2009-01-01

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

Sultana, Razia
Daffodil International University

http://hdl.handle.net/20.500.11948/481

Downloaded from http://dspace.library.daffodilvarsity.edu.bd, Copyright Daffodil International University Library
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 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 $O(n^{2(k+1)+2} \Delta^{(k+1)+2} \log_2^{(k+1)} n)$ time, where n is the number of vertices, $\Delta$ 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 $O(n^{2(k+1)+2} \Delta^{(k+1)+2} \log_2^{(k+1)} n)$ 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 $G'=(V \setminus \{w\}, E \setminus \{(v_i, w) | 1 \leq i \leq k\})$ 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 | x ∈ V_T} is a collection of subsets of V satisfying the following three conditions [9]:

(a) Y ∈ V_T, X_Y = V;
(b) For every edge e = (v, w) ∈ E, there exists a node x ∈ V_T with v, w ∈ X_x and
(c) For all x, y, z ∈ V_T, if node y lies on the path from node x to node z in T, then X_x ∩ X_y ⊆ X_z.

So every partial k-tree G has a tree-decomposition (T, S) with ≤ k and n_T ≤ 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 ≤ 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, . . . , y_d, with d + 1 new nodes x_1, x_2, . . . , x_{d+1} such that x_1 = x_{y_1} = X_{y_2} = . . . = X_{y_d}, where x_1, 1 ≤ i ≤ 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 = Y {X_y | y=x or y is a descendant of x in T_b}; and
(b) E_x = {v, w} ∈ E | v, w ∈ 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^1_x, H^2_x, . . . , H^m_x. Let H^p_x = (V_x, E^p_x) be a spanning subgraph of G_x, where E^p_x ⊆ E_x.

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

For a rank i, 1 ≤ i ≤ #ϕ, denote by E(H^p_x, ϕ,i) the set of edges e in H^p_x with ϕ(e) = i, and let n (H^p_x, ϕ,i) = |E(H^p_x, ϕ,i)|. Then ϕ is the edge-ranking of H^p_x if and only if n(D, ϕ, i) ≤ 1 for any i, 1 ≤ i ≤ #ϕ, and any connected component D of the graph obtained from H^p_x 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', (T) ≥ max {Δ_T, ⌊log_2 n⌋}, 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', (T) ≤ ⌊log_2 n⌋ if Δ_T = 0, 1, 2
X', (T) ≤ (Δ_T − 2)log_2 n log_2 Δ_T − 1 if Δ_T ≥ 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 \( e \in E^p_x \) is said to be visible from a vertex \( v \in V_x \) under \( \varphi \) in \( H^p_x \) if \( H^p_x \) 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 \( \varphi \) 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 \( \varphi \) of a spanning subgraph \( H^p_x \) of \( G_x \) is an edge-ranking of \( H^p_x \) if and only if

(a) \( \varphi |_{H^p_y} \) and \( \varphi |_{H^p_z} \) are edge-rankings of \( H^p_y \) and \( H^p_z \), respectively, where \( H^p_y \) and \( H^p_z \) are spanning subgraphs 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 \( \varphi \) in \( H^p_x \).

**Proof.** \( \Rightarrow: \) Suppose that \( \varphi \) is an edge-ranking of a spanning subgraph \( H^p_x \) of \( G_x \). Then for any label \( i \), deletion of all edges from \( H^p_x \) 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 \( \varphi |_{H^p_y} \) and \( \varphi |_{H^p_z} \) be the restrictions of \( \varphi \) to \( H^p_y \) and \( H^p_z \), where \( H^p_y \) and \( H^p_z \) are spanning subgraphs of \( G_y \) and \( G_z \) respectively. Since \( \varphi \) is an edge-ranking of \( H^p_y \) of \( G_x \) and \( H^p_z \) is a subgraph of \( H^p_y \), for any label \( i \), deletion of all the edges from \( H^p_y \) with label \( > i \) leaves connected components, each having at most one edge with label \( i \). Therefore \( \varphi |_{H^p_y} \) is an edge-ranking of \( H^p_y \). Similarly \( \varphi \) to \( H^p_z \) is an edge-ranking of \( H^p_z \).

(b) Let \( i \) be any rank. Delete all edges with rank \( > i \) from \( H^p_x \). 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 \( \varphi \) in \( H^p_x \). Since \( \varphi \) is an edge-ranking of \( H^p_x \), we have \( n(D, \varphi, i) \leq 1 \). Therefore, at most one edge of rank \( i \) is visible from \( v \) under \( \varphi \) in \( H^p_x \).

