ELSEVIER

Theoretical Computer Science 240 (2000) 407–427

Theoretical Computer Science

www.elsevier.com/locate/tcs

Algorithms for generalized vertex-rankings of partial k-trees

Md. Abul Kashem[a], Xiao Zhou[b,*], Takao Nishizeki[a]

[a] *Graduate School of Information Sciences, Tohoku University, Aoba-yama 05, Sendai 980-8579, Japan*
[b] *Education Center for Information Processing, Tohoku University, Aoba-yama 05, Sendai 980-8579, Japan*

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. A c-vertex-ranking is optimal if the number of labels used is as small as possible. We present sequential and parallel algorithms to find an optimal c-vertex-ranking of a partial k-tree, that is, a graph of treewidth bounded by a fixed integer k. The sequential algorithm takes polynomial-time for any positive integer c. The parallel algorithm takes $O(\log n)$ parallel time using a polynomial number of processors on the common CRCW PRAM, where n is the number of vertices in G.

Keywords: Algorithm; Partial k-tree; Separator tree; Treewidth; Vertex-ranking

1. Introduction

An ordinary *vertex-ranking* of a graph G is a labeling (ranking) of the vertices of G with positive integers such that every path between any two vertices with the same label i contains a vertex with label $j > i$ [11]. 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*, also called the *ordered coloring problem* [16], 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 [11]. The vertex-ranking problem is NP-hard in general [3, 21], while Iyer

* Corresponding author.
E-mail addresses: kashem@nishizeki.ecei.tohoku.ac.jp (Md.A. Kashem), zhou@ecip.tohoku.ac.jp (X. Zhou), nishi@ecei.tohoku.ac.jp (T. Nishizeki).

0304-3975/00/$ - see front matter
PII: S0304-3975(99)00240-6

et al. presented an $O(n \log n)$ time sequential algorithm to solve the vertex-ranking problem for trees [11], where n is the number of vertices of the input tree. Then Schäffer obtained a linear-time sequential algorithm by refining their algorithm and its analysis [24]. Recently Deogun et al. gave sequential algorithms to solve the vertex-ranking problem for interval graphs in $O(n^3)$ time and for permutation graphs in $O(n^6)$ time [9]. Bodlaender et al. presented a polynomial-time sequential algorithm to solve the vertex-ranking problem for partial k-trees, that is, graphs of treewidth bounded by a fixed integer k [3]. Borie gave general frameworks to construct families of algorithms on partial k-trees [6]. However, his result does not directly imply a polynomial-time algorithm to solve the vertex-ranking problem for partial k-trees. Very recently Kloks et al. have presented a sequential algorithm for computing the vertex-ranking number of an asteroidal triple-free graph in time polynomial in the number of vertices and the number of minimal separators [17]. On the other hand, de la Torre et al. presented a parallel algorithm to solve the vertex-ranking problem for trees in $O(\log n)$ time using $O(n^2/\log n)$ processors on the CREW PRAM [7]. However, no such parallel algorithms were previously known for the vertex-ranking problem on partial k-trees with $k \geqslant 2$.

A natural generalization of an ordinary vertex-ranking is the c-vertex-ranking [27]. 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. The problem is NP-hard in general, since the ordinary vertex-ranking problem is NP-hard [3, 21]. Zhou et al. have obtained a linear-time sequential algorithm to solve the c-ranking problem for trees [27]. Fig. 1(a) depicts an optimal 2-ranking of a graph G using three ranks, where vertex names 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 names 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 [10, 20].

The edge-ranking problem [12] and the c-edge-ranking problem [26] for a graph G are defined similarly. The edge-ranking problem, that is, 1-edge-ranking problem, is

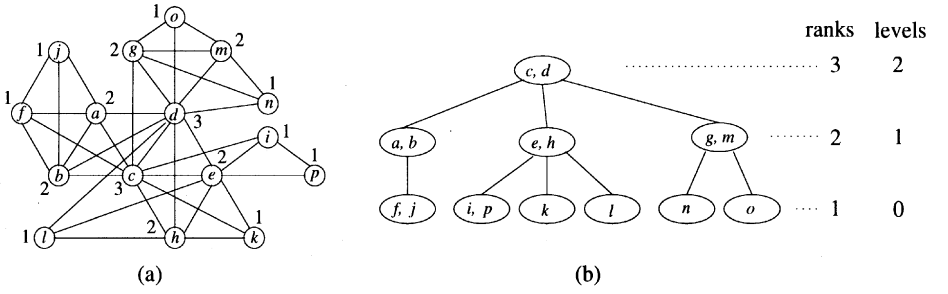

Fig. 1. (a) An optimal 2-ranking of a graph G and (b) a 2-vertex-separator tree of G.

NP-hard in general [18], while de la Torre et al. obtained a sequential algorithm to solve the edge-ranking problem for trees in $O(n^3 \log n)$ time [8]. On the other hand, Zhou et al. presented an $O(n^2 \log \Delta)$ time sequential algorithm to solve the c-edge-ranking problem on trees T for any positive integer c, where Δ is the maximum degree of T [26]. Recently Lam and Yue presented a linear-time algorithm to solve the edge-ranking problem for trees [19].

In this paper we first give a polynomial-time sequential algorithm to solve the c-vertex-ranking problem for any partial k-tree with bounded k and any positive integer c. It is the first polynomial-time sequential algorithm for the c-vertex-ranking problem on partial k-trees. We next give a parallel algorithm for the c-vertex-ranking problem. It is the first parallel algorithm, which takes $O(\log n)$ time using a polynomial number of processors on the common CRCW PRAM. Note that the common CRCW PRAM model allows concurrent writes only when all processors are attempting to write the same value into the same memory location. The results in this paper imply a polynomial-time sequential algorithm and an $O(\log n)$ time parallel algorithm of the c-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 $\leqslant k$, graphs with cutwidth $\leqslant k$, chordal graphs with maximum clique-size k, and graphs that do not contain some fixed planar graph as a minor [1]. The early conference versions of this paper appear in [14, 15].

The remainder of this paper is organized as follows. Section 2 gives preliminary definitions and easy observations. Section 3 defines an equivalence class to solve the c-ranking problem for a partial k-tree. Section 4 gives a sequential algorithm, verifies the correctness of the algorithm, and analyzes its complexity. Section 5 gives a parallel algorithm with its correctness and complexity analysis. Finally, Section 6 concludes with a more generalized vertex-ranking problem.

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

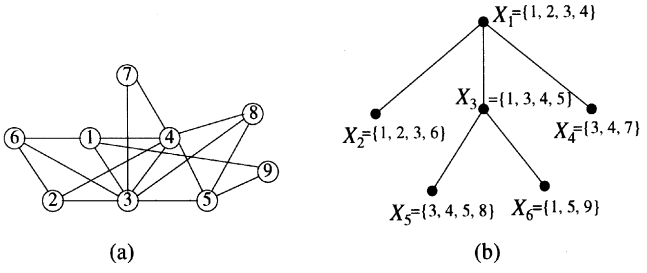

Fig. 2. (a) A partial 3-tree and (b) a tree-decomposition.

$E(G)$ 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*, *root*, *height*, and *level* in their usual meaning.

A *tree-decomposition* of a graph $G=(V,E)$ is a pair (T,S), where $T=(V_T,E_T)$ is a tree and $S=\{X_x \mid x\in V_T\}$ is a collection of subsets of V satisfying the following three conditions [23]:

- $\bigcup_{x\in V_T} X_x = V$;
- for every edge $e=(v,w)\in E$, there exists 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 $X_x\cap X_z\subseteq X_y$.

Fig. 2(b) depicts a tree-decomposition of a graph in Fig. 2(a). 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. A graph G with treewidth $\leqslant k$ is called a *partial k-tree*. Every partial k-tree G has a tree-decomposition (T,S) with width $\leqslant k$ and $n_T\leqslant n$, where n_T is the number of nodes in T [2, 25].

