AN ALGORITHM FOR SOLVING MINIMUM EDGE-RANKING SPANNING TREE PROBLEM ON PARTIAL K-TREES

Razia Sultana
Department of CSE, CIS and CS
Daffodil International University, Dhaka-1207, Bangladesh
E-mail: razia_sultana_renu@yahoo.com

Abstract: *An edge-ranking of a graph G is a labeling of its edges with positive integers such that every path between two edges with the same label i contains an intermediate edge with label j>i. The minimum edge-ranking spanning tree problem is to find a spanning tree of a graph G whose edge-ranking needs least number of ranks. In this paper, we present an algorithm to solve the minimum edge-ranking spanning tree problem on a partial k-tree G in* $O(n^{2\Delta(k+1)+2}\ \Delta^{k(k+1)+2}\ \log_2^{k(k+1)+2} n)$ *time, where n is the number of vertices, Δ is the maximum vertex degree of the graph G and k is bounded by a constant value.*

Keyword: *Algorithm, partial k-trees, edge-ranking, spanning tree.*

1 Introduction

An *edge-ranking* of a graph G is a labeling (ranking) of the edges of G with positive integers such that every path in G with end edges of the same label i contain an internal edge with label $j \geq i+1$ [1, 2]. 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 integer label of an edge is called the *rank* of the edge. The minimum number of ranks needed for an edge-ranking of G is called the *edge-ranking number* and is denoted by $X_r^{'}(G)$. The *edge-ranking problem* is to find an edge-ranking of a given graph G using $X_r^{'}(G)$ ranks. This problem has applications in scheduling the parallel assembly of a complex multi-part product from its components [1].

The problem of finding an optimal edge-ranking was first studied by Iyer *et al.* in 1991 as they found that the problem has an application in scheduling the parallel assembly of multipart products. Lam and Yue have proved that the edge-ranking problem is NP-hard for general graphs [3] and also they have solved the optimal edge-ranking problem on trees in linear-time [4]. A polynomial-time algorithm of this problem is also available on partial k-trees [5].

Makino *et al.* introduced a minimum edge-ranking spanning tree problem which is related to the minimum edge-ranking problem but is essentially different. The *minimum edge-ranking spanning tree problem* (MERST) is to find a spanning tree of G whose edge-ranking is minimum. They proved that this problem is NP-hard and proposed a polynomial-time approximation algorithm for general graphs [6]. Exact polynomial-time algorithm of this problem is available only for threshold graphs [7]. This problem has interesting applications, e.g., to scheduling the parallel assembly of a multi-part product from its components and the relational database [6].

In this paper, for the first time, we give an algorithm for solving the minimum edge-ranking spanning tree problem on partial k-trees that needs $O(n^{2\Delta(k+1)+2}\ \Delta^{k(k+1)+2}\ \log_2^{k(k+1)+2} n)$ time.

2 Definitions and Preliminary Results

2.1 Partial k-tree

All the graphs we consider in this paper are finite and undirected. Let $G=(V,E)$ be a graph with vertex set V and edge set E. The set of vertices and the set of edges of G are often denoted by $V(G)$ and $E(G)$, respectively. A natural generalization of ordinary trees is the so-called k-trees. The class of k-trees is defined recursively as follows [8]:

(a) A complete graph with k vertices is a k-tree.
(b) if $G=(V,E)$ is a k-tree and k vertices $v_1,v_2, \ldots ,v_k$ 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 bounded by a fixed integer.

2.2 Tree-Decomposition

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 [9]:

(a) $\bigcup_{x\in V_T} X_x = V$;

(b) for every edge $e=(v, w)\in E$, there exists a node $x\in V_T$ with $v, w \in X_x$; and

(c) for all $x, y, z \in V_T$, if node y lies on the path from node x to node z in T, then $X_x \cap X_z \subseteq X_y$.

So every partial k-tree G has a tree-decomposition (T, S) with $\le k$ and $n_T \le n$, where n_T is the number of nodes in T and every node of the tree-decomposition can contain at most k+1 vertices.

The construction of a partial k-tree can be represented by a "binary decomposition tree" T_b. Let (T, S) be a tree-decomposition of a graph G with width $\le k$ then it can be transformed into a binary tree-decomposition as follows[8]: Regard T as rooted tree by choosing an arbitrary node as the root and replace every node x of 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 $X_x = X_{x_1} = X_{x_2} = \ldots = X_{x_{d+1}}$, where x_i, $1 \le i \le d$, is the father of x_i+1 and the i-th child y_i of x, and x_{d+1} is a leaf of the tree. This transformation can be done in $O(n)$ time.