\( \Leftarrow: \) Suppose for a contradiction that an edge-labeling \( \varphi \) satisfies (a) and (b), but \( \varphi \) is not an edge-ranking of \( H^p_x \). Then there exists a rank \( i \) such that the deletion of all edges with labels \( > i \) from \( H^p_x \) leaves a connected component \( D \) containing more than one edges containing \( i \). Since (a) and (b) hold, \( D \) is neither a subgraph of \( H^p_y \) nor \( H^p_z \).

Furthermore \( H^p_y \) and \( H^p_z \) 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 \( H^p_x \). Therefore, more than one edges of rank \( i \) are visible from \( v \) in \( H^p_x \), 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 \( R = \{1, 2, \ldots, m\} \) be the set of ranks. Let \( x \) be node in \( T_e \) and let \( \varphi: E^p_x \to R \) be an edge-labeling of the spanning subgraph \( H^p_x = (V_x, E^p_x) \) of \( G_x \). For an integer \( i \), we denote by count(\( \varphi, v, i \)) the number of edges ranked by \( i \) and visible from \( v \in X_x \) under \( \varphi \) in \( H^p_x \).

If \( \varphi \) be an edge-ranking of \( H^p_x \), then by Lemma 3, count(\( \varphi, v, i \)) \( \leq 1 \) 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 \( H^p_x \), by deleting all edges \( w \) with \( \varphi(w) > i \). Then count(\( \varphi, v, i \)) = count(\( D, \varphi, i \)).

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:

\[
L(\varphi, v) = \{ \varphi(v) \mid e \in E^p_x \text{ is visible from a vertex } v \text{ under } \varphi \text{ in } H^p_x \}.
\]

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

For an edge-labeling \( \varphi \) of \( H^p_x \), we now define a function obstacle \( \lambda_\varphi: X_X \times X_x \to \mathbb{N}_0 \) as follows:

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

Let \( \lambda_\varphi(v, w) = 0 \) if \((v, w) \in E^p_x \) or \( v = w \), and let \( \lambda_\varphi(v, w) = \infty \) if \( H^p_x \) has no path from \( v \) to \( w \). Clearly \( \lambda_\varphi(v, w) = \lambda_\varphi(w, v) \).

\( \lambda_\varphi \) also indicates the type of spanning subgraph of \( H^p_x \). If \( \lambda_\varphi(v, w) = \infty \) for each \((v, w) \in X_x \times X_x \), then there is a path between any two vertices in \( H^p_x \), that is, \( H^p_x \) 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 \( H^p_x \) 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 \( H^p_x \) as follows:

\[
L(\varphi) = \{ L(\varphi, v) \mid v \in X_e \}.
\]

\[
R(\varphi) = \{ R(\varphi) \mid v \in X_e \}.
\]

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

An edge-ranking \( \varphi \) of \( H^p_x \) is defined to be extensible if it can be extended to an edge-ranking \( \varphi' \) of a spanning tree \( T \) of \( G \) without changing the labeling of edges in \( H^p_x \). We then have the following lemma.

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

**Proof.** Let \( \varphi \) and \( \eta \) be two edge-rankings of two different spanning subgraphs \( H^p_x = (V_x, E^p_x) \) and \( H^q_x = (V_x, E^q_x) \) of \( G_x \). It suffices to prove that if \( \varphi \) is extensible then \( \eta \) 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 \( \varphi \) is extensible. Then \( \varphi \) can be extended to an edge-ranking \( \varphi' \) of a spanning tree \( H' = (H^p_x \cap H^q_x) \) of \( G = (V, E) \) such that \( \varphi'(e) = \varphi(e) \) for any \( e \in E^p_x \). Let \( n(G, \varphi, i) \) be the number of edges in \( E \) having rank \( i \) for the edge-ranking \( \varphi \). Extend the edge-ranking \( \eta \) of \( H^q_x \) to an edge-labeling \( \eta' \) of a spanning tree \( H_{\eta'} = (H^q_x \cap H^* \eta) \) as follows:

\[
\eta' = \begin{cases} 
\eta(e) & \text{if } e \in E^p_x; \text{ and} \\
\varphi(e) & \text{if } e \in E^*.
\end{cases}
\]