For any fixed integer k, determining whether the treewidth of a graph G is at most k and finding a corresponding tree-decomposition can be done in $O(n)$ sequential time [2, 5]. Let (T,S) be a tree-decomposition of G with width $\leqslant k$ and $n_T\leqslant n$. 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 having d children, say $y_1,y_2,\ldots,y_d$, with $d+1$ new nodes $x_1,x_2,\ldots,x_{d+1}$ such that $X_x=X_{x_1}=X_{x_2}=\cdots=X_{x_{d+1}}$, where x_i, $1\leqslant i\leqslant d$, is the father of x_{i+1} and the ith child y_i of x, and x_{d+1} is a leaf of the tree (see Fig. 3). This transformation can be done in $O(n)$ sequential time. The resulting tree-decomposition (T,S) of $G=(V,E)$ has the following characteristics:

- the width of (T,S) is $\leqslant k$, and the number n_T 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 x in T such that $v,w\in X_x$.

On the other hand, for any fixed integer k, one can determine whether the treewidth of a graph G is k and find a corresponding tree-decomposition in $O(\log_2^2 n)$ parallel

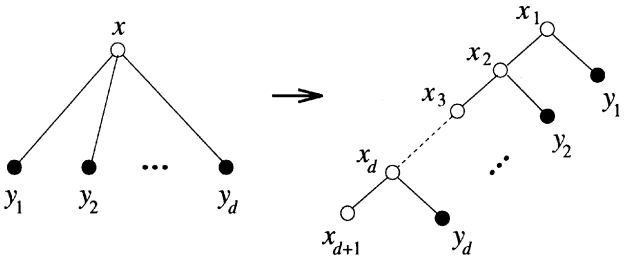

Fig. 3. Illustration of the binary transformation.

time using $O(n)$ operations on the EREW PRAM [4, 5, 22]. Furthermore, from such a tree-decomposition, one can compute a binary tree-decomposition of G with height $O(\log_2 n)$ and width at most $3k+2$ in $O(\log_2 n)$ parallel time using $O(n)$ operations on the EREW PRAM [4].

We next show that the c-ranking number $r_c(G)$ of a partial k-tree G is $O(\log_{c+1} n)$. We first cite Lemma 2.1 from [27].

Lemma 2.1. *Let T be a tree of n_T ($\geqslant 1$) nodes, and let α be any positive integer. Then T has at most α nodes whose removal leaves subtrees each having at most n_T/q nodes, where*

$$q = 2^{\lfloor \log_2(\alpha+3) \rfloor - 1} > (\alpha+3)/4$$

and hence $q \geqslant 2$.

We then have the following lemma on $r_c(G)$.

Lemma 2.2. *Let k and c be any positive integers which are not always bounded, and let G be a partial k-tree of n vertices. Then*

$$r_c(G) \leqslant 1 + a \log_{c+1} n,$$

where

$$a \leqslant \begin{cases} \left\lceil \frac{k+1}{c} \right\rceil \log_2(c+1) & \text{if } c \leqslant k, \\ 2 + 2\log_2(k+1) & \text{if } k+1 \leqslant c \leqslant 4(k+1)^2 - 1, \text{ and} \\ 4 & \text{if } c \geqslant 4(k+1)^2. \end{cases}$$

Proof. Let (T, S) be a tree-decomposition of G with width $\leqslant k$ which is not necessarily binary. Then one may assume that $n_T \leqslant n$ [2].

Recursively applying Lemma 2.1 to T, we first construct an α-vertex-separator tree $T_\alpha(T)$ of tree T, where $\alpha = \max\{1, \lfloor c/(k+1) \rfloor\}$. The height $h_{T_\alpha}(n_T)$ of tree $T_\alpha(T)$

satisfies the following recurrent relation

$$h_{T_\alpha}(n_T) \leqslant 1 + h_{T_\alpha}\left(\left\lfloor \frac{n_T}{q} \right\rfloor\right), \tag{1}$$

where $q = 2^{\lfloor \log_2(\alpha+3) \rfloor - 1} > (\alpha+3)/4$. Solving the recurrence (1) with $h_{T_\alpha}(1) = 0$, we have

$$h_{T_\alpha}(n_T) \leqslant \log_q n_T. \tag{2}$$

We next claim that the α-vertex-separator tree $T_\alpha(T)$ of T can be transformed to a c-vertex-separator tree $T_c(G)$ of G with height

$$h_{T_c}(n) \leqslant \left\lceil \frac{k+1}{c} \right\rceil \log_q n.$$

First consider the case in which $c \geqslant k+1$. In this case $\alpha = \lfloor c/(k+1) \rfloor$, and any node of $T_\alpha(T)$ contains at most α nodes of T, each corresponding to a separator of G having at most $k+1$ vertices of G. Thus any node of $T_\alpha(T)$ corresponds to a separator of G having at most c vertices of G, and hence $T_\alpha(T)$ immediately yields a c-vertex-separator tree $T_c(G)$ whose height $h_{T_c}(n)$ is at most $h_{T_\alpha}(n_T)$. Then by (2) we have

$$h_{T_c}(n) \leqslant h_{T_\alpha}(n_T) \leqslant \log_q n_T \leqslant \log_q n = \left\lceil \frac{k+1}{c} \right\rceil \log_q n.$$

Next, consider the case in which $c \leqslant k$. Then $\alpha = 1$. Let $s = \lceil (k+1)/c \rceil$, and replace each node x of $T_\alpha(T)$ with s new nodes $x_1, x_2, \ldots, x_s$; node x_j is the father of x_{j+1}, $1 \leqslant j \leqslant s-1$, in a new tree; let node x correspond to a set $X_x \subseteq V$, then each node x_j, $1 \leqslant j \leqslant s$, contains at most c vertices in X_x, and every vertex in X_x is contained in some x_j. The resulting tree has height $\lceil (k+1)/c \rceil h_{T_\alpha}(n_T)$, and immediately yields a c-vertex-separator tree $T_c(G)$. The height $h_{T_c}(n)$ of $T_c(G)$ satisfies

$$h_{T_c}(n) \leqslant \left\lceil \frac{k+1}{c} \right\rceil h_{T_\alpha}(n_T) \leqslant \left\lceil \frac{k+1}{c} \right\rceil \log_q n_T \leqslant \left\lceil \frac{k+1}{c} \right\rceil \log_q n.$$

Thus we have verified the claim above.

We finally obtain a c-ranking of G from the c-vertex-separator tree $T_c(G)$ of G as follows: for each i, $0 \leqslant i \leqslant h_{T_c}(n)$, label by rank $i+1$ all vertices of G corresponding to the nodes of $T_c(G)$ at level i, as illustrated in Fig. 1. Then the resulting vertex-labeling is a c-ranking of G using $1 + h_{T_c}(n)$ ranks. Therefore we have

$$\begin{aligned} r_c(G) &\leqslant 1 + h_{T_c}(n) \\ &\leqslant 1 + \left\lceil \frac{k+1}{c} \right\rceil \log_q n \\ &= 1 + \left\lceil \frac{k+1}{c} \right\rceil \log_q(c+1) \log_{c+1} n \\ &\leqslant 1 + a \log_{c+1} n, \end{aligned}$$

where $a = \lceil (k+1)/c \rceil \log_q(c+1)$.

Depending upon the value of c, we have the following three cases.

Case 1: $c \leqslant k$.

In this case $\alpha = 1$ and $q = 2$. Therefore

$$a = \left\lceil \frac{k+1}{c} \right\rceil \log_q(c+1) = \left\lceil \frac{k+1}{c} \right\rceil \log_2(c+1).$$

Case 2: $k+1 \leqslant c \leqslant 4(k+1)^2 - 1$.

In this case $\alpha \geqslant 1$, $q \geqslant 2$, $\lceil (k+1)/c \rceil = 1$, and $c+1 \leqslant 4(k+1)^2$. Therefore

$$a = \left\lceil \frac{k+1}{c} \right\rceil \log_q(c+1) = \log_q(c+1) \leqslant 2 + 2\log_2(k+1).$$

Case 3: $c \geqslant 4(k+1)^2$.

