Dynamic Steiner Tree Problem

Makoto Imase and Bernard M. Waxman

This paper proposes a new problem, which we call the Dynamic Steiner Tree Problem. This is related to multipoint connection routing in communications networks, where the set of nodes to be connected changes over time. This problem can be divided into two cases, one in which rearrangement of existing routes is not allowed and a second in which rearrangement is allowed. In the first case, we show that there is no algorithm whose worst error ratio is less than $1/2 \log n$ where $n$ is the number of nodes to be connected. In the second case, we present an...

Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Recommended Citation

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.
Dynamic Steiner Tree Problem

Makoto Imase and Bernard M. Waxman

Complete Abstract:

This paper proposes a new problem, which we call the Dynamic Steiner Tree Problem. This is related to multipoint connection routing in communications networks, where the set of nodes to be connected changes over time. This problem can be divided into two cases, one in which rearrangement of existing routes is not allowed and a second in which rearrangement is allowed. In the first case, we show that there is no algorithm whose worst error ratio is less than 1/2 log n where n is the number of nodes to be connected. In the second case, we present an algorithm whose error rate is bounded by a constant and rearrangement is relatively small.
DYNAMIC STEINER TREE PROBLEM

Makoto Imase
Bernard M. Waxman

WUCS-89-11

March 1989

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

ABSTRACT

This paper proposes a new problem, which we call the Dynamic Steiner Tree Problem. This is related to multipoint connection routing in communications networks, where the set of nodes to be connected changes over time. This problem can be divided into two cases, one in which rearrangement of existing routes is not allowed and a second in which rearrangement is allowed. In the first case, we show that there is no algorithm whose worst error ratio is less than $\frac{1}{2} \log n$ where $n$ is the number of nodes to be connected. In the second case, we present an algorithm whose error ratio is bounded by a constant and rearrangement is relatively small.

Makoto Imase is with NTT Software Laboratories. This work described here was performed while on leave at Washington University.

This work supported by Bell Communications Research, Italtel SIT, NEC and National Science Foundation (DCI-8600947).
Dynamic Steiner Tree Problem

Makoto Imase and Bernard M. Waxman

1. Introduction

With the growth of interest in flexible multipoint communications networks for supporting a wide class of applications, the importance of routing technique for multipoint connections is being emphasized [1,2]. Routing a multipoint connection is typically treated as the problem of finding the shortest subtree of the network containing a set of nodes, called a terminal node set. If the terminal node set is known in advance and does not change, this problem is a classical problem in graph theory, the Steiner Tree Problem (ST), which has been studied extensively [3,4] including the implementation of a distributed algorithm [5].

In order to support some services, for example video broadcasts and multi-person conferences [1], we need facilities for adapting to changes in the terminal node set. There are relatively few studies dealing with this problem [6] in spite of its practical importance.

This problem, which we call the Dynamic Steiner Tree Problem (DST), comes in two flavors, one in which rearrangement of existing routes is not allowed and a second in which rearrangement is allowed. In the second case, we consider the number of rearrangements in addition to the cost of a generated tree.

This paper considers algorithms for DST mainly focusing on the worst case error ratio of the generated trees relative to minimum Steiner trees \((T_{OPT})\). After the formal definitions of these problems in Section 2, Section 3 shows that in the non-rearrangeable case there is no algorithm whose worst error ratio is less than \(\frac{3}{2} \log n\), where \(n\) is the cardinality of a terminal node set. Section 4 proposes an algorithm for the rearrangeable case whose error ratio is bounded by a constant with relatively little rearrangement.

2. Definition

In DST we are given a graph \(G = (V, E)\), a cost function \(\text{cost} : E \rightarrow \mathbb{R}^+\) and a sequence of requests \(R = \{r_1, r_2, \ldots, r_k, \ldots, r_K\}\), where each \(r_k\) is a pair \((v_k, \rho_k)\),
$v_k \in V, \rho_k \in \{\text{add, remove}\}$. Each request is to add a node to, or to remove a node from, a connection. We let $S_k =$

$$\{v_i | (v_i, \text{add}) = r_i \text{ for some } i \ (0 \leq i \leq k), \ (v_i, \text{remove}) \neq r_j \text{ for any } j \ (i < j \leq k)\},$$

and refer to $S_k$ as a terminal node set at Step $k$.

