having the same feasible vector;
2 for each internal node x of T, compute a table of all feasible vectors from those on the two children of x, and keep a c-ranking φ of G_x arbitrarily chosen from the c-rankings having the same feasible vector;
3 repeat line 2 up to the root of T;
4 check whether there exists a feasible vector in the table at the root;
end.

Line 1 can be done in $O(\log^{k+1} n)$ time for each leaf as mentioned before, and hence line 1 can be done in $O(n \log^{k+1} n)$ time in total. As mentioned above, line 2 can be done in $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)+1} \cdot \log^{k(k+1)+1} n)$ time per node. Since line 2 is executed for $O(n)$ nodes in total in line 3, line 3 can be done in $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)+2} \cdot \log^{k(k+1)+1} n)$ time in total. Line 4 can be done in $O(n^{(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)} \cdot \log^{k(k+1)/2+1} n)$ time in total. Thus checking whether a partial k-tree G has a c-ranking φ such that $\#\varphi \leq m$ can be done in $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)+2} \cdot \log^{k(k+1)+1} n)$ time.

Using the binary search technique over the range of m, $1 \leq m \leq \left\lceil\frac{k+1}{c}\right\rceil (1 + \log n)$, one can find the smallest integer $r_c(G)$ such that G has a c-ranking φ with $\#\varphi = r_c(G)$ by calling CHECK $O(\log\log n)$ times. Therefore, an optimal c-ranking of a partial k-tree G of n vertices can be found in time $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil \log(c+1)+2} \cdot \log^{k(k+1)+1} n \cdot \log\log n)$ for any positive integer c and any bounded integer k.

4 Conclusion

We give a polynomial-time algorithm for finding an optimal c-ranking of a given partial k-tree with bounded k. The algorithm takes time $O(n^{2(k+1)\lceil\frac{k+1}{c}\rceil\log(c+1)+2} \cdot \log^{k(k+1)+1} n \cdot \log\log n)$ for any positive integer c. This is the first polynomial-time algorithm for the generalized vertex-ranking problem on partial k-trees, and is faster than the best algorithm of complexity $O(n^{2(k+1)^2(k+2)+2} \cdot \log^{(k+1)(k+3)+2} n)$ known for the ordinary vertex-ranking problem [3]. Our algorithm can be implemented as an NC parallel algorithm, which takes $O(\log n)$ parallel time with $O(n^{4(k+1)\lceil\frac{k+1}{c}\rceil\log(c+1)+1} \cdot \log^{2(k+1)(2k+1)} n)$ processors.

We may replace the positive integer c by a function $f : \{1, 2, \cdots, n\} \rightarrow \mathbf{N}$ to define a more generalized vertex-ranking of a graph as follows: an *f-ranking* 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 [14]. By some trivial modifications of our algorithm for the c-ranking of a partial k-tree, we can find an optimal f-ranking of a given partial k-tree in the same polynomial-time.

References

[1] H.L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, *Journal of Algorithms*, **11** (1990), pp. 631–643.

[2] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, *SIAM J. Comput.*, **25** (1996), pp. 1305–1317.

[3] H.L. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Zs. Tuza, Rankings of graphs, *Proc. of the International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer-Verlag*, **903** (1994), pp. 292–304.

[4] P. de la Torre, R. Greenlaw, and A.A. Schäffer, Optimal edge ranking of trees in polynomial time, *Algorithmica*, **13** (1995), pp. 592–618.

[5] J.S. Deogun, T. Kloks, D. Kratsch, and H. Müller, On vertex ranking for permutation and other graphs, *Proc. of the 11th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Springer-Verlag*, **775** (1994), pp. 747–758.

[6] A.V. Iyer, H.D. Ratliff, and G. Vijayan, Optimal node ranking of trees, *Information Processing Letters*, **28** (1988), pp. 225–229.

[7] 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.

[8] T.W. Lam and F.L. Yue, The NP-completeness of edge ranking, *Manuscript*, 1996.

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

[10] A. Pothen, The complexity of optimal elimination trees, *Technical Report CS-88-13, Pennsylvania State University, USA*, 1988.

[11] N. Robertson and P.D. Seymour, Graph minors. II. Algorithmic aspect of tree-width, *Journal of Algorithms*, **7** (1986), pp. 309–322.

[12] A.A. Schäffer, Optimal node ranking of trees in linear time, *Information Processing Letters*, **33** (1989/90), pp. 91–96.

[13] X. Zhou, M.A. Kashem, and T. Nishizeki, Generalized edge-rankings of trees, *Technical Report of SIGAL, 95-AL-46, Information Processing Society of Japan*, 1995, pp. 73–80. Also to appear in the *Proc. of the 22nd. International Workshop on Graph-Theoretic Concepts in Computer Science.*

[14] X. Zhou, H. Nagai, and T. Nishizeki, Generalized vertex-rankings of trees, *Information Processing Letters*, **56** (1995), pp. 321–328.