We have $\alpha \geqslant \lfloor c/(k+1) \rfloor \geqslant (c-k)/(k+1)$. Therefore in this case we have

$$\begin{aligned} q &> \frac{\alpha+3}{4} \\ &\geqslant \frac{1}{2}\left(1 + \frac{c+1}{2(k+1)}\right) \\ &\geqslant \left(\frac{c+1}{2(k+1)}\right)^{1/2} \\ &= \frac{(c+1)^{1/4}}{(2(k+1))^{1/2}}(c+1)^{1/4} \\ &> \frac{(2(k+1))^{1/2}}{(2(k+1))^{1/2}}(c+1)^{1/4} \\ &= (c+1)^{1/4}. \end{aligned}$$

Therefore $a = \lceil (k+1)/c \rceil \log_q(c+1) \leqslant \log_q(c+1) \leqslant \log_{(c+1)^{1/4}}(c+1) = 4$. □

In the remaining paper we assume that the value of a is given as in Lemma 2.2. It should be noted that $a = O(1)$ and hence $r_c(G) = O(\log_{c+1} n)$ even if c is not a bounded integer, because k is assumed to be a bounded integer.

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 $1, 2, \ldots, \#\varphi$ as the ranks. For a rank i, $1 \leqslant i \leqslant \#\varphi$, denote by $V(G, \varphi, i)$ the set of vertices v in G with $\varphi(v) = i$, and let $n(G, \varphi, i) = |V(G, \varphi, i)|$. Then φ is a c-ranking of G if and only if $n(D, \varphi, i) \leqslant c$ for any i, $1 \leqslant i \leqslant \#\varphi$, and any connected component D of the graph obtained from G by deleting all vertices with ranks $> i$.

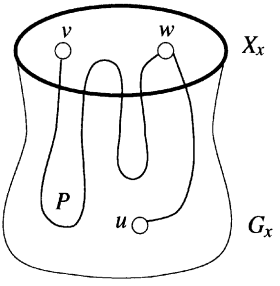

Fig. 4. Graph G_x.

Consider hereafter a binary tree-decomposition (T,S) of a partial k-tree G. We associate a subgraph $G_x=(V_x,E_x)$ of G with each node x of T, where

$$V_x=\bigcup\{X_y \mid y=x \text{ or } y \text{ is a descendant of } x \text{ in } T\}$$

and

$$E_x=\{(v,w)\in E \mid v,w\in V_x\}.$$

(Fig. 4 depicts graph G_x, where X_x is indicated by an oval drawn in a thick line.) Thus G is associated with the root of T.

For a subgraph $G_x=(V_x,E_x)$ of G, $x\in V_T$, we denote by $\varphi|G_x$ the *restriction* of φ to G_x: let $\eta=\varphi|G_x$, then $\eta(v)=\varphi(v)$ for $v\in V_x$. A vertex $u\in V_x$ is said to be *visible* from a vertex $v\in V_x$ under φ in G_x if G_x has a path P from u to v every vertex of which has a rank $\leqslant\varphi(u)$. (See Fig. 4.) The rank $\varphi(u)$ of u is also said to be *visible* from v under φ in G_x if the vertex u is so. Thus the smallest rank visible from v 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 2.3. *Let (T,S) be a binary tree-decomposition of a partial k-tree G, and let x be a node in T. Then a vertex-labeling φ of G_x is a c-ranking of G_x if and only if*

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

(b) *if x is an internal node in T and has two children y and z, then $\varphi|G_y$ and $\varphi|G_z$ are c-rankings of G_y and G_z, respectively.*

Proof. Since the necessity is trivial, we give a proof only for the sufficiency.

Suppose for a contradiction that a vertex-labeling φ satisfies (a) and (b), but φ is not a c-ranking of G_x. Then there exists a rank i such that deletion of all vertices with labels $>i$ from G_x leaves a connected component D such that $n(D,\varphi,i)>c$. Since (a) and (b) hold, x is an internal node of T and D is neither a subgraph of G_y nor a subgraph of G_z. Furthermore, G_y and G_z have common vertices only in X_x. Therefore D has a vertex $v\in X_x$. Then all vertices with label i in D are visible from v in G_x.

Therefore, more than c vertices of rank i are visible from v under φ in G_x, contrary to (a). □

3. Equivalence class

Many algorithms on partial k-trees use dynamic programming. On each node of the tree-decomposition, 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 a sequential algorithm and the number of operations needed for a parallel algorithm mainly depends on the size of the table. Therefore, we shall find a suitable equivalence class for which the table has a polynomial size. The table of our sequential algorithm has a size $O(n^{(k+1)(a+1)}\log_2^{k(k+1)/2} n)$, and the table of our parallel algorithm has a size $O(n^{3(k+1)(a+1)}\log_2^{3(k+1)(3k+2)/2} n)$. Before defining the equivalence class, we need to define a few terms.

Let (T,S) be a binary tree-decomposition of a partial k-tree $G=(V,E)$. Let $R=\{1,2,\ldots,m\}$ be the set of ranks. Let x be a node in T, and let $\varphi: V_x \to R$ be a vertex-labeling of the subgraph $G_x=(V_x,E_x)$. For an integer $i\in R$, we denote by $count(\varphi,v,i)$ the number of vertices ranked by i and visible from $v\in X_x$ under φ in G_x. If φ is a c-ranking of G_x, then by Lemma 2.3 $count(\varphi,v,i)\leqslant c$ for any vertex $v\in X_x$ and any integer $i\in R$. If $i<\varphi(v)$, then $count(\varphi,v,i)=0$. On the other hand, if $i\geqslant\varphi(v)$, then $count(\varphi,v,i)=n(D,\varphi,i)$, where D is the connected component containing v in the graph obtained from G_x by deleting all vertices w with $\varphi(w)>i$.

Iyer et al. introduced an idea of a "critical list" to solve the ordinary vertex-ranking problem for trees [11], while we define a *count-list* $L(\varphi,v)$ and a *list-set* $\mathscr{L}(\varphi)$ as follows:

$$L(\varphi,v)=\{(i,count(\varphi,v,i))\mid \text{rank } i\in R \text{ is visible from } v \text{ under } \varphi \text{ in } G_x\},$$

and

$$\mathscr{L}(\varphi)=\{L(\varphi,v)\mid v\in X_x\}.$$

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

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

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 $\lambda_\varphi(v,w)=\lambda_\varphi(w,v)$. Thus, if $\lambda_\varphi(v,w)\neq\infty$, then v and w are connected by a path in G_x each internal vertex of which has a rank $\leqslant\lambda_\varphi(v,w)$, but are not connected by a path each internal vertex of which has a rank $<\lambda_\varphi(v,w)$. If vertex $u\in V_x$ is visible from w and $\varphi(u)\geqslant\max\{\varphi(v),\lambda_\varphi(v,w),\varphi(w)\}$, then u is visible from v, as illustrated in Fig. 4.

We next define a pair $\mathscr{R}(\varphi)$ as follows:

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

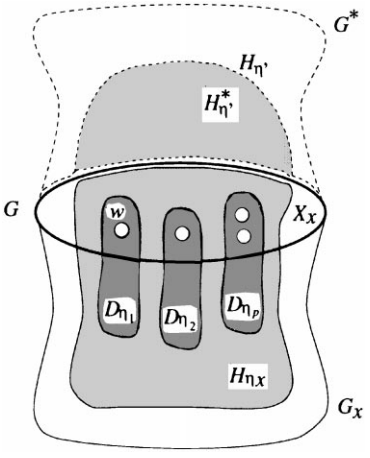

Fig. 5. Graph G.

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

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 without changing the labeling of vertices in G_x. We then have the following lemma.

