Minimum Edge Ranking Spanning Tree Problem on Interval Graphs

Ruei-Yuan Chang and Sheng-Lung Peng*
Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien 974, Taiwan

Abstract
The minimum edge ranking spanning tree problem on graph \( G \) is to find a spanning tree \( T \) of \( G \) such that the minimum edge ranking of \( T \) is minimum among all possible spanning trees of \( G \). In this paper, we propose a linear-time algorithm for this problem on interval graphs.

1 Introduction
Let \( G = (V, E) \) be a finite, simple, and undirected graph. Let \( n = |V| \) and \( m = |E| \). Let \( N(v) = \{ u \mid (u, v) \in E \} \). \( P = (v_1, v_2, \ldots, v_s) \) is a path of \( G \) if \( (v_i, v_{i+1}) \in E(G) \).

An edge ranking of a graph \( G \) is a labeling (ranking) of the edges of \( G \) with positive integers such that every path in \( G \) between any two edges of the same label \( i \) contains at least one edge on the path with label \( j > i \). The integer label of an edge is called the rank of the edge. An edge ranking is minimum if the largest rank \( k \) in it is the smallest among all possible edge rankings of \( G \) and is denoted by \( \gamma(G) \). The minimum edge ranking problem is to find a minimum edge ranking of a given graph \( G \). Lam et al. proved that this problem is NP-hard on general graphs [2]. They also presented a linear-time algorithm for trees [3].

In this paper, we study the related problem of the minimum edge ranking problem. Makino et al. introduced the minimum edge ranking spanning tree problem (MERST problem for short) [4]. MERST problem is to find a spanning tree \( T \) of \( G \) with \( \gamma(T) \) being minimum among all possible spanning trees of \( G \). In this paper, we denote \( T \) to be the MERST of \( G \) for convenience. We denote \( \gamma_E(G) = \gamma(T) \) to be the rank of MERST of \( G \). Makino et al. showed that MERST problem is NP-hard on general graphs [4] and they showed that MERST problem can be solved in polynomial time on split graphs [6] and in linear time on threshold graphs [5]. Recently, Arefin et al. proved the MERST problem on series-parallel graphs is NP-complete [1]. The MERST problem has a number of practical applications in many fields such as parallel processing, integration of distributed data, and so on [4].

There is a vertex version of MERST problem, namely, the minimum vertex ranking spanning tree problem (MVRST problem for short). The definition of the MVRST problem is similar to the MERST problem. MVRST problem is to find a spanning tree of \( G \) whose vertex ranking is minimum among all possible spanning trees of \( G \). Due to the difference between the definitions of vertex ranking and edge ranking, the MERST and MVRST of \( G \) seem to be as a path and a star respectively as possible. Unfortunately, there is not always a path with \( n \) vertices in a graph, so we still need to find a spanning tree which is not a path. Since the maximum degree of \( G \) decides the lower bound of edge ranking of the spanning tree, intuitively, we should keep the maximum degree as few as possible. That is, for each subtree, we should keep it as a path as we can and do not connect too many subtrees to one vertex unless necessary. Thus, to solve the MERST problem, we first try to find the fewest disjoint paths whose vertex union is equal to \( V(G) \). If the number of paths is equal to 1, then we obtain the vertex-disjoint longest path with order \( n \). If not, we are going to determine how to connect these paths to obtain a spanning tree with smallest edge ranking. That is, we have to find the fewest disjoint paths whose vertex union is equal to \( V(G) \). Based on this idea, we propose a linear-time algorithm for MERST problem on interval graphs.

*Corresponding author: slpeng@mail.ndhu.edu.tw
2 Interval Graphs

A graph $G$ is called an interval graph if it has an intersection model $F$ consisting of intervals on a straight line such that any vertex of $G$ corresponds to an interval in $F$, and two vertices are adjacent if their corresponding intervals overlap. Figure 1 gives an example of an interval graph.

![Interval Graph Example](image)

**Figure 1:** An interval graph $G$.

Ramalingam et al. showed that the vertices of an interval graph can be numbered such that the property listed in the following lemma holds.

**Lemma 1.** [7] The vertices can be numbered with integers $1, 2, \ldots, |V(G)|$ such that if $i < j < k$ and $(v_i, v_k) \in E(G)$, then $(v_j, v_k) \in E(G)$.

In this paper, we call such ordering as right-end ordering of $G$ and denote it by $\pi = (v_1, v_2, \ldots, v_n)$. It is well-known that $\pi$ can be obtained in $O(n + m)$. For example, the vertices in Figure 1 are numbered according to the order of their right ends.