Let (T_b, S) be the binary tree-decomposition of a partial k-tree $G=(V, E)$, where $T_b=(V_{T_b}, E_{T_b})$. Let x be a node in T_b and $T(x)$ be the subtree of T_b rooted at x. We associate a subgraph $G_x=(V_x, E_x)$ of G with each node x of tree T_b, where

(a) $V_x = \bigcup \{X_y \mid y=x$ or y is a descendant of x in $T_b\}$; and

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

The graph associated with the root of T_b is the given graph G itself. G_x may have m spanning subgraphs H_x^1, $H_x^2, \ldots, H_x^m$. Let $H_x^p=(V_x, E_x^p)$, be a spanning subgraph of G_x, where $E_x^p \subseteq E_x$.

Let x be a node in T_b and let φ be an edge-labeling of the spanning subgraph $H_x^p=(V_x, E_x^p)$ of G_x. The label (rank) of an edge $e\in E_x^p$ is denoted by $\varphi(e)$. The number of ranks used by an edge-labeling φ is denoted by $\#\varphi$. One may assume without loss of generality that φ uses consecutive integers 1, 2, 3, . . ., $\#\varphi$ as the ranks.

For a rank i, $1 \le i \le \#\varphi$, denote by $E(H_x^p, \varphi, i)$ the set of edges e in H_x^p with $\varphi(e)=i$, and let $n(H_x^p, \varphi, i) = |E(H_x^p, \varphi, i)|$. Then φ is the edge-ranking of H_x^p if and only if $n(D, \varphi, i) \le 1$ for any i, $1 \le i \le \#\varphi$, and any connected component D of the graph obtained from H_x^p by deleting all edges with ranks $>i$.

2.3 Upper Bound and Lower Bound of Edge-ranking Number for Trees

Makino *et al.* presented a top-down algorithm for tree-ranking and analyzed the lower and upper bound of the ranking. We next cite the two lemmas [6].

Lemma 1 For any tree $T=(V_T, E_T)$, we have $X'_r(T) \ge \max\{\Delta_T, \lceil \log_2 n \rceil\}$, where Δ_T is the maximum vertex degree in T and $n=|V_T|$.

Lemma 2 Let $T=(V_T, E_T)$ be a tree with $|V_T|=n$. Then

$$X'_r(T) = \lceil \log_2 n \rceil \qquad \text{if } \Delta_T = 0, 1, 2$$

$$X'_r(T) \le \frac{(\Delta_T - 2)\log_2 n}{\log_2 \Delta_T - 1} \qquad \text{if } \Delta_T \ge 3.$$

3 Main Idea of Algorithm

As like many other algorithms on partial k-trees, dynamic programming and bottom-up tree computation technique is used in this algorithm to solve the problem. On each node of the binary tree-decomposition of the input graph, a table of all possible partial solutions of the problem is computed from leaves to root, where each entry in the table represents an equivalence class. The time complexity of such an algorithm mainly depends on the number of partial solutions generated at each node.

We now characterizes minimum edge-ranking spanning tree problem of a partial k-tree in terms of visible vertices, and also describes types of spanning subgraphs of a partial k-tree.

3.1 Visible Vertices

The rank of an edge $e \in E_x^p$ is said to be visible from a vertex $v \in V_x$ under φ in H_x^p if H_x^p has a path P from v to e every edge of which has a rank $\leq \varphi(e)$.

For a subgraph $G' = (V', E')$ of G, we denote by $\varphi | G'$ a restriction of φ to G'. Let $\varphi' = \varphi | G'$, then $\varphi'(e) = \varphi(e)$ for $e \in E_x$. We can have the following lemma which characterizes the edge-ranking of a spanning subgraph of a partial k-tree by the number of visible vertices.

Lemma 3 *Let T_b be a binary decomposition tree of a partial k-tree G and let x be an internal node in T_b with two children y and z. Then an edge-labeling φ of a spanning subgraph H_x^p of G_x is an edge-ranking of H_x^p if and only if*

(a) *$\varphi | H_y^q$ and $\varphi | H_z^r$ are edge-rankings of H_y^q and H_z^r, respectively, where H_y^q and H_z^r are spanning sub-graphs of G_y and G_z, respectively; and*

(b) *at most one edge of any rank is visible from any vertex $v \in X_x$ under φ in H_x^p.*

Proof. $\Rightarrow$: Suppose that φ is an edge-ranking of a spanning subgraph H_x^p of G_x. Then for any label i, deletion of all edges from H_x^p with labels $>i$ leaves connected components, each having at most one edge with label i.