Lemma 3.1. *Let φ and η be two c-rankings of G_x such that $\mathscr{R}(\varphi) = \mathscr{R}(\eta)$. Then φ is extensible if and only if η is extensible.*

Proof. It suffices to prove that if φ is extensible then η is extensible. Suppose that φ is extensible. Then φ can be extended to a c-ranking φ' of $G = (V, E)$ such that $\varphi'(v) = \varphi(v)$ for any vertex $v \in V_x$. Let $V^* = V - V_x$, and let G^* be the subgraph of G induced by V^*. Extend the c-ranking η of G_x to a vertex-labeling η' of G as follows:

$$\eta'(v) = \begin{cases} \eta(v) & \text{if } v \in V_x; \text{ and} \\ \varphi'(v) & \text{if } v \in V^*. \end{cases}$$

Then it suffices to prove that η' is a c-ranking of G, that is, $n(H_{\eta'}, \eta', i) \leqslant c$ for any rank $i \in R$ and any connected component $H_{\eta'} = (V_{\eta'}, E_{\eta'})$ of the graph obtained from G by deleting all vertices $v \in V$ with $\eta'(v) > i$. Fig. 5 depicts a graph G, where its subgraphs G_x and G^* are indicated by a solid line and a dotted line, respectively, and $H_{\eta'}$ is drawn in a shaded region. There are the following two cases to consider.

Case 1: *$H_{\eta'}$ has no vertex in X_x.*

In this case $H_{\eta'}$ is a subgraph of either G_x or G^*, since G_x is connected with G^* only through the vertices in X_x. Furthermore, $\eta'|G_x = \eta$ and $\eta'|G^* = \varphi'|G^*$ are c-rankings of G_x and G^*, respectively. Therefore $n(H_{\eta'}, \eta', i) \leqslant c$.

Case 2: *$H_{\eta'}$ has a vertex w in X_x as illustrated in Fig.* 5.

In this case obviously $\eta'(w) \leqslant i$. The smallest ranks in the count-lists $L(\varphi, w)$ and $L(\eta, w)$ are equal to $\varphi(w)$ and $\eta(w)$, respectively. Since $\mathscr{R}(\varphi) = \mathscr{R}(\eta)$, we have $L(\varphi, w) = L(\eta, w)$, and hence $\varphi(w) = \eta(w)$. Therefore $\varphi'(w) = \varphi(w) = \eta(w) = \eta'(w) \leqslant i$. Hence, deletion of all vertices $v \in V$ with $\varphi'(v) > i$ from G leaves a connected component $H_{\varphi'} = (V_{\varphi'}, E_{\varphi'})$ containing the vertex w. Since φ' is a c-ranking of G, $n(H_{\varphi'}, \varphi', i) \leqslant c$. Therefore it suffices to prove that $n(H_{\eta'}, \eta', i) = n(H_{\varphi'}, \varphi', i)$.

Since $\mathscr{R}(\eta) = \mathscr{R}(\varphi)$, we have $\eta(v) = \varphi(v)$ for each vertex $v \in X_x$ and $\lambda_\eta(u, v) = \lambda_\varphi(u, v)$ for each pair of vertices $u, v \in X_x$. Furthermore, $\eta'|G^* = \varphi'|G^*$. Therefore one can observe that $V_{\eta'} \cap X_x = V_{\varphi'} \cap X_x$ and $V_{\eta'} \cap V^* = V_{\varphi'} \cap V^*$ although $V_{\eta'} \cap V_x = V_{\varphi'} \cap V_x$ does not necessarily hold. Let $H_{\eta x}$ be the subgraph of $H_{\eta'}$ induced by $V_{\eta'} \cap V_x$, and let $H^*_{\eta'}$ be the subgraph of $H_{\eta'}$ induced by $V_{\eta'} \cap V^*$. Similarly, let $H_{\varphi x}$ be the subgraph of $H_{\varphi'}$ induced by $V_{\varphi'} \cap V_x$, and let $H^*_{\varphi'}$ be the subgraph of $H_{\varphi'}$ induced by $V_{\varphi'} \cap V^*$. Then $n(H_{\eta'}, \eta', i) = n(H_{\eta x}, \eta, i) n(H^*_{\eta'}, \eta', i)$ and $n(H_{\varphi'}, \varphi', i) = n(H_{\varphi x}, \varphi, i) + n(H^*_{\varphi'}, \varphi', i)$. Since $V_{\eta'} \cap V^* = V_{\varphi'} \cap V^*$ and $\eta'|G^* = \varphi'|G^*$, we have $n(H^*_{\eta'}, \eta', i) = n(H^*_{\varphi'}, \varphi', i)$. Therefore it suffices to prove that $n(H_{\eta x}, \eta, i) = n(H_{\varphi x}, \varphi, i)$.

We next show that each of the connected components of $H_{\eta x}$ and $H_{\varphi x}$ contains at least one vertex in X_x. Suppose for a contradiction that a connected component D of $H_{\eta x}$ or $H_{\varphi x}$, say $H_{\eta x}$, contains no vertex in X_x. Since $H_{\eta'}$ is a connected graph containing $w \in X_x$, w is connected to a vertex of D by a path in $H_{\eta'}$. However, it is impossible because D has no vertex in X_x and $H_{\eta x}$ is connected with $H^*_{\eta'}$ only through the vertices in X_x.

Let u be any vertex in $V(H_{\eta x}) \cap X_x = V(H_{\varphi x}) \cap X_x$ $(= V_{\eta'} \cap X_x)$, let D_η be the connected component of $H_{\eta x}$ that contains u, and let D_φ be the connected component of $H_{\varphi x}$ that contains u. We now claim that $V(D_\eta) \cap X_x = V(D_\varphi) \cap X_x$ and $n(D_\eta, \eta, i) = n(D_\varphi, \varphi, i)$. Let v be any vertex in $V(D_\varphi) \cap X_x$. Then obviously $\varphi(v) \leqslant i$ and $\lambda_\varphi(u, v) \leqslant i$. Since $\mathscr{R}(\eta) = \mathscr{R}(\varphi)$, we have $\eta(v) = \varphi(v) \leqslant i$ and $\lambda_\eta(u, v) = \lambda_\varphi(u, v) \leqslant i$. Therefore $v \in V(D_\eta) \cap X_x$. Similarly, we can show that $v \in V(D_\varphi) \cap X_x$ for any vertex $v \in V(D_\eta) \cap X_x$. Hence we have proved that $V(D_\eta) \cap X_x = V(D_\varphi) \cap X_x$. Clearly $n(D_\eta, \eta, i) = count(\eta, u, i)$ and $n(D_\varphi, \varphi, i) = count(\varphi, u, i)$. Since $L(\eta, u) = L(\varphi, u)$, we have $count(\eta, u, i) = count(\varphi, u, i)$ and hence $n(D_\eta, \eta, i) = n(D_\varphi, \varphi, i)$. Thus we have verified the claim.

The claim above implies that $H_{\eta x}$ and $H_{\varphi x}$ have the same number of connected components $D_{\eta_1}, D_{\eta_2}, \ldots, D_{\eta_p}$ and $D_{\varphi_1}, D_{\varphi_2}, \ldots, D_{\varphi_p}$, respectively. (In Fig. 5 $D_{\eta_1}, D_{\eta_2}, \ldots, D_{\eta_p}$ are drawn in darkly shaded regions.) Furthermore, one may assume that $n(D_{\eta_j}, \eta, i) = n(D_{\varphi_j}, \varphi, i)$ for each j, $1 \leqslant j \leqslant p$. Since $n(H_{\eta x}, \eta, i) = \sum_{j=1}^{p} n(D_{\eta_j}, \eta, i)$ and $n(H_{\varphi x}, \varphi, i) = \sum_{j=1}^{p} n(D_{\varphi_j}, \varphi, i)$, we have $n(H_{\eta x}, \eta, i) = n(H_{\varphi x}, \varphi, i)$. □

Thus a feasible vector $\mathscr{R}(\varphi)$ of φ on x can be seen as an equivalence class of extensible c-rankings of G_x.

4. Sequential algorithm

The main result of this section is the following theorem.