By the definition of edge ranking, we can easily obtain the lower bound and upper bound of $\gamma(T)$ which is shown in the following lemma.

**Lemma 2.** Let $G$ be an interval graph and $T$ be a MERST of $G$. Then $|\log n| \leq \gamma(T) \leq n - 1$.

In [3], Lam et al. defined that an edge ranking $\phi$ is said to be critical if $L(\phi) \leq L(\phi')$ for any edge ranking $\phi'$ of $T$ where $L(\phi)$ is the visible labels of $T$ under $\phi$. A critical edge ranking must be an optimal edge ranking. Torre et al. give the following lemma.

**Lemma 3.** [8] A critical edge ranking of $T$ can be formed by any critical edge rankings of $T_1, T_2, \ldots, T_d$ together with a suitable labeling of branches.

Based on this lemma, Lam et al. present a linear-time algorithm. We briefly describe this algorithm. This algorithm computes edge ranking from bottom to top. During processing, it updates the visible labels for roots of all subtrees and then computes the optimal edge ranking by the visible labels. The rank of the root of a subtree is larger if the lager visible label is seen by the root. By this algorithm, we get the idea that we should keep the subtree of $T$ optimal during constructing. We try to do this by scanning $\pi$ from $v_1$ to $v_n$. At beginning, we choose $v_1$ as the starting vertex. Then we choose $\min \{x : (v_x, v_1) \in E(G)\}$ to connect. Next we connect $v_x$ to $\min \{y : (v_y, v_x) \in E(G)\}$. Continue this operation until $\min \{v_s : (v_s, v_t) \in E(G)\}$ is empty where $v_t$ is the tail vertex of the path $P_1$. Then we start a new path $P_2$ from $\min \{v_z : v_z \in V(G) \setminus P_1\}$. The following lemma shows the vertex to start the next path.

**Lemma 4.** Let $P_t = (v_{s_1}, \ldots, v_{s_t})$ be the path found by the operation described above. Then $v_{t+1}$, $t \neq n$, is the vertex $\min \{v_z : v_z \in V(G) \setminus P_1 \cup \ldots \cup P_t\}$.

**Proof.** Let $v_r = \min \{v_z : v_z \in V(G) \setminus P_1 \cup \ldots \cup P_t\}$. Suppose that $v_{t+1} \neq v_r$. Then there are two cases. The first one is $t + 1 < r$. This means, $v_{t+1}$ is connected by some vertex $v_w \in P_1 \cup \ldots \cup P_t$. Clearly, $w \neq t$. If $w < t$, then $(v_{t+1}, v_w) \in E(G)$ by Lemma 1. This is a contradiction that we should connect $v_w$ to $v_{t+1}$ since $t + 1 < r$. It is impossible that $w > t + 1$ since $v_t$ is the tail of $P_t$. Considering the second case $t + 1 > r$. This means $v_r$ is abort during connecting $P_t$. We may suppose $r < i$ and $(v_w, v_r), (v_w, v_i) \in E(G)$ where $v_w, v_i \in P_1$. Obviously, it causes a contradiction to the operation of connecting path since $r < i$ but $v_w$ connects to $v_i$. Hence $v_{t+1} = v_r$. \(\square\)

In our algorithm, we try to obtain $T$ by scanning $\pi$ once. According to Lemma 4, we can start connecting new path soon by accessing the tail of previous path. But, an important question is how to connect the paths to be an optimal minimum edge ranking spanning tree of the induced subgraph of $G$. Intuitively, we should connect the vertex nearest the tail of previous path to the vertex nearest the head of next path as possible. This will make this tree having the longest path and fewest branches. But note that, there might be no edge between $P_t$ and $P_{t+1}$. This is because there is a case that $\max \{x : v_x \in P_t\}$ is greater than $\max \{y : v_y \in P_t\}$ and $\max \{x : v_x \in P_{t+1}\}$. Therefore, we have to keep $v_x$ until the current connecting path $P_x$ with rightmost vertex $v_w$ where $w > x$. 


We denote such vertex by \( v_{pre} \). Then the paths whose rightmost vertices are smaller than \( v_{pre} \) in \( \pi \) can connect to \( v_{pre} \). But, sometimes these paths can connect to not only \( v_{pre} \) but also other vertices in the path \( P_i \) containing \( v_{pre} \). When we meet such situation, we keep these paths whose rightmost vertices are smaller than \( v_{pre} \) in \( \pi \) and connect them as equally as possible.