(a) Let x be an internal node of T_b with two children y and z. Let $\varphi | H_y^q$ and $\varphi | H_z^r$ be the restrictions of φ to H_y^q and H_z^r, where H_y^q and H_z^r are spanning subgraphs of G_y and G_z respectively. Since φ is an edge-ranking of spanning subgraph H_x^p of G_x and H_y^q is a subgraph of H_x^p, for any label i, deletion of all the edges from H_y^q with label $>i$ leaves connected components, each having at most one edge with label i. Therefore $\varphi | H_y^q$ is an edge-ranking of H_y^q. Similarly φ to H_z^r is an edge-ranking of H_z^r.

(b) Let i be any rank. Delete all edges with rank $>i$ from H_x^p. Among the connected components of the resulting graph, let D be the one containing a vertex $v \in X_x$. Let $n(D, \varphi, i)$ be the number of edges in D with rank i. Then exactly $n(D, \varphi, i)$ edges with rank i are visible from v under φ in H_x^p. Since φ is an edge-ranking of H_x^p, we have $n(D, \varphi, i) \leq 1$. Therefore, at most one edge of rank i is visible from v under φ in H_x^p.

$\Leftarrow$: Suppose for a contradiction that an edge-labeling φ satisfies (a) and (b), but φ is not an edge-ranking of H_x^p. Then there exists a rank i such that the deletion of all edges with labels $>i$ from H_x^p leaves a connected component D containing more than one edges containing i. Since (a) and (b) hold, D is neither a subgraph of H_y^q nor H_z^r. Furthermore H_y^q and H_z^r have common vertices only in X_x. Therefore D has a vertex $v \in X_x$. Then all edges with label i in D are visible from v in H_x^p. Therefore, more than one edges of rank i are visible from v in H_x^p, contrary to (b).

3.2 Types of Spanning Subgraphs

Consider x be a node in T_b. To compute a table of all possible partial solutions for each node x of T_b, all possible spanning subgraphs of G_x are generated and for each spanning subgraph, all possible edge-labelings are generated. From all these edge-labelings of spanning subgraphs of G_x, feasible edge-labelings are calculated which form a table of partial solutions at node x. While generating spanning subgraphs of G_x, both spanning trees and spanning forests of G_x are considered. Therefore, for each node x of T_b, 1 to $|X_x|$-tree type spanning subgraphs are considered. A one-tree type spanning subgraph of G_x is a spanning tree of G_x and i-tree type, $2 \leq i \leq (k+1)$, spanning subgraph is a spanning forest of G_x, having exactly i components (trees).

4 Equivalence class

Many algorithms on partial k-trees use dynamic programming. On each node of the tree-decomposition of the input graph, a table of all possible partial solutions of the problem is computed, where each entry in the table represents an equivalence class. The complexity of the algorithm largely depends on the size of the table. So we need to find appropriate equivalence class to reduce the table size. Before defining the equivalence class, we need to define a few terms.

Let $R = \{1,2, . . . ,m\}$ be the set of ranks. Let x be node in T_b and let $\varphi : E_x^p \rightarrow R$ be an edge-labeling of the spanning subgraph $H_x^p = (V_x, E_x^p)$ of G_x. For an integer i, we denote by count(φ, v, i) the number of edges ranked by i and visible from $v \in X_x$ under φ in H_x^p. If φ be an edge-ranking of H_x^p, then by Lemma 3, count(φ, v, i)≤ 1 for any vertex $v \in X_x$ and any integer $i \in R$. Let D be the connected component containing a vertex $v \in X_x$ in the graph obtained from H_x^p, by deleting all edges w with $\varphi(w) > i$. Then count (φ, v, i) $= n(D, \varphi, i)$.

Iyer et al introduced the idea of "critical list" to solve the ordinary vertex-ranking problem for trees [10]. Later similar idea was used to define *visible-list* $L(\varphi, v)$. We use the similar concept for edge-ranking and define *visible-list* $L(\varphi, v)$ as:

$$L(\varphi, v) = \{ \varphi(e) \mid e \in E_x^p \text{ is visible from a vertex } v \text{ under } \varphi \text{ in } H_x^p \}.$$

The ranks in the visible-list $L(\varphi, v)$ are stored in non-increasing order.