The object is to find a minimum cost tree connecting terminal nodes in $S_k$ for each $k$ without knowledge of $r_j$ for any $j > k$. This problem can be divided into two cases. In the first case, once a particular set of edges has been used in a route, no rearrangement is allowed as the algorithms proceeds. In the second case, rearrangement is allowed.

**Problem 1 (DTS-N)** Given an instance $(G, \text{cost}, R)$, find a sequence of trees, $\{T_1, T_2, \ldots, T_K\}$, satisfying the following conditions and minimize some cost function, for example $\sum_k \text{cost}(T_k)$:

1. Each $T_k$ spans $S_k$.
2. If $r_k$ is an add request, $T_k$ includes $T_{k-1}$ as a subgraph.
3. If $r_k$ is a remove request, $T_{k-1}$ includes $T_k$.

Conditions 2 and 3 imply that edges and nodes are added to a tree only for an add request and removed only for a remove request.

In the rearrangeable case, we also minimize the rearrangements to get $T_k$ from $T_{k-1}$. In other words, we want to find an algorithm minimizing the cost of generated trees and the modification. Two operations, a primitive path deletion and a primitive path insertion, which correspond to point-to-point call connection and deletion, are defined to measure the number of rearrangements. In deleting a subgraph $p$ from $T$, if $p$ is a (simple) path and every intermediate node in it is neither a terminal node nor a Steiner node, the operation is called primitive path deletion. A Steiner node is a node in $T$ having degree larger than 2. In adding a subgraph $p$ of $G$ to $T$, if $p$ is a path and every intermediate node in it is not contained in $T$, the operation is called a primitive path insertion. Any modification to $T$ is restricted to primitive path deletions and insertions. Then the number of rearrangements $\alpha_k$ at step $k$ is defined as follows:

- If $r_k$ is an add request, $\alpha_k$ is the number of primitive path deletions necessary to get $T_k$ from $T_{k-1}$.
- If $r_k$ is an remove request, $\alpha_k$ is the number of primitive path insertions necessary to get $T_k$ from $T_{k-1}$.

**Problem 2 (DST-R)** Given an instance $(G, \text{cost}, R)$, find a sequence of trees, $\{T_k\}$ ($k = 1, 2, \ldots, K$) where each $T_k$ spans $S_k$ and minimize some cost function and some function of the $\{\alpha_k\}$, for example $\sum \alpha_k$. 

2
3. Non-rearrangeable Case

We first consider the worst case performance in terms of the error ratio for DST-N restricting our attention to the case where each request is an operation.

We begin by defining graphs $G_{\kappa}$, $\kappa \in \mathbb{Z}_0^+$ and a constant cost function $c_{\kappa}$ on the edges of $G_{\kappa}$. $G_0 = K_2$, the complete graph with two nodes, where its single edge has a cost of 1. We name the two nodes $v_{0,0}$ and $v_{0,1}$ and refer to them as level 0 nodes. Graph $G_{\kappa}$ for $\kappa > 0$ is defined using graph $G_{\kappa-1}$. For each edge $(u, v)$ in $G_{\kappa-1}$ we introduce a pair of nodes $\alpha, \beta$ and replace $(u, v)$ with two paths $(u, \alpha, v)$ and $(u, \beta, v)$. We refer to the nodes $\alpha$ and $\beta$ as level $\kappa$ sister nodes. Each edge in $G_{\kappa}$ is then assigned a cost of $2^{-\kappa}$. Note that $v_{0,0}$ and $v_{0,1}$ will be connected by (simple) paths of cost 1. (See Fig. 1.)

![Diagram of graphs](image)

**Figure 1:** Example of $G_8$

We say that two nodes $u, v \in V(G_{\kappa})$ are $i$-adjacent, $0 \leq i \leq \kappa$ if the level of both $u$ and $v$ is no more than $i$ and there is a path from $u$ to $v$ which has no intermediate node from level $j$, $j \leq i$. That is the corresponding nodes in graph $G_i$ would actually be adjacent. Note that exactly one of the two $i$-adjacent nodes must be a level $i$ node and that the distance between $i$-adjacent nodes is $2^{-i}$.