**Lemma 5.** Let \( S \) be a tree and \( S' \) be a subtree of \( S \) with \( x \) vertices. Then \( \gamma(S) \) is smallest when \( S' \) is a path.

**Proof.** Let \( S_p \) tree with \( S' \) is a path and \( S_o \) otherwise. Let \( r \in S \setminus S' \) be the vertex \( S' \) connect to. Let \( r' \in S' \) be the vertex of \( S' \). It is obvious that if \( S' \) is a path, then \( \gamma(S') = [\log x] \) and \( \gamma(S') > [\log x] \) otherwise. Clearly, the rank of \((r, r')\) must be at least more than \( \gamma(S') \) to satisfy the critical edge ranking. Therefore, \( \gamma(S_p) \leq \gamma(S_o) \).

Clearly, the operation that connect \( v_{pre} \) to the vertex nearest the head of next path will make the result tree with the longest path and fewest branches. To speed up this operation, we keep the candidate vertex in previous path, i.e. \( v_{pre} \). \( v_{pre} \) is the vertex nearest the tail of previous path and can connect to the next path. During connecting the previous path, if we connect \( v_a \) to \( v_b \) where \( a > b \), then \( v_{pre} = v_b \). Note that we only keep one vertex, that is, \( v_{pre} \) is updated while this condition hold. Suppose \( v_{pre} \in P_i \). Due to the property of interval graph, since Lemma 4 showed us the head of the next path is \( v_{i+1} \) where \( v_{i+1} \) is the tail of the previous path, it is clear that \((v_{i+1}, v_{pre}) \in E(G)\). Thus the operation that connecting previous path to next path can be solved in \( O(1) \).

**Lemma 6.** Let \( P_j \) and \( P_{j+1} \) be the paths found by the operation described above. Suppose \( |N(P_j) \cap P_{j+1}| > 1 \) and \( N(P_j) \cap P_{j+1} = \phi \) for \( j < k \leq s \). When connecting \( P_j \) to \( P_{j+1} \), we connect to \( P_i \) such that \( \min_k \{ P_{i+1} \mid P_{i+1} \in N(P_i) \} \), choosing the vertex in \( P_{i+1} \) which is nearest to the head of \( P_{i+1} \) leads to the longest path in \( T_p \).

After connecting all paths \( P_1, \ldots, P_s \) and there is no remaining vertices that are not in \( P_1 \cup \ldots \cup P_s \), we obtain an MERST \( T \). It is obvious that \( T \) contains the longest path among all spanning trees. Summarizing the above operations, we give our algorithm, MERST-Interval.

**Theorem 1.** Algorithm MERST-Interval finds an MERST \( T \) of \( G \) in \( O(n + m) \) time.

**Algorithm MERST-Interval**

**Data:** the right-end ordering \( \pi \) of \( G \).

**Result:** an MERST \( T \) of \( G \).

```plaintext
for \( v_i \in V \) do
  cover[i] = 0;
num = 1;
P_num = (v_1);
V = V \{v_1\};
v_{pre} = v_{premp} = \phi;
T = U = \phi;
while \( V \neq \phi \) do
  Let the tail vertex of \( P_{num} \) be \( v_t \);
  S = \{v_k \mid (v_k, v_k) \in E, cover[k] = 0\};
  if \( S \neq \phi \) then
    \( v_m = \min S; \)
    Add \( v_m \) to the tail of \( P_{num} \);
    if \( t > m \) then
      \( v_{premp} = v_t; \)
  else
    \( v_m = \min V; \)
    num = num + 1;
P_{num} = (v_m);
  if \( v_{pre} < v_{premp} \) then
    if \( U \neq \phi \) then
      forall the \( P_i \in U \) do
        Let \( v_{head} = \min P_i; \)
        Add \( P_i \) to \( T \) by connecting \( v_{head} \) to \( \min_{degree} \{ N(v_{head}) \cap P_{num-1} \} \};
        U = \phi;
        Add \( P_{num-1} \) to \( T \) by connecting \( v_{pre} \) to \( v_{premp} \);
      \( v_{pre} = v_{premp} \);
    else
      \( U = U \cup P_{num-1}; \)
  V = V \{v_m\};
cover[m] = 1;
return T;
```

202
3 Conclusion

In this paper, we present an $O(n + m)$-time algorithm to solve the MERST problem on interval graphs. By using the right-end ordering, we obtain the MERST efficiently. The detail will be shown in journal version. It is interesting that if the permutation graphs can be solve similarly since the permutation graphs can be represented by a permutation of vertices.

References