Theorem 4.1. *For any positive integer c and any bounded integer k, an optimal c-ranking of a partial k-tree G with n vertices can be found in time*

$$O(n^{2(k+1)(a+1)+2}\log_2^{k(k+1)+1} n \log_2 \log_2 n).$$

In the remaining section we give an algorithm to find an optimal c-ranking of a partial k-tree G in time $O(n^{2(k+1)(a+1)+2}\log_2^{k(k+1)+1} n \log_2 \log_2 n)$. Let (T,S) be a binary tree-decomposition of G with width $\leqslant k$. We first give an algorithm to decide, for a given positive integer m, whether G has a c-ranking φ with $\#\varphi \leqslant m$. We use 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, $1 \leqslant m \leqslant 1 + a\log_{c+1} n$, we determine the minimum value of m such that G has a c-ranking φ with $m = \#\varphi$, and find an optimal c-ranking of G.

A feasible vector $\mathscr{R}(\varphi)$ of φ on x can be seen as an equivalence class of extensible c-rankings of G_x. Remember that $R = \{1,2,\ldots,m\}$ is the set of ranks. Since $|R| = m$ and $0 \leqslant count(\varphi, v, i) \leqslant c$ 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 $|X_x| \leqslant k+1$. Therefore, the number of distinct list-sets $\mathscr{L}(\varphi)$ is at most $(c+1)^{(k+1)m}$. On the other hand, the number of distinct functions $\lambda_\varphi : X_x \times X_x \to R \cup \{0, \infty\}$ 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 $(c+1)^{(k+1)m}(m+2)^{k(k+1)/2}$. One may assume that $c \leqslant n$ and $m \leqslant 1 + a\log_{c+1} n = O(\log_2 n)$ by Lemma 2.2. Therefore, the total number of different feasible vectors on x is $O(n^{(k+1)(a+1)}\log_2^{k(k+1)/2} n)$ for any fixed integer 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 $\#\varphi \leqslant m$.

We first show how to find the table of all feasible vectors $\mathscr{R}(\varphi) = (\mathscr{L}(\varphi), \lambda_\varphi)$ on a leaf x of T. This can be done as follows:

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

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

Since $|V_x| = |X_x| \leqslant k+1$ and $|R| = m$, the number of vertex-labelings $\varphi : V_x \to R$ is at most m^{k+1}. For each vertex-labeling φ, λ_φ can be computed in time $O(1)$. Furthermore, 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 can be done by Lemma 2.3 in time $O(1)$, and if so, computing $\mathscr{L}(\varphi)$ can be done in time $O(1)$. Therefore, steps (1) and (2) can be executed for leaf x in time $O(m^{k+1}) = O(\log_2^{k+1} n)$, and hence the table on x can be found in time $O(\log_2^{k+1} n)$.

We next show how to compute all feasible vectors on an internal node x of T from those on two children y and z of x. One may assume that $X_x = X_y$. By the definition

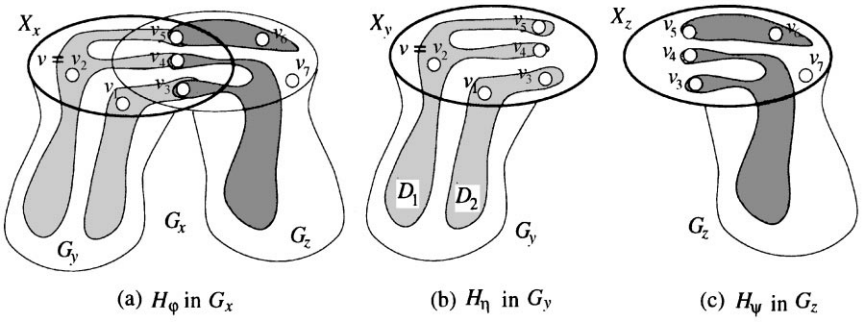

Fig. 6. Graphs G_x, G_y and G_z.

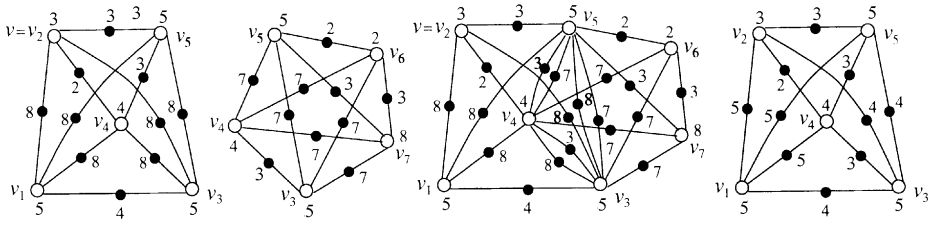

Fig. 7. Graphs $G(\eta)$, $G(\psi)$, $G'(\varphi)$ and $G(\varphi)$.

of $G_x=(V_x,E_x)$, we have $V_x=V_y\cup V_z$ and $E_x=E_y\cup E_z$. Let η and ψ, respectively, 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 $\varphi\,|\,G_z=\psi$. Fig. 6 illustrates G_x, G_y and G_z where X_x, X_y and X_z are drawn by ovals.

We first show how to compute λ_φ from vectors $\mathscr{R}(\eta)$ and $\mathscr{R}(\psi)$. Let $G(\eta)$ be a *λ-graph* defined for η as follows: let $K_{|X_y|}$ be a complete graph of the vertices in X_y; assign a weight of $\eta(v)$ to each vertex v in $K_{|X_y|}$; place a dummy vertex on each edge (v,w) in $K_{|X_y|}$; and assign a weight of $\lambda_\eta(v,w)$ to the dummy vertex, as illustrated in Fig. 7(a) where $X_y=\{v_1,v_2,\ldots,v_5\}$ and the vertices in X_y are drawn by white circles, the dummy vertices by filled circles, and the weights next to the circles. Then the total number of vertices in $G(\eta)$ is at most $k+1+k(k+1)/2=(k+1)(k+2)/2$ and the total number of edges is at most $k(k+1)$. Similarly define a λ-graph $G(\psi)$ for ψ. Fig. 7(b) illustrates $G(\psi)$, where $X_z=\{v_3,v_4,\ldots,v_7\}$. Identify each pair of the same vertices in $X_y\cap X_z$, one in $G(\eta)$ and the other in $G(\psi)$, as illustrated in Fig. 7(c). Let $G'(\varphi)$ be the resulting weighted graph. Then the total number of vertices in $G'(\varphi)$ is at most $(k+1)(k+2)$ and the total number of edges is at most $2k(k+1)$. Then, by the construction of $G'(\varphi)$, the function $\lambda_\varphi: X_x\times X_x\to R\cup\{0,\infty\}$ can be computed as follows:

$$\lambda_\varphi(v,w)=\min\{\lambda\,|\,G'(\varphi)\text{ has a path from }v\in X_x\text{ to }w\in X_x\text{ every internal vertex of which has a weight }\leqslant\lambda\}.$$

Since $G'(\varphi)$ has a constant number of vertices and edges, λ_φ can be computed in time O(1). It is also easy to construct a λ-graph $G(\varphi)$ from $G'(\varphi)$. Fig. 7(d) illustrates $G(\varphi)$, where $X_x = X_y = \{v_1, v_2, \ldots, v_5\}$.

We next show how to compute $\mathscr{L}(\varphi)$ from η and ψ. 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 $i \geqslant \varphi(v)$. Delete all vertices with ranks $> i$ from G_x. Among the connected components of the resulting graph, let H_φ be the one containing v. Then $count(\varphi, v, i) = n(H_\varphi, \varphi, i)$. Since $|E_x| = O(n)$ and $|R| = m = O(\log_2 n)$, the count-lists $L(\varphi, v)$, $v \in X_x$, can be computed in time $O(n \cdot m) = O(n \log_2 n)$. Then checking whether a vertex-labeling φ is a c-ranking of G_x can be done by Lemma 2.3 in time $O(\log_2 n)$, and if so, computing $\mathscr{L}(\varphi)$ can be done in time $O(n \log_2 n)$.