For an edge-labeling φ of H_x^p, we now define a function *obstacle* λ_φ: $X_x \times X_x \rightarrow \mathrm{R} \cup \{0, \infty\}$ as follows:

$$\lambda_\varphi(v, w) = \min\{ \lambda \mid H_x^p \text{ has a path } P \text{ from } v \in X_x \text{ to } w \in X_x \text{ such that } \varphi(e) \leq \lambda \text{ for each internal edge } e \text{ of } P \}.$$

Let $\lambda_\varphi(v, w) = 0$ if $(v, w) \in E_x^p$ or $v=w$, and let $\lambda_\varphi(v, w) = \infty$ if H_x^p has no path from v to w. Clearly $\lambda_\varphi(v, w) = \lambda_\varphi(w, v)$.

λ_φ also indicates the type of spanning subgraph of H_x^p. If $\lambda_\varphi(v, w) \neq \infty$ for each $(v, w) \in X_x \times X_x$, then there is a path between any two vertices in H_x^p, that is, H_x^p is a one-tree type spanning subgraph (spanning tree). But if $\lambda_\varphi(v, w) = \infty$ for any $(v, w) \in X_x \times X_x$ then H_x^p is a spanning forest with more than one connected components (trees).

Finally we define a visible *list-set* $L(\varphi)$ and vector $R(\varphi)$ for H_x^p as follows:

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

$$R(\varphi) = (L(\varphi), \lambda_\varphi).$$

We call such a vector $R(\varphi)$ the vector of φ on node x. $R(\varphi)$ is called a feasible vector if the edge-labeling φ is an edge-ranking of H_x^p.

An edge-ranking φ of H_x^p is defined to be extensible if it can be extended to an edge-ranking φ' of a spanning tree T of G without changing the labeling of edges in H_x^p. We then have the following lemma.

Lemma 4 *Let φ and η be two edge-rankings of the same spanning subgraphs or two different spanning subgraphs of G_x such that $R(\varphi) = R(\eta)$. Then φ is extensible if and only if η is extensible.*

Proof. Let φ and η are two edge-rankings of two different spanning subgraphs $H_x^p = (V_x, E_x^p)$ and $H_x^q = (V_x, E_x^q)$ of G_x. It suffices to prove that if φ is extensible then η is extensible. Let $V^* = V - V_x$ and let G^* be the subgraph of G induced by V^* and let $H^* = (V^*, E^*)$ be the spanning subgraph of G^*. Assume that φ is extensible. Then φ can be extended to an edge-ranking φ' of a spanning tree $H^p = (H_x^p \cup H^*)$ of $G=(V, E)$ such that $\varphi'(e) = \varphi(e)$ for any edge $e \in E_x^p$. Let $n(G, \varphi, i)$ be the number of edges in E having rank i for the edge-ranking φ. Extend the edge-ranking η of H_x^q to an edge-labeling η' of a spanning tree $H_q = (H_x^q \cup H^*)$ as follows:

$$\eta' = \begin{cases} \eta(e) & \text{if } e \in E_x^p; \text{and} \\ \varphi'(e) & \text{if } e \in E^*. \end{cases}$$

Then it suffices to prove that η' is a valid edge-ranking of H^q, that is, $n(F_{\eta'}, \eta', i) \leq 1$ for any rank $i \in R$ and for any connected component $F_{\eta'} = (V_{\eta'}, E_{\eta'})$ of the graph obtained from H^q by deleting all the edges $e \in E$ with $\eta'(e) > i$. Then there are the following two cases to consider:

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

In this case, $F_{\eta'}$ is a subgraph of either H^* or H_x^q, since H^* is connected to H_x^q only through vertices in X_x. Moreover $\eta' | H^* = \eta' | H^*$ and $\eta' | H_x^q = \eta$ are edge-rankings of H^* and H_x^q respectively. Therefore $n(F_{\eta'}, \eta', i) \le 1$.

Case 2: $F_{\eta'}$ has a vertex w in X_x.