Lemma 3.1 In graph $G_i$, $i > 1$,

i. each level $i$ node $\alpha$ is adjacent to exactly two nodes, node $v$ at level $i - 1$ and node $u$ at level $j$, $0 \leq j < i - 1$. In addition node $\alpha$ has a sister node $\beta$ at level $i$ which is also adjacent to both $u$ and $v$. 

3
ii. each level $i-1$ node $v$ is adjacent to exactly four level $i$ nodes, consisting of two sister pairs.

Thus, in graph $G_\kappa$, $\kappa \geq i$, (i) and (ii) hold if the term adjacent is replaced by $i$-adjacent.

Proof This lemma follows by a simple induction on $i$ based on the recursive definition of $G_i$. Figure 2 illustrates the properties described here for two sister pairs $\alpha, \beta$ and $\alpha', \beta'$ in graph $G_i$.

A sequence of node sets $N = \{N_0, N_1, \ldots, N_\kappa\}$ is called a $\kappa$-sequence for graph $G_\kappa$ if there exists a path $p$ from $v_{0,0}$ to $v_{0,1}$ such that $N_i$ is the set of level $i$ nodes in path $p$. Note that $N_0 = \{v_{0,0}, v_{0,1}\}$, that $|N_i| = 2^{i-1}$ for $0 < i \leq k$ and that $\bigcup_{i=0}^{\kappa} N_i = V(p)$, the set of nodes in path $p$. We now define a minimal tree sequence $\hat{T} = \{\hat{T}_0, \hat{T}_1, \ldots, \hat{T}_\kappa\}$ for graph $G_\kappa$, with respect to a $\kappa$-sequence $N$. $\hat{T}_0$ is any tree that spans the nodes in $N_0$ such that no proper subgraph of $\hat{T}_0$ also spans $N_0$. $\hat{T}_i$, $0 < i \leq \kappa$ must contain tree $\hat{T}_{i-1}$ as a subgraph and all of the nodes in $N_i$ from the $\kappa$-sequence $N$. The requirement that $\hat{T}_i$ is minimal means that no subgraph of $\hat{T}_i$ also satisfies these properties.

In the next lemma we consider a minimal tree sequence $\{\hat{T}_i\}$ constructed by an algorithm, which generates each $\hat{T}_i$ based only on knowledge of $\hat{T}_{i-1}$ and $N_i$. We choose the terminal nodes for each $N_i$ based on $\hat{T}_{i-1}$, $i > 0$ in order to make the cost of each $\hat{T}_i$ as large as possible. Restricting $N$ to a $\kappa$-sequence insures that there exists a minimum Steiner tree for the entire terminal set with cost 1.

Lemma 3.2 Given a graph $G_\kappa$, $\kappa \in \mathbb{Z}_0^+$, and any algorithm $A$ which constructs a minimal tree sequence, where each tree $\hat{T}_i$ is generated based only on knowledge of
\( \hat{T}_{i-1} \) and \( N_i \), there exists a \( \kappa \)-sequence \( N \) such that the minimal tree sequence for \( N \) satisfies the inequality
\[
\text{cost}(\hat{T}_i) \geq 1 + \frac{1}{2^i}
\]
for all \( i, 0 \leq i \leq \kappa \).

**Proof:** We prove this lemma for each \( \hat{T}_i \), \( 0 \leq i \leq \kappa \) by induction on \( i \). Inequality (1) clearly holds for \( i = 0 \) since the distance from \( v_{0,0} \) to \( v_{0,1} \) in \( G_0 \) is 1. If we choose \( N_1 \) so that it contains the level 1 node not in \( \hat{T}_0 \) then \( \text{cost}(\hat{T}_1) \geq 1.5 \). Note that there is a path of cost 1 in \( G_\kappa \) that contains all nodes in \( N_0 \) and \( N_1 \), so that \( N_0, N_1 \) is an initial segment of some \( \kappa \)-sequence.