Thus each vector on an internal node can be computed in time $O(n \log_2 n)$. The table of all feasible vectors on an internal node x can be obtained from the pairs of tables of all feasible vectors on the two children of x, and the number of these pairs is $O(n^{2(k+1)(a+1)} \log_2^{k(k+1)} n)$. Therefore the table on x can be computed in time $O(n^{2(k+1)(a+1)+1} \log_2^{k(k+1)+1} n)$.

We thus have the following algorithm CHECK to determine whether G has a c-ranking φ with $\#\varphi \leqslant m$ 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 $O(\log_2^{k+1} n)$ time for each leaf as mentioned before. Since there are $O(n)$ leaves, line 1 can be done in $O(n \log_2^{k+1} n)$ time in total for all leaves. As mentioned above, line 2 can be done in $O(n^{2(k+1)(a+1)+1} \log_2^{k(k+1)+1} n)$ time per node. Note that we keep a c-ranking φ of G_x for each feasible vector to compute $count(\varphi, v, i)$ and check condition (a) in Lemma 2.3. Since line 2 is executed for $O(n)$ nodes in total in line 3, line 3 can be done in $O(n^{2(k+1)(a+1)+2} \log_2^{k(k+1)+1} n)$ time in total. Line 4 can be done in $O(n^{(k+1)(a+1)} \log_2^{k(k+1)/2+1} n)$ time in total. Thus checking whether a partial k-tree G has a c-ranking φ such that $\#\varphi \leqslant m$ can be done in $O(n^{2(k+1)(a+1)+2} \log_2^{k(k+1)+1} n)$ time.

Using the binary search technique over the range of m, $1 \leqslant m \leqslant 1 + a \log_{c+1} n = O(\log_2 n)$, 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 $O(\log_2 \log_2 n)$ times. Therefore, an optimal c-ranking of a partial k-tree G of n vertices can be found in time $O(n^{2(k+1)(a+1)+2} \log_2^{k(k+1)+1} n \log_2$

$\log_2 n$) for any positive integer c and any bounded integer k. This completes the proof of Theorem 4.1.

For the particular case $c=1$, our algorithm finds an optimal ordinary ranking of a partial k-tree G in time $O(n^{2(k+1)^2+2}\log_2^{k(k+1)+1} n \log_2 \log_2 n)$. Thus the algorithm takes essentially $n^{O(k^2)}$ time, while the algorithm of Bodlaender et al. takes $n^{O(k^3)}$ time [3]. The improvement is due to the fact that our equivalent classes are different from those in [3].

5. Parallel algorithm

In this section we prove the following theorem.

Theorem 5.1. *Let G be a partial k-tree of n vertices given by its tree-decomposition with height $O(\log_2 n)$ and width $\leqslant 3k+2$. Then an optimal c-ranking of G can be found in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1}\log_2^{3(k+1)(3k+2)+2} n)$ operations on the common CRCW PRAM for any positive integer c and any bounded integer k.*

Thus the c-ranking problem for partial k-trees is in NC. The following general lemma is well known [3].

Lemma 5.2. *Let A be a given algorithm with $O(\log_2 n)$ parallel computation time. If A involves a total number of q operations, then A can be implemented using p processors in $O(q/p+\log_2 n)$ parallel time.*

If there is an algorithm A which solves the c-ranking problem in $O(\log_2 n)$ parallel time using a total of $q=O(n^{6(k+1)(a+1)+1}\log_2^{3(k+1)(3k+2)+2} n)$ operations, then by adapting Lemma 5.2 with choosing $p=q/\log_2 n$ one can know that A can be implemented using $O(q/\log_2 n)$ processors in $O(\log_2 n)$ parallel time.

Thus by Theorem 5.1 and Lemma 5.2 we have the following corollary.

Corollary 5.3. *For any positive integer c and any bounded integer k, the c-ranking problem for partial k-trees can be solved in $O(\log_2 n)$ parallel time with a polynomial number of processors on the common CRCW PRAM.*

The following lemma is also well known [13].

Lemma 5.4. *Given a parallel algorithm that can be implemented to run in time t on a p-processor common CRCW PRAM, this algorithm can be implemented on a p-processor EREW PRAM to run in $O(t\log p)$ time.*

Thus by Corollary 5.3 and Lemma 5.4 we have the following corollary.

Corollary 5.5. *For any positive integer c and any bounded integer k, the c-ranking problem for partial k-trees can be solved in $O(\log_2^2 n)$ parallel time with a polynomial number of processors on the EREW PRAM.* □

In the remaining section we prove Theorem 5.1. Let (T,S) be a binary tree decomposition of G with height $O(\log_2 n)$ and width $\leqslant 3k+2$. Checking condition (a) in Lemma 2.3 requires $O(\log_2 n)$ parallel time for each internal node x of T, because finding the connected components of a graph obtained from G_x by deleting all vertices with ranks $>i$ needs $O(\log_2 n)$ parallel time for each rank $i \in R$. Thus the straightforward implementation of our sequential algorithm in the preceding section would yield a parallel algorithm of $O(\log_2^2 n)$ time, since the height of T is $O(\log_2 n)$. However, as we show below, checking condition (a) can be done in $O(1)$ parallel time, and hence an optimal c-ranking can be found in $O(\log_2 n)$ parallel time. More precisely, we give a parallel algorithm to find an optimal c-ranking of a partial k-tree G in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1} \log_2^{3(k+1)(3k+2)+2} n)$ operations on the common CRCW PRAM.

We first give a parallel algorithm to decide, for a given positive integer m, whether G has a c-ranking φ with $\#\varphi \leqslant m$. We use parallel 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 parallel search over the range of m, $1 \leqslant m \leqslant 1 + a\log_{c+1} n$, we determine the minimum value of m such that G has a c-ranking φ with $m = \#\varphi$, and find an optimal c-ranking of G.

A feasible vector $\mathscr{R}(\varphi)$ of φ on x can be seen as an equivalence class of extensible c-rankings of G_x. The number of distinct count-lists $L(\varphi, v)$ is at most $(c+1)^m$ for each vertex $v \in X_x$ as mentioned in Section 4. Since $|X_x| \leqslant 3(k+1)$, the number of distinct list-sets $\mathscr{L}(\varphi)$ is at most $(c+1)^{3(k+1)m}$. On the other hand, the number of distinct functions $\lambda_\varphi : X_x \times X_x \to R \cup \{0, \infty\}$ is at most $(m+2)^{3(k+1)(3k+2)/2}$. Therefore, the total number of different feasible vectors on node x is at most $(c+1)^{3(k+1)m}(m+2)^{3(k+1)(3k+2)/2}$. One may assume that $c \leqslant n$ and $m \leqslant 1 + a\log_{c+1} n = O(\log_2 n)$ by Lemma 2.2. Thus the total number of different feasible vectors on x is $O(n^{3(k+1)(a+1)} \log_2^{3(k+1)(3k+2)/2} n)$ for any fixed integer k.

We first compute the table of all feasible vectors $\mathscr{R}(\varphi) = (\mathscr{L}(\varphi), \lambda_\varphi)$ on a leaf x of T by the same method as described in Section 4. Since $|V_x| \leqslant 3(k+1)$ and $|R| = m$, $\mathscr{R}(\varphi)$ can be computed for a leaf in $O(1)$ parallel time using $O(m^{3(k+1)}) = O(\log_2^{3(k+1)} n)$ operations on the EREW PRAM.

We next compute all feasible vectors on an internal node x of T from those on two children y and z of x. One may assume that $X_x = X_y$. Let η and ψ, respectively, 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 as in the case of the sequential algorithm.

We first compute λ_φ by constructing the graph $G'(\varphi)$ as described in Section 4. Since $G'(\varphi)$ has a constant number of vertices and edges, λ_φ can be computed in $O(1)$ time using $O(1)$ operations on the EREW PRAM for each vector $\mathscr{R}(\varphi)=(\mathscr{L}(\varphi),\lambda_\varphi)$.