Then it suffices to prove that \( \eta' \) is a valid edge-ranking of \( H^p_x \), 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 \( H^p \) by deleting all the edges.
e ∈ E with η'(e) > i. Then there are the following two cases to consider:

Case 1: $F_{\eta'}$ has no vertex in $X_e$.
In this case, $F_{\eta'}$ is a subgraph of either $H^*$ or $H^*_{\eta'}$, since $H^*$ is connected to $H^*_{\eta'}$ only through vertices in $X_e$. Moreover $\eta'[H^*] = \eta'[H^*_{\eta'}]$ and $\eta'[H^*_{\eta'}] = \eta$ are edge-rankings of $H^*$ and $H^*_{\eta'}$ respectively. Therefore $n(F_{\eta'}, \eta', i) ≤ 1$.

Case 2: $F_{\eta'}$ has a vertex w in $X_e$.
Let $e ∈ E_{\eta'}$ be an edge adjacent to $w$ and $e$ has the smallest rank of all edges adjacent to $w$ under $\eta'$. In this case, obviously $\eta'(e) ≤ i$. If $e ∈ E^*$ then $\eta'(e) = \varphi'(e) ≤ i$. On the other hand if $e ∈ E^*_{\eta'}$ then smallest rank in $L(\eta, w)$ is $\eta(e) ≤ i$. Since $R(\varphi) = R(\eta)$ we have $L(\varphi, w) = L(\varphi, w)$. So smallest rank in $L(\varphi, w)$ equal to $\eta(e)$. Hence, there must be an edge having rank equal to $\eta(e) ≤ i$ adjacent to $w$ under $\varphi$. In both cases there is an edge adjacent to $w$ having rank $\leq i$ under $\varphi'$. So deletion of all edges $e ∈ E$ with $\varphi'(e) > i$ from $H_\varphi$ leaves a connected component $F_{\varphi'} = (V_{\varphi'}, E_{\varphi'})$ containing the vertex $w$. Since $\varphi'$ is valid ranking of $H^*$, $n(F_{\varphi'}, \varphi', i) ≤ 1$. 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 $V_{\eta'} \cap X_e = V_{\varphi'} \cap X_e$ and $E_{\eta'} \cap E^* = E_{\varphi'} \cap E^*$. Let $F_{\eta'}$ be the subgraph of $F_{\eta'}$ induced by $E_{\eta'} \cap E_e$ and $F_{\varphi'}$ be the subgraph of $F_{\varphi'}$ induced by $E_{\varphi'} \cap E_e$. Similarly, let $F_{\varphi'}$ be the subgraph of $F_{\varphi'}$ induced by $E_{\varphi'} \cap E_e$ and $F_{\varphi'}$ be the subgraph of $F_{\varphi'}$ induced by $E_{\varphi'} \cap E_e$. Then $n(F_{\eta'}, \eta', i) = n(F_{\eta'}, \eta, i) + n(F_{\eta'} \cap E_e, \eta', i)$ and $n(F_{\varphi'}, \varphi', i) = n(F_{\varphi'}, \varphi, i) + n(F_{\varphi' \cap E_e}, \varphi', i)$. Since $E_{\eta'} \cap E^* = E_{\varphi'} \cap E^*$ and $\eta'[H^*] = \varphi'[H^*]$ we have $n(F_{\eta'}, \eta', i) = n(F_{\eta'}, \varphi', i)$. Therefore it suffices to prove that $n(F_{\eta'}, \eta, i) = n(F_{\varphi'}, \varphi, i)$.

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

Let $u ∈ V(F_{\eta'}) \cap X_e = V(F_{\varphi'}) \cap X_e$. Let $D_u$ be the connected component of $F_{\eta'}$ that contains $u$, and let $D_{\varphi'}$ be the connected component of $F_{\varphi'}$ that contains $u$. Let $v ∈ V(D_u) \cap X_e$, then obviously $\lambda_{\eta'}(v, u) ≤ i$. As $R(\eta') = R(\varphi')$, $\lambda_{\varphi'}(v, u) = \lambda_{\eta'}(v, u) ≤ i$. Therefore $v ∈ V(D_{\varphi'}) \cap X_e$.

Similarly, we can show that $v ∈ V(D_{\varphi'}) \cap X_e$ for any vertex $v ∈ V(D_{\eta'}) \cap X_e$. Hence we have proved that $V(D_{\eta'}) \cap X_e = V(D_{\varphi'}) \cap X_e$. 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 $\text{count}(\eta, u, i) = \text{count}(\varphi, u, i)$ and hence $n(D_{\eta'}, \eta, i) = n(D_{\varphi'}, \varphi, i)$.