Let us now assume that (1) holds for every \( j \), \( 0 \leq j < i \) where \( 1 < i \leq \kappa \) and that \( N_0, N_1, \ldots, N_{i-1} \) is an initial segment for some \( \kappa \)-sequence. Since \( \hat{T}_{i-1} \) is a minimal tree each leaf node must be in one of the \( N_j \) and there are no cycles in \( \hat{T}_{i-1} \). Consider a node \( v \) from \( N_{i-1} \). Node \( v \) is \( i \)-adjacent to exactly four nodes at level \( i \) consisting of two sister pairs by Lemma 3.1 (ii). If \( \hat{T}_{i-1} \) contains both nodes of a level \( i \) sister pair then \( \hat{T}_{i-1} \) has a cycle or a leaf node at a level greater than \( i - 1 \). But this contradicts the minimality of \( \hat{T}_{i-1} \), hence \( \hat{T}_{i-1} \) can contain at most one of the nodes from each sister pair adjacent to \( v \).

For each node \( v \in N_{i-1} \) we select one node from each sister pair of \( v \), that is not in \( \hat{T}_{i-1} \), to place in \( N_i \). Note that if there is a path of cost 1 which contains all nodes from every \( N_j \), \( 0 \leq j < i \) then there is a path of cost 1 which also contains the nodes in \( N_i \). The cost of a shortest path from each of the nodes in \( N_i \) to a node in \( T_{i-1} \) will be \( 2^{-i} \) since each is \( i \)-adjacent to a node in \( N_{i-1} \) and every leaf node of \( \hat{T}_{i-1} \) must be at a level no greater than \( i - 1 \). All the nodes we select for \( N_i \) are distinct since each is \( i \)-adjacent to only one node at level \( i - 1 \) by Lemma 3.1 (i). Since \( |N_{i-1}| = 2^{i-2} \) we will have selected \( 2^{i-1} \) level \( i \) nodes to include in \( N_i \), and \( N_i \) will contain all level \( i \) nodes along a path from \( v_{0,0} \) to \( v_{0,1} \). Thus, it follows that \( \text{cost}(\hat{T}_i) \geq \text{cost}(\hat{T}_{i-1}) + \frac{1}{2} \) and that \( N_0, N_1, \ldots, N_i \) is an initial segment for some \( \kappa \)-sequence.

Using Lemma 3.2 we derive a lower bound for the best possible worst case error ratio given any algorithm for DST-N.

**Theorem 3.1** Given any algorithm \( A \) for DST-N there is an instance of the problem \( (G, \text{cost}, R) \) such that for all \( i, 0 < i \leq K \)
\[
\frac{\text{cost}(T_i)}{\text{cost}(T_{opt})} \geq 1 + \frac{1}{2} \left[ \log(n_i - 1) \right]
\]
where \( \{T_i, 0 \leq i \leq K\} \) is the sequence of trees generated by algorithm \( A \) and \( n_i = |S_i| \). Here \( T_{OPT} \) is a minimum Steiner tree for \( S_i \). Furthermore this bound holds even if each request is restricted to node addition.
Proof: We consider an instance of the dynamic Steiner problem \((G, c, R)\) where \(R = \{(u_i, \text{add}), 0 \leq i \leq 2^k\}\), all \(u_i\) are distinct with \(u_0 = v_{0,0}\) and for \(i > 0\) \(u_i \in N_j, j = \lfloor \log i \rfloor\) for a \(\kappa\)-sequence \(N\). At steps \(i, 2^j \leq i < \min(2^{j+1}, 2^\kappa + 1)\) for \(0 \leq j \leq \kappa\) the solution, generated by any algorithm \(A\), must contain some minimal tree \(T_j\). Note that a minimum Steiner tree for nodes in the sequence \(N\) has cost 1 by the definition of a \(\kappa\)-sequence. By Lemma 3.2 we can construct a \(\kappa\)-sequence so that \(\text{cost}(T_j) \geq 1 + 1/2j\). This gives us a lower bound on the worst case performance of any algorithm without rearrangement and the theorem follows.