We next show how to compute $\mathscr{L}(\varphi)$. For the purpose we shall show how to compute $count(\varphi,v,i)$. 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 $i\geqslant\varphi(v)$. Delete all vertices with ranks $>i$ from G_x. Among the connected components of the resulting graph, let H_φ be the one containing v. Let H_η be the subgraph of H_φ induced by $V(H_\varphi)\cap V_y$, and let H_ψ be the subgraph of H_φ induced by $V(H_\varphi)\cap V_z$. In Fig. 6 both H_η and H_ψ have two connected components, and H_φ and the connected components of H_η and H_ψ are drawn by filled regions. Since $V(H_\varphi)=V(H_\eta)\cup V(H_\psi)$, we have

$$V(H_\varphi,\varphi,i)=V(H_\eta,\eta,i)\cup V(H_\psi,\psi,i). \tag{3}$$

Let

$$Q_\varphi(v,i)=V(H_\eta,\eta,i)\cap V(H_\psi,\psi,i). \tag{4}$$

Since $count(\varphi,v,i)=n(H_\varphi,\varphi,i)=|V(H_\varphi,\varphi,i)|$, $n(H_\eta,\eta,i)=|V(H_\eta,\eta,i)|$ and $n(H_\psi,\psi,i)=|V(H_\psi,\psi,i)|$, by (3) and (4) we have

$$count(\varphi,v,i)=n(H_\eta,\eta,i)+n(H_\psi,\psi,i)-|Q_\varphi(v,i)|. \tag{5}$$

Since

$$Q_\varphi(v,i)=\{w\in X_y\cap X_z \mid \varphi(w)=i \text{ and } \lambda_\varphi(v,w)\leqslant i\},$$

$Q_\varphi(v,i)$ can be found in O(1) parallel time for all $v\in X_x$ and $i\in R$. Thus we shall show how to compute $n(H_\eta,\eta,i)$ and $n(H_\psi,\psi,i)$. For the purpose we consider the weighted graph $G'(\varphi)$. (See Fig. 7.) Delete all vertices with weights $>i$ from $G'(\varphi)$. Among the connected components of the resulting graph, let F_φ be the one containing v. Let F_η be the subgraph of F_φ induced by $V(F_\varphi)\cap V(G(\eta))$, and let F_ψ be the subgraph of F_φ induced by $V(F_\varphi)\cap V(G(\psi))$. Fig. 8 depicts F_φ, F_η and F_ψ for $G'(\varphi)$ in Fig. 7(c), where $v=v_2$ and $i=5$. By the construction of $G'(\varphi)$, we have $V(H_\eta)\cap X_y=V(F_\eta)\cap X_y$. H_η and F_η may have more than one connected components, but each of them has at least one vertex in X_y. Furthermore, there is a one-to-one correspondence between the sets of connected components of H_η and F_η: for each connected component D of H_η there exists exactly one connected component C_D of F_η such that $V(D)\cap X_y=V(C_D)\cap X_y$, and vice versa. For each connected component D of H_η, we choose an arbitrary vertex w_D in $V(D)\cap X_y=V(C_D)\cap X_y$ and call w_D the *representative vertex* of both D and C_D. Clearly, all vertices in D that have rank i and are visible from w_D under η in G_y are also visible from any other vertex in $V(D)\cap X_y$ under η in G_y. Let $S_\eta(v,i)$ be the set of representative vertices w_D of all connected components D of H_η. Then $S_\eta(v,i)$ is the same as the set of representative vertices of all connected components of F_η. We then have

$$n(H_\eta,\eta,i)=\sum_{w_D\in S_\eta(v,i)} count(\eta,w_D,i). \tag{6}$$

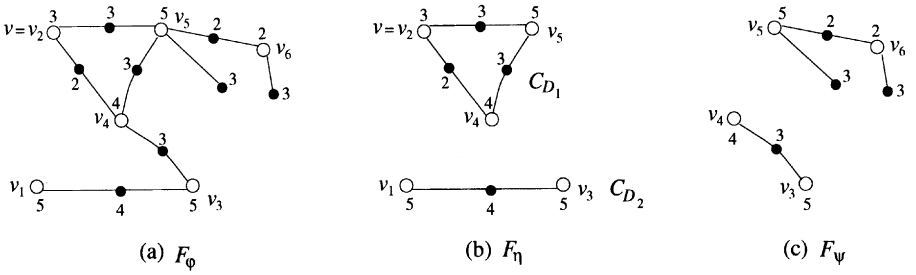

Fig. 8. Graphs F_φ, F_η and F_ψ.

Similarly, we define $S_\psi(v,i)$ for H_ψ. Then we have

$$n(H_\psi,\psi,i)=\sum_{w_D\in S_\psi(v,i)} count(\psi,w_D,i). \tag{7}$$

Therefore, by (5)–(7) we have

$$count(\varphi,v,i)=\sum_{w_D\in S_\eta(v,i)} count(\eta,w_D,i)+\sum_{w_D\in S_\psi(v,i)} count(\psi,w_D,i)-|Q_\varphi(v,i)|.$$

Thus one can easily compute $count(\varphi,v,i)$ from $\mathscr{R}(\eta)$ and $\mathscr{R}(\psi)$.

We now have the following lemma.

Lemma 5.6. *Let η and ψ, respectively, 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. Let $Q_\varphi(v,i)$, $S_\eta(v,i)$ and $S_\psi(v,i)$ be the sets as defined before. Then*
(a) *for any vertex $v\in X_x$ and any rank $i\in R$*

$$count(\varphi,v,i)=\sum_{w_D\in S_\eta(v,i)} count(\eta,w_D,i)+\sum_{w_D\in S_\psi(v,i)} count(\psi,w_D,i)-|Q_\varphi(v,i)|;$$

(b) *φ is a c-ranking of G_x if and only if $count(\varphi,v,i)\leqslant c$ for any vertex $v\in X_x$ and any rank $i\in R$; and*
(c) *$\mathscr{L}(\varphi)$ can be computed in* O(1) *parallel time using* $O(\log_2 n)$ *operations on the EREW PRAM.*

Proof. (a) Has been proved above.

(b) Immediate from Lemma 2.3.

(c) Since $|X_x|,|X_y|,|X_z|\leqslant 3k+2$, the numbers of the vertices and edges in the weighted graphs $G(\eta)$, $G(\psi)$, $G'(\varphi)$ and $G(\varphi)$ are $O(k^2)=O(1)$. Therefore the sets $Q_\varphi(v,i)$, $S_\eta(v,i)$ and $S_\psi(v,i)$ can be computed in O(1) time using O(1) operations for each rank $i\in R$. Since $|R|=m=O(\log_2 n)$, the count-lists $L(\varphi,v)$, $v\in X_x$, can be computed by (a) in O(1) parallel time using $O(m)=O(\log_2 n)$ operations. Then checking whether a vertex-labeling φ is a c-ranking of G_x can be done by (b) in O(1) parallel time using $O(\log_2 n)$ operations, and if so, computing $\mathscr{L}(\varphi)$ can be done in O(1) parallel time using $O(\log_2 n)$ operations on the EREW PRAM. □

Thus each feasible vector $\mathscr{R}(\varphi)=(\mathscr{L}(\varphi),\lambda_\varphi)$ on an internal node can be computed in O(1) parallel time using $O(\log_2 n)$ operations on the EREW PRAM. The table of all feasible vectors on an internal node x can be obtained from the pairs of tables of all feasible vectors on the two children y and z of x, and the number of these pairs is $O(n^{6(k+1)(a+1)}\log_2^{3(k+1)(3k+2)} n)$. To compute the table on x in O(1) parallel time, we need a common CRCW PRAM model. A concurrent read capability is needed, because each feasible vector of the table on y needs to be simultaneously accessed by all the $O(n^{3(k+1)(a+1)}\log_2^{3(k+1)(3k+2)/2} n)$ processors corresponding to the table on z and each feasible vector of the table on z needs to be simultaneously accessed by all the $O(n^{3(k+1)(a+1)}\log_2^{3(k+1)(3k+2)/2} n)$ processors corresponding to the table on y. A concurrent write up of the same value is required, because different pairs of feasible vectors on y and z may compute the same feasible vector on x. Thus the table on x can be computed in O(1) parallel time using $O(n^{6(k+1)(a+1)}\log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM.