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

We have proved that whenever $\varphi$ and $\eta$ are edge-rankings of two different spanning subgraphs of $G_e$, then $\varphi$ is extensible if and only if $\eta$ is extensible. Similarly we can prove that $\varphi$ is extensible if and only if $\eta$ is extensible, when $\varphi$ and $\eta$ are two edge-ranking of the same spanning subgraph of $G_e$.

### 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 \( \#\phi \leq m \). 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 \( \phi \) such that \( \#\phi \leq m \). Then, by using a linear search over the range of \( m \), \( 1 \leq m \leq \Delta \log_2 n \), the minimum value of \( m \) is determined such that a spanning tree \( T \) of \( G \) has an edge-ranking \( \phi \) with \( m = \#\phi \) and find an optimal vertex ranking spanning tree \( \phi \).

**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 \( \phi \) 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 \( G \) 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 \( R(\phi) \) of \( \phi \) on \( x \) can be seen as an equivalence class of extensible edge-rankings of spanning subgraphs of \( G_y \) by Lemma 4. Since \( |R| = m \) and \( 0 \leq \text{count}(\phi, v, i) \leq 1 \) for an edge-ranking \( \phi \) and a rank \( i \in R \), the number of distinct visible-lists \( L(\phi, v) \) is at most \( 2^m \) for each vertex \( v \in V_x \). Furthermore \( |X_x| \leq (k + 1) \). Therefore, the number of distinct list-sets \( L(\phi) \) is at most \( 2^{m(k + 1)} \). On the other hand, the number of distinct functions \( \lambda_\phi: X_x \times X_x \rightarrow R \ Y \{0, \infty\} \) is at most \( (m + 2)^{k(k + 1)/2} \), since \( \lambda_\phi(v, v) = 0 \) and \( \lambda_\phi(v, w) = \lambda_\phi(w, v) \) for any \( v, w \in X_x \).

Therefore, the total number of different feasible vectors on \( x \) is at most \( 2^m(k + 1) \). \( (m + 2)^{k(k + 1)/2} \). One may assume that \( m \leq \Delta \log_2 n \) = \( O(\Delta \log_2 n) \) by Lemma 1. Therefore the total number of different feasible vectors on \( x \) is \( O(n^{\Delta (k + 1)/2} \log_n (k + 1)^2 n) \) for any fixed integer \( k \).

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

1. enumerate all edge-labelings \( \phi: E_x \rightarrow R \) of a spanning subgraph \( H'_x \) of \( G_x \);
2. compute all feasible vectors \( R(\phi) \) from the edge-labelings \( \phi \) of \( H'_x \); 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 \( |V_x| = |X_x| \leq k + 1 \) and \( |R| = m \), the number of edge-labelings \( \phi: E_x \rightarrow R \) is at most \( m^k \). For each edge-labeling \( \phi \), \( \lambda_\phi \) can be computed in time \( O(1) \). Furthermore, the visible-lists \( L(\phi, v) \), \( v \in X_x = V_x \), can be done by Lemma 3 in time \( O(1) \), and if so, computing \( L(\phi) \) can be done in time \( O(1) \). Therefore step (1) and (2) can be executed in time \( O(m^k) = O(\Delta^k \log^k n) \). Since the number of spanning subgraphs of \( G_x \) is a function of \( k \), step (3) can be executed for a leaf \( x \) in time \( O(\Delta^k \log^k n) \) and thus the table on \( x \) can be found in time \( O(\Delta^k \log^k n) \).

We next show how to compute all feasible vectors on an internal node \( x \) of \( T_b \) on 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 \( E_y = E_y' \cup E_z' \). Let \( \eta \) and \( \psi \) respectively be the edge-rankings of the spanning subgraphs \( H'_y \) and \( H'_z \) of \( G_y \) and \( G_z \).
respectively. $H^p_x$ be the resultant graph such that $H^p_x = H^p_y \cup H^p_z$ so obviously it is a subgraph of $G_x$.

First we have to check whether $H^p_x$ be a spanning subgraph of $G_x$ or not. If for any pair of vertices $(v, w) \in (X_1, X_2) \times (X_1, X_2)$, there are two different paths in $H^p_x$ then $H^p_x$ is not a spanning subgraph of $G_x$, so discard the vertex. But if this condition is false for all pairs $(v, w) \in (X_1, X_2) \times (X_1, X_2)$ then $H^p_x$ is spanning subgraph of $G_x$. This spanning subgraph checking can be done in time $O(1)$.