We present a simple dynamic greedy algorithm (DGA) for DST-N which has performance within two times the best possible bound in the case where each request is restricted to node addition. For each add request DGA joins the new node by a shortest path to a nearest node already in the connection. In the case of a remove request a terminal node is dropped by simply deleting the portion of the connection which serves only that terminal node. See Fig. 3 for complete details.

\[
T_0 := \{(v_0), \varnothing\}; S_0 := \{v_0\}; k = 1;
\]

\[\text{do } k \leq K \rightarrow\]

\[\quad\text{if } r_k \text{ is an add request } \rightarrow\]

\[\quad\quad\text{Choose the shortest path } p_k \text{ from } v_k \text{ to } T_{k-1};\]

\[\quad\quad T_k := T_{k-1} \cup p_k;\]

\[\quad\quad S_k := S_{k-1} \cup \{v_k\} \text{ and increment } k \text{ by one;}\]

\[\quad\text{if } r_k \text{ is a remove request } \rightarrow\]

\[\quad\quad S_k := S_{k-1} - \{v_k\};\]

\[\quad\quad T_k := T_{k-1};\]

\[\quad\text{do } V(T_k) - S_k \text{ contains node } w \text{ with degree } 1 \rightarrow\]

\[\quad\quad T_k := T_k - w;\]

\[\quad\text{od}\]

\[\text{od}\]

Figure 3: Dynamic Greedy Algorithm (DGA)

Theorem 3.2 For any instance \((G, c, R)\) of DST-N let \(\{T_k, 0 \leq k \leq K\}\) be a sequence of trees generated by DGA and let \(n_k = |S_k|\). Then if each \(r_k\) is an add request

\[
\frac{\text{cost}(T_k)}{\text{cost}(T_{opt})} \leq \log(n_k)
\]

for \(0 < k \leq K\).

Proof: Let \(v_0, v_{a_1}, \ldots, v_{a_i}, \ldots, v_{a_k}\) be a preordering of the tree \(T_{opt}\) for the node set \(S_k\) with root \(v_0\), that is, we list a node the first time we visit it during a depth-first
traversal of $T_{opt}$ from $v_0$. First construct a new graph $L_k$ which consists of a path from $v_0$ to $v_0'$ joining $v_{\alpha_i}$ and $v_{\alpha_{i+1}}$ for $0 \leq i < k$ by an edge with cost $\text{dis}(v_{\alpha_i}, v_{\alpha_{i+1}}; T_{opt})$. In the case of $i = k$, join $v_{\alpha_i}$ and $v_0$ with cost $\text{dis}(v_{\alpha_i}, v_0; T_{opt})$. Then the cost of this path, $\text{cost}(L_k)$, is equal to twice of the cost of $T_{opt}$, because the depth-first traversal visits every edge exactly twice.

In $L_k$, let $l_{v_i}$ be the distance from $v_i$ to the nearest node whose subscript is less than $i$, that is,

$$l_{v_i} := \min_{j < i} \text{dis}(v_i, v_j; L_k).$$

Since $\text{dis}(v_i, v_j; G) \leq \text{dis}(v_i, v_j; T_{opt}) = \text{dis}(v_i, v_j; L_k)$ and the node $v_j$ is contained in $S_{i-1}$,

$$\text{cost}(p_i) \leq l_{v_i} \quad \text{for} \quad 1 \leq i \leq k$$

where $p_i$ is the path selected by DGA to join $v_i$ to tree $T_{i-1}$. If we show that

$$\sum_{i=1}^{k} l_{v_i} \leq \frac{\lg(k + 1)}{2} \text{cost}(L_k)$$

(4)

the proof will be completed because $n_i = k + 1$ and

$$\text{cost}(T_k) = \sum_{i=1}^{k} \text{cost}(p_i) \leq \sum_{i=1}^{k} l_{v_i} \quad \text{and} \quad \text{cost}(L_k) = 2\text{cost}(T_{opt}).$$

In the remainder of the proof we show that (4) holds for any path $L_k$.

Given nodes, $v_i$ and $v_j$, the path between them in $L_k$ is denoted by $L(v_i, v_j)$. Let $\text{sum}(L(v_i, v_j))$ be the summation of $l_\alpha$ where $\alpha$ is an intermediate node in $L(v_i, v_j)$. If the subscript of every intermediate node is larger than $i$ and $j$, this path is called regular. Note that $L_k$ is also denoted by $L(v_0, v'_0)$ and is regular. Let $v_l$ be the intermediate node in $L(v_i, v_j)$ with the smallest subscript. If $L(v_i, v_j)$ is regular, $l_{v_l} = \min\{\text{dis}(v_i, v_l), \text{dis}(v_l, v_j)\}$. Furthermore both $L(v_i, v_l)$ and $L(v_l, v_j)$ are regular.