We thus have the following parallel algorithm **Ranking** to find an optimal c-ranking of a partial k-tree G, where h is the height of tree T.

```
   Algorithm Ranking;
   begin
1    for m := 1 to 1 + a log_{c+1} n in parallel do
     begin
2      compute a table of all feasible vectors on each leaf of T for
       R = {1, 2, ..., m} in parallel;
3      for l := 1 to h do
4        for each internal node x in T at level l in parallel do
5          compute a table of all feasible vectors from those on the two children
           of x;
6      check whether there exists a feasible vector in the table at the root;
     end
7    find the smallest integer m such that there exists a feasible vector in the table
     at the root for R = {1, 2, ..., m}, and output m as r_c(G).
   end.
```

Lines 2–6 are executed for all m in parallel in Line 1. We now show that Lines 2–6 can be done in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1}\log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM for each m. Line 2 can be done in O(1) parallel time using $O(\log_2^{3(k+1)} n)$ operations on the EREW PRAM for each leaf as mentioned before. Since there are $O(n)$ leaves, Line 2 can be done in O(1) parallel time using $O(n\log_2^{3(k+1)} n)$ operations on the EREW PRAM for all leaves. As mentioned above, Line 5 can be done in O(1) parallel time using $O(n^{6(k+1)(a+1)}\log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM for each node. Since $h=O(\log_2 n)$, Line 3 and Line 4 can be done in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1}\log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM. Line 6 can be done in O(1) parallel time using $O(n^{3(k+1)(a+1)}\log_2^{3(k+1)(3k+2)/2+1} n)$ operations on the common CRCW PRAM.

Thus Lines 2–6 can be done in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1} \log_2^{3(k+1)(3k+2)+1} n)$ operations on the common CRCW PRAM. Therefore Line 1 can be done in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1} \log_2^{3(k+1)(3k+2)+2} n)$ operations on the common CRCW PRAM. Then Line 7 can be done in $O(\log_2 n)$ time using $O(\log_2 n)$ operations on the EREW PRAM.

Thus an optimal c-ranking of a partial k-tree G of n vertices can be found in $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1} \log_2^{3(k+1)(3k+2)+2} n)$ operations on the common CRCW PRAM for any positive integer c and any bounded integer k. This completes the proof of Theorem 5.1. □

6. Conclusion

We first give a polynomial-time sequential algorithm for finding an optimal c-ranking of a given partial k-tree with bounded k. The algorithm takes time $O(n^{2(k+1)(a+1)+2} \log_2^{k(k+1)+1} n \log_2 \log_2 n)$ for any positive integer c, where $a = O(1)$ is given in Lemma 2.3. This is the first polynomial-time sequential algorithm for the generalized vertex-ranking problem on partial k-trees.

We next give a parallel algorithm for finding an optimal c-ranking of a given partial k-tree for any positive integer c and any bounded integer k. The algorithm takes $O(\log_2 n)$ parallel time using $O(n^{6(k+1)(a+1)+1} \log_2^{3(k+1)(3k+2)+1} n)$ processors on the common CRCW PRAM. This is the first parallel algorithm for the generalized vertex-ranking problem on partial k-trees. Note that our algorithm can be implemented in $O(\log_2^2 n)$ parallel time using the same number of processors on the EREW PRAM [13].

We may replace the positive integer c by a function $f : \{1, 2, \ldots, n\} \to \mathbf{N}$ 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 [27]. By minor modifications of our sequential and parallel algorithms for the c-ranking of a partial k-tree, one can obtain algorithms for finding an optimal f-ranking of a given partial k-tree in the same complexity.

References

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

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

[3] H.L. Bodlaender, J.S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, Zs. Tuza, Rankings of graphs, SIAM J. Discrete Math. 21 (1998) 168–181.

[4] H.L. Bodlaender, T. Hagerup, Parallel algorithms with optimal speedup for bounded treewidth, Proc. 22nd Int. Colloq. on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 944, Springer, Berlin, 1995, pp. 268–279.

[5] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (1996) 358–402.
[6] R. Borie, Generation of polynomial time algorithms for some optimization problems on tree-decomposable graphs, Algorithmica 14 (1995) 123–137.
[7] P. de la Torre, R. Greenlaw, T.M. Przytycka, Optimal tree ranking is in NC, Parallel Process. Lett. 2 (1992) 31–41.
[8] P. de la Torre, R. Greenlaw, A.A. Schäffer, Optimal edge ranking of trees in polynomial time, Algorithmica 13 (1995) 592–618.
[9] J.S. Deogun, T. Kloks, D. Kratsch, H. Müller, On vertex ranking for permutation and other graphs, Proc. 11th Ann. Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Vol. 775, Springer, Berlin, 1994, pp. 747–758.
[10] I.S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Software 9 (1983) 302–325.
[11] A.V. Iyer, H.D. Ratliff, G. Vijayan, Optimal node ranking of trees, Inform. Process. Lett. 28 (1988) 225–229.
[12] A.V. Iyer, H.D. Ratliff, G. Vijayan, On an edge-ranking problem of trees and graphs, Discrete Appl. Math. 30 (1991) 43–52.
[13] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, New York, 1992.
[14] M.A. Kashem, X. Zhou, T. Nishizeki, Generalized vertex-rankings of partial k-trees, Proc. 3rd. Ann. Int. Computing and Combinatorics Conf. (COCOON'97), Lecture Notes in Computer Science, Vol. 1276, Springer, Berlin, 1997, pp. 212–221.
[15] M.A. Kashem, X. Zhou, T. Nishizeki, An NC parallel algorithm for generalized vertex-rankings of partial k-trees, Proc. Int. Symp. on Parallel Architectures, Algorithms and Networks (I-SPAN'97), pp. 105–111.
[16] M. Katchalski, W. McCuaig, S. Seager, Ordered colorings, Discrete Math. 142 (1995) 141–154.
[17] T. Kloks, H. Müller, C.K. Wong, Vertex ranking of asteroidal triple-free graphs, Proc. 7th Int. Symp. on Algorithms and Computation (ISAAC'96), Lecture Notes in Computer Science, Vol. 1178, Springer, Berlin, 1996, pp. 174–182.
[18] T.W. Lam, F.L. Yue, Edge ranking of graphs is hard, Discrete Appl. Math. 85 (1998) 71–86.
[19] T.W. Lam, F.L. Yue, Optimal edge ranking of trees in linear time, Proc. Ninth ACM-SIAM Symp. Discrete Algorithms, 1998, pp. 436–445.
[20] J.W.H. Liu, The role of elimination trees in sparse factorization, SIAM J. Matrix Anal. Appl. 11 (1990) 134–172.
[21] A. Pothen, The complexity of optimal elimination trees, Technical Report CS-88-13, Pennsylvania State University, USA, 1988.
[22] B.A. Reed, Finding approximate separators and computing tree-width quickly, Proc. 24th Ann. ACM Symp. on Theory of Computing, 1992, pp. 221–228.
[23] N. Robertson, P.D. Seymour, Graph minors. II. Algorithmic aspect of tree-width, J. Algorithms 7 (1986) 309–322.
[24] A.A. Schäffer, Optimal node ranking of trees in linear time, Inform. Process. Lett. 33 (1989/90) 91–96.
[25] J. van Leeuwen, Graph algorithms, in: Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory, North-Holland, Amsterdam, 1990, pp. 527–631.
[26] X. Zhou, M.A. Kashem, T. Nishizeki, Generalized edge-rankings of trees, Proc. 22nd. Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'96), Lecture Notes in Computer Science, Vol. 1197, Springer, Berlin, 1997, pp. 390–404, also IEICE Trans. on Fundamentals of Electronics, Commun. Comput. Sci. E81-A (2) (1998) 310–320.
[27] X. Zhou, H. Nagai, T. Nishizeki, Generalized vertex-rankings of trees, Informat. Process. Lett. 56 (1995) 321–328.