Let $e \in E_{\eta'}$ is an edge adjacent to w and e has the smallest rank of all edges adjacent to w under η'. In this case, obviously $\eta'(e) \le i$. If $e \in E^*$ then $\eta'(e) = \varphi'(e) \le i$. On the other hand if $e \in E_x^q$ then smallest rank in $L(\eta, w)$ is $\eta(e) \le i$. Since $R(\varphi) = R(\eta)$ we have $L(\eta, w) = L(\varphi, w)$. So smallest rank in $L(\varphi, w)$ equal to $\eta(e)$. Hence, there must be an edge having rank equal to $\eta(e) \le i$ adjacent to w under φ. In both cases there is an edge adjacent to w having rank $\le i$ *un*der φ'. So deletion of all edges $e \in E$ with $\varphi'(e) > i$ from H^p leaves a connected component $F_{\varphi'} = (V_{\varphi'}, E_{\varphi'})$ containing the vertex w. Since φ' is valid ranking of H^p, $n(F_{\varphi'}, \varphi', i) \le 1$. Therefore, it suffices to prove that $n(F_{\eta'}, \eta', i) = n(F_{\varphi'}, \varphi', i)$.

Since $\eta' | H^* = \varphi' | H^*$ one can observe that $V_{\eta'} \cap X_x = V_{\varphi'} \cap X_x$ and $E_{\eta'} \cap E^* = E_{\varphi'} \cap E^*$. Let F_η be the subgraph of $F_{\eta'}$ induced by $E_{\eta'} \cap E_x$ and $F_{\eta'}^*$ be the subgraph of $F_{\eta'}$ induced by $E_{\eta'} \cap E^*$. Similarly, let F_φ be the subgraph of $F_{\varphi'}$ induced by $E_{\varphi'} \cap E_x$ and $F_{\varphi'}^*$ be the subgraph of $F_{\varphi'}$ induced by $E_{\varphi'} \cap E^*$. Then $n(F_{\eta'}, \eta', i) = n(F_\eta, \eta, i) + n(F_{\eta'}^*, \eta', i)$ and $n(F_{\varphi'}, \varphi', i) = n(F_\varphi, \varphi, i) + n(F_{\varphi'}^*, \varphi', i)$. Since $E_{\eta'} \cap E^* = E_{\varphi'} \cap E^*$ and $\eta' | H^* = \varphi' | H^*$ we have $n(F_{\eta'}^*, \eta', i) = n(F_{\varphi'}, \varphi', i)$. Therefore it suffices to prove that $n(F_\eta, \eta, i) = n(F_\varphi, \varphi, i)$.

Each of the connected components of F_η and F_φ contains at least one vertex of X_x. Suppose for a contradiction that a connected component D of F_η or F_φ, say F_η, contains no vertex in X_x. Since $F_{\eta'}$ is a connected graph containing $w \in X_x$, w is connected to a vertex of D by a path in $F_{\eta'}$. However, it is impossible because D has no vertex in X_x and F_η is connected with $F_{\eta'}^*$ only through the vertices in X_x.

Let $u \in V(F_\eta) \cap X_x = V(F_\varphi) \cap X_x$. Let D_η be the connected component of F_η that contains u, and let D_φ be the connected component of F_φ that contains u. Let $v \in V(D_\varphi) \cap X_x$, then obviously $\lambda_\varphi(v, u) \le i$. As $R(\eta) = R(\varphi)$, $\lambda_\varphi(v, u) = \lambda_\eta(v, u) \le i$. 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 $n(D_\eta, \eta, i) = \text{count}(\eta, u, i)$ and $n(D_\varphi, \varphi, i) = \text{count}(\varphi, u, i)$. Since $L(\eta, u) = L(\varphi, u)$, we have $\text{count}(\eta, u, \underline{i}) = \text{count}(\varphi, u, i)$ and hence $n(D_\eta, \eta, i) = n(D_\varphi, \varphi, i)$.

Thus we have proved that F_η and F_φ have same number of connected components $D_{\eta1}, D_{\eta2}, ..., D_{\eta p}$ and $D_{\varphi1}, D_{\varphi2}, ..., D_{\varphi p}$, respectively. Since $n(F_\eta, \eta, i) = \sum_{j=1}^{p} n(D_{\eta j}, \eta, i)$ and $n(F_\varphi, \varphi, i) = \sum_{j=1}^{p} n(D_{\varphi j}, \varphi, i)$. We have $n(D_{\eta j}, \eta, i) = n(D_{\varphi j}, \varphi, i)$.