Now let $\varphi$ be the edge-labeling of $H^p_x$ extended from $\eta$ and $\psi$, then we have $\varphi | H^p_x = \eta$ and $\varphi | H^p_z = \psi$.

Now we show how to compute $\lambda_\varphi$ from vectors $R(\eta)$ and $R(\psi)$. Let $G(\eta)$ be an edge-$\lambda$-graph defined for $\eta$ as follows: let $K^{|x|}_{v}$ be a complete graph of the vertices in $X_v$; assign a weight of $\lambda_\eta(v, w)$ to each edge $(v, w)$ in $K^{|x|}_{v}$. 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-$\lambda$-graph $G(\psi)$ for $\psi$. Identify each pair of the same vertices in $(X_1, X_2)$, 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 $k(k+1)/2$. Then, by the construction of $G'(\varphi)$, the function $\lambda_\varphi: X_v \times X_v \rightarrow \mathbb{R}$ $\{0, \infty\}$ can be computed as follows:

$$\lambda_\varphi(v, w) = \min \{ \lambda | G'(\varphi) \text{ has a path P from v to w} \}$$

Since $G'(\varphi)$ has a constant number of vertices and edges, $\lambda_\varphi$ can be computed in time $O(1)$. It is also easy to construct a $\lambda$-graph $G(\varphi)$ from $G'(\varphi)$.

We next show how to compute $L(\varphi)$ from $\eta$ and $\psi$. Let $i \in R$ be any rank. Delete all the vertices with rank $> i$ from $H^p_x$. Among the connected components of the resulting graph, let $H^p_\varphi$ be the one containing a vertex $v \in X_v$. Then count($\varphi, v, i$) = $n(\varphi, v, i)$. Since $|E_i| = O(n)$ and $|R| = m = O(\Delta \log_2 n)$, the count-lists $L(\varphi, v)$, $v \in X_v$ can be computed in time $O(nm) = O(n \Delta \log_2 n)$. Then checking whether an edge-labeling $\varphi$ is an edge-ranking of $H^p_\varphi$ can be done by Lemma 3 in time $O(\Delta \log_2 n)$, and if so, computing $L(\varphi)$ can be done in time $O(\Delta \log_2 n)$. 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 $O(n^2 \Delta (k+1)^2 \Delta^{k+1} \log_2 (k+1) n)$. Therefore the table on $x$ can be computed in time $O(n^2 \Delta (k+1)^2 \Delta^{k+1} \log_2 (k+1) n)$. 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 $O(n^2 \Delta (k+1)^2 \Delta^{k+1} \log_2 (k+1)^2 n)$ time. And then checking whether an edge-ranking $\varphi$ is a valid solution for one-tree type spanning subgraph $H^p_\varphi$ 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 $H^p_\varphi$ is connected, which implies that $H^p_\varphi$ is a one tree-type spanning subgraph of $G$. Consequently, we can say $G$ has a vertex-ranking spanning tree $\varphi$. 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 $O(n^2 \Delta (k+1)^2 \Delta^{k+1} \log_2 (k+1)^2 n)$ where $n$ is the number of vertices and $\Delta$ 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 $O(\Delta^{k+1} \log_2 n)$ time. Since there are $O(n)$ leaves, line 2 can be done in $O(n \Delta^{k+1} \log_2 n)$ time in total for all leaves. Since line 3 is executed for $(\text{node})$ nodes in total in line 4, line 4 can be done in $O(n^2 \Delta (k+1)^2 \Delta^{k+1} \log_2 (k+1)^2 n)$ time in total. At last Line 5 can be done in $O(n^2 \Delta (k+1)^2 \Delta^{k+1} \log_2 (k+1)^2 n)$ time in total. Thus checking whether a
spanning tree of a partial $k$-tree $G$ has an edge-ranking $\phi$ such that $\#\phi \leq m$ can be done in $O(n^{2}\Delta(k+1)^{2}\Delta(k+1)^{1}\log k(k+1)^{1}n)$ time.

Using the linear search technique over the range of $m$, $1 \leq m \leq \Delta \log n$, one can find the smallest integer $X(T)$ such that $T$ has an edge-ranking $\phi$ with $\#\phi = X(T)$ by calling the algorithm edge ranking $O(\Delta \log n)$ 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


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.