We have proved that whenever φ and η are edge-rankings of two different spanning subgraphs of G_x, and then φ is extensible if and only if η is extensible. Similarly we can prove that φ is extensible if and only if η is extensible, when φ and η are two edge-ranking of the same spanning subgraph of G_x.

5 The Algorithm

We first give an algorithm to decide, for a positive integer m, whether a spanning tree T of G has an edge-ranking using m ranks exists with $\#\varphi \le m$. We use dynamic programming and bottom-up tree computation on the binary tree T_b: for each node x of T_b from leaves to the root, all (equivalence classes of) edge-rankings of all spanning subgraphs of G_x are constructed from those of two subgraphs G_y and G_z associated with the children y and z of x. If the root of T_b has at least one feasible spanning tree solution, then the partial k-tree G has a spanning tree T with edge-ranking φ such that $\#\varphi \le m$. Then, by using a linear search over the range of m, $1 \le m \le \Delta \log_2 n$, the minimum value of m is determined such that a spanning tree T of G has an edge-ranking φ with $m = \#\varphi$ and find an optimal vertex ranking spanning tree T of G.

We first presented the algorithm edge-ranking to determine whether a spanning tree T of G has an edge-ranking φ with $\#\varphi \le m$ for a positive integer m where $\#\varphi$ is the largest rank assigned by φ.

```
Algorithm edge-ranking
begin
   1. obtain a binary decomposition tree $T_b$ of
      the partial $k$-tree $G$;
   2. for each leaf $x$ of $T_b$ do
         compute a table of all feasible
         vectors;
   3. for each internal node $x$ of $T_b$ do
         compute a table of all feasible-vectors
         from those on the two children of $x$,
         and keep an edge-ranking $\varphi$ of a
         spanning subgraph of $G_x$ arbitrarily
         chosen from the edge-rankings having
         the same feasible-vector;
   4. repeat step 3 up to the root of the tree $T_b$;
   5. check whether there exists a feasible-
      vector for one-tree type spanning
      subgraph of G in the table at the root;
end;
```

Now we describe the procedure of the above algorithm. We first calculate the total number of different equivalence class on any node x of T_b. A feasible vector $R(\varphi)$ of φ on x can be seen as an equivalence class of extensible edge-rankings of spanning subgraphs of G_x by Lemma 4. Since $|R| = m$ and $0 \le \text{count}(\varphi, v, i) \le 1$ for an edge-ranking φ and a rank $i \in R$, the number of distinct visible-lists $L(\varphi, v)$ is at most 2^m for each vertex $v \in V_x$. Furthermore $|X_x| \le (k+1)$. Therefore, the number of distinct list-sets $L(\varphi)$ is at most $2^{m(k+1)}$. 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 x is at most $2^{m(k+1)} \cdot (m+2)^{k(k+1)/2}$. One may assume that $m \le \Delta \log_2 n = O(\Delta \log_2 n)$ by Lemma 1. Therefore the total number of different feasible vectors on x is $O(n^{\Delta (k+1)} \Delta^{k(k+1)/2} \log_2^{k(k+1)/2} n)$ for any fixed integer k.

Next we show how to find the table of all feasible vectors $R(\varphi) = (L(\varphi), \lambda(\varphi))$ on a leaf x of T_b. This can be done as follows:

1. enumerate all edge-labelings $\varphi : E_x^p \to$R of a spanning subgraph H_x^p of G_x;
2. compute all feasible vectors $R(\varphi)$ from the edge-labelings φ of H_x^p; and repeat step (1) and (2) for all spanning subgraphs of G_x on x.
3. repeat step (1) and (2) for all spanning subgraphs of G_x on x.

Since $|V_x| = |X_x| \le k+1$ and $|R| = m$, the number of edge-labelings $\varphi : E_x^p \to R$ is at most m^k. For each edge-labeling φ, λ_φ can be computed in time $O(1)$. Furthermore, the visible-lists $L(\varphi, v)$, $v \in X_x = V_x$, can be done by Lemma 3 in time $O(1)$, and if so, computing $L(\varphi)$ can be done in time $O(1)$. Therefore step (1) and (2) can be executed in time $O(m^k) = O(\Delta^k \log^k{}_2 n)$. Since the number of spanning subgraphs of G_x is a function of k, so step (3) can be executed for a leaf x in time $O(\Delta^k \log^k{}_2 n)$ and thus the table on x can be found in time $O(\Delta^k \log^k{}_2 n)$.

We next show how to compute all feasible vectors on an internal node x of T_b from those on two children y and z of x. One may assume that $X_x = X_y$. By the definition of $G_x = (V_x, E_x)$, we have $V_x = V_y \cup V_z$ and $E_x = E_y \cup E_z$ and $E_x = E_y \cap E_z = \phi$. Let η and ψ respectively be the edge-rankings of the spanning subgraphs H_y^q and H_z^r of G_y and G_z

respectively. H_x^p be the resultant graph such that $H_x^p = H_y^q \bigcup H_z^r$ so obviously it is a subgraph of G_x.

First we have to check whether H_x^p be a spanning subgraph of G_x or not. If for any pair of vertices $(v, w) \in (X_y \bigcap X_z) \times (X_y \bigcap X_z)$, there is two different paths in H_x^p then H_x^p is not a spanning subgraph of G_x, so discard the vector. But if this condition is false for all pairs $(v, w) \in (X_y \bigcap X_z) \times (X_y \bigcap X_z)$ then H_x^p is spanning subgraph of G_x. This spanning subgraph checking can be done in time $O(1)$.

Now let φ be the edge-labeling of H_x^p extended from η and ψ, then we have $\varphi \mid H_y^q = \eta$ and $\varphi \mid H_z^r = \psi$.

Now we show how to compute λ_φ from vectors $R(\eta)$ and $R(\psi)$. Let $G(\eta)$ be an edge-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 $\lambda_\eta(v,w)$ to each edge (v,w) in $K_{|X_y|}$.Then the total number of vertices in $G(\eta)$ is at most $k+1$ and the total number of edges is at most $k(k+1)/2$. Similarly define an edge-λ-graph $G(\psi)$ for ψ. Identify each pair of the same vertices in $(X_y \bigcap X_z)$, one in $G(\eta)$ and the other in $G(\psi)$. Let $G'(\varphi)$ be the resulting weighted graph. Then the total number of vertices in $G'(\varphi)$ is at most $2(k+1)$ and the total number of edges is at most $k(k+1)$. Then, by the construction of $G'(\varphi)$, the function $\lambda_\varphi : X_x \times X_x \rightarrow \mathrm{R} \bigcup \{0, \infty\}$ can be computed as follows:

$\lambda_\varphi(v, w) = \min\{\ \lambda \mid G'(\varphi)$ has a path P from
$v \in X_x$ to $w \in X_x$ every internal edges of which has a eight $\leq \lambda\ \}$.

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 $G'(\varphi)$.

We next show how to compute $L(\varphi)$ from η and ψ. Let $i \in R$ be any rank. Delete all the vertices with rank $> i$ from H_x^p. Among the connected components of the resulting graph, let H_φ be the one containing a vertex $v \in X_x$. Then count(φ, v, i)= $n(H_\varphi, \varphi, i)$. Since $|E_x|$=O(n) and $|R|=m$=O($\Delta \log_2 n$), the count-lists L(φ, v), $v \in X_x$, can be computed in time O($n.m$)=O($n \Delta \log_2 n$). Then checking whether an edge-labeling φ is an edge-ranking of H_x^p can be done by Lemma 3 in time O($\Delta \log_2 n$), and if so, computing L(φ) can be done in time O($\Delta n \log_2 n$). The table of all feasible vectors on an internal node x can be obtained from the pairs of tables of all vectors on the to children of x, and the number of these pairs is $O(n^{2\Delta(k+1)} \Delta^{k(k+1)} \log_2^{k(k+1)} n)$. Therefore the table on x can be computed in time $O(n^{2\Delta(k+1)+1} \Delta^{k(k+1)+1} \log_2^{k(k+1)+1} n)$.

Finally, we have to check all feasible-vectors at root to find out whether there exists a feasible-vector for one-tree type spanning subgraph (spanning tree) of G. At root, computing all feasible-vectors needs $O(\mathrm{n}^{\Delta(k+1)+1} \Delta^{k(k+1)/2+1} \log_2^{\mathrm{k(k+1)/2+1}} n)$ time. And then checking whether an edge-ranking φ is a valid solution for one-tree type spanning subgraph H_x^p of G can be done by examining each $w \in \lambda_\varphi$ of $R(\varphi)$. If $w \neq \infty$ for all $w \in \lambda_\varphi$, then all vertices of H_x^p is connected, which implies that H_x^p is a one tree-type spanning subgraph of G. Consequently, we can say G has a vertex-ranking spanning tree φ. This checking can be done in $O(1)$ time.

6 Time complexity of the Algorithm

The main result of this paper is this following theorem.

Theorem 1 *A minimum edge-ranking spanning tree of a partial k-tree with n vertices can be found in time* $O(n^{2\Delta(k+1)+2} \Delta^{k(k+1)+2} \log_2^{k(k+1)+2} n)$ *where n is the number of vertices and* Δ *is the maximum vertex degree of the graph.*

Line 1 of the algorithm can be done in $O(n)$ time [9]. Line 2 can be done for each leaf in O($\Delta^{\mathrm{k}} \log^k{}_2 n$) time. Since there are $O(n)$ leaves, line 2 can be done in O($n \Delta^{\mathrm{k}} \log^k{}_2 n$) time in total for all leaves. Since line 3 is executed for $O(n)$ nodes in total in line 4, line 4 can be done in $O(n^{2\Delta(k+1)+2} \Delta^{k(k+1)+1} \log_2^{k(k+1)+1} n)$ time in total. At last Line 5 can be done in $O(n^{\Delta(k+1)+1} \Delta^{k(k+1)/2+1} \log_2^{\mathrm{k(k+1)/2+1}} n)$ time in total. Thus checking whether a spanning tree of a partial *k*-tree *G* has an edge-ranking φ such that $\#\varphi \leq m$ can be done in $O(n^{2\Delta(k+1)+2}\Delta^{k(k+1)+1}\log_2{}^{k(k+1)+1}n)$ time. Using the linear search technique over the range of *m*, $1 \leq m \leq \Delta\log_2 n$, one can find the smallest integer $X_r^{'}(T)$ such that *T* has an edge-ranking φ with $\#\varphi = X_r^{'}(T)$ by calling the algorithm edge ranking $O(\Delta \log_2 n)$ times. Therefore, a minimum edge-ranking spanning tree *T* of a partial *k*-tree *G* with *n* vertices can be found in time for any bounded integer *k*. This completes the proof of Theorem 1.

7 Conclusion

In this paper, we present an algorithm for solving the minimum edge-ranking spanning tree problem on partial *k*-trees. It is the first polynomial-time algorithm for solving the problem on partial *k*-trees for small values of *k*. With some trivial modifications, our algorithm can be used to solve the *c*-edge-ranking spanning tree problem on partial *k*-trees.

References

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

[2] J. S. Deogun, and Y. Peng, “Edge ranking of trees”, Congressus Numerantium, vol. 79, 1990, pp. 19–28.

[3] T. W. Lam, and F. L. Yue, “Edge Ranking of graphs is hard”, *Discrete Applied Mathematics*, Vol. 85, 1998, pp. 71–86.

[4] T. W. Lam, and F. L. Yue, “Optimal edge ranking of trees in linear time”, *Algorithmica*, Vol. 30, 2001, pp. 12–33.

[5] M. A. Kashem, X. Zhou, and T. Nishizeki, “Algorithms for generalized edge rankings of partial *k*-trees with bounded maximum degree”, *Proceedings of the 1st International Conference on Computer and Information Technology (ICCIT)*, 1998, pp. 45–51.

[6] K. Makino, Y. Uno, and T. Ibaraki, “On minimum edge ranking spanning trees”, *Journal of Algorithms*, Vol. 38, 2001, pp.411-437.

[7] K. Makino, Y. Uno, and T. Ibaraki, “Minimum edge ranking spanning trees of threshold graphs”, *LNCS*, Vol. 2518, 2002, pp. 428–440.

[8] H. L. Bodlaender, “Polynomial algorithms for graph isomorphism and chromatic index on partial *k*-trees,” *Journal of Algorithms*, Vol. II, 1990, pp. 631–643.

[9] H. L. Bodlaender, “A linear time algorithm for finding tree- decompositions of small tree width,” *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing*, Vol. 25, 1996, pp. 1305–1317.

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

Razia Sultana has completed her M.Sc. Engg. in Information and Communication Technology from Institute of Information and Communication Technology (IICT) of Bangladesh University of Engineering and Technology (BUET) and B.Sc. Engg. in Computer Science and Engineering from Rajshahi University of Engineering and Technology (RUET). Now she is working as Senior Lecturer in the Department of CSE, CIS & CS of Daffodil International University. Her topics of interest are Graph Theory, Bioinformatics, and Networking.