Thus, for a regular path $L(v_i, v_j)$, the summation $\text{sum}(L(v_i, v_j))$ is calculated by a recursive function as follows:

$$\text{sum}(L(v_i, v_j)) = \begin{cases} 
\min\{\text{dis}(v_i, v_l), \text{dis}(v_l, v_j)\} + \text{sum}(L(v_i, v_l)) + \text{sum}(L(v_l, v_j)) & \text{if } v_l \in L(v_i, v_j) \\
0 & \text{if } v_l \notin L(v_i, v_j)
\end{cases}$$

(5)

where $v_l$ is the intermediate node with the smallest subscript.

Let $L(v_i, v_j)$ be a regular path whose end nodes are $v_i$ and $v_j$ and let $m$ be the number of intermediate nodes. The following inequality will be shown by induction on $m$,

$$\text{sum}(L(v_i, v_j)) \leq \frac{\lg(m + 1)}{2} \text{dis}(v_i, v_j)$$

(6)

which implies (4).
1. $m = 0$

   From the second case of Eq. 5, it is clear that (6) is valid.

2. $m > 0$

   Assume for any regular path with $m'$ intermediate nodes ($m' < m$), (6) holds.
   Let $v_i$ be the intermediate node whose subscript is less than that of any other
   intermediate node and let $x$, $x_1$ and $x_2$ be $\text{dis}(v_i, v_j)$, $\text{dis}(v_i, v_i)$ and $\text{dis}(v_i, v_j)$,
   respectively, then $x = x_1 + x_2$. From Eq. 5,
   \[
   \text{sum}(L(v_i, v_j)) = \min\{x_1, x_2\} + \text{sum}(L(v_i, v_i)) + \text{sum}(L(v_i, v_j))
   \]
   (7)

   As we can assume $x_1 \leq x_2$ without loss of generality,
   \[
   \text{sum}(L(v_i, v_j)) = x_1 + \text{sum}(L(v_i, v_i)) + \text{sum}(L(v_i, v_j)).
   \]
   (8)

   Let $m_1$ and $m_2$ be the number of intermediate nodes in $L(v_i, v_i)$ and $L(v_i, v_j)$,
   respectively, then $m_1 + m_2 = m - 1$, which implies $m_1 < m$ and $m_2 < m$. By
   the inductive hypothesis and Eq. (8)
   \[
   \text{sum}(L(v_i, v_j)) \leq x_1 + \frac{\lg(m_1 + 1)}{2} x_1 + \frac{\lg(m_2 + 1)}{2} x_2
   \]
   \[
   = x_1 + \frac{\lg(m_1 + 1)}{2} x_1 + \frac{\lg(m_2 + 1)}{2} (x - x_1).
   \]

   Let $g(m_1, x_1)$ be the right side of this inequality, where $0 \leq m_1 < m$ and $0 \leq x_1 < x/2$, then
   \[
   \text{sum}(L(v_i, v_j)) \leq \max\{g(m_1, 0), g(m_1, x/2)\}.
   \]

   because $g$ is a linear function of $x_1$.

   \[
   g(m_1, 0) = \frac{x}{2} \lg(m - m_1) \leq \frac{x}{2} \lg m < \frac{x}{2} \lg(m + 1)
   \]
   (9)

   \[
   g(m_2, x/2) = \frac{x}{4} (2 \lg(m_1 + 1) + \lg(m - m_1))
   \]
   \[
   = \frac{x}{2} \lg2 \sqrt{(m_1 + 1)(m - m_1)} \leq \frac{x}{2} \lg(m + 1)
   \]
   (11)

   Thus, $\text{sum}(L(v_i, v_j))$ is not larger than $x \lg(m + 1)/2$. 

   We now consider the general case of DST-$N$ where we allow both the addition and
   removal of nodes. In this case the situation is even worse. In the next theorem we
   show that any algorithm for DST-$N$ has worst case performance that is unbounded
   as a function of the number of terminal nodes in the solution tree.

   **Theorem 3.3** Let $A$ be any algorithm for DST-$N$, and let $\{T_i, 0 \leq i \leq K\}$ be a
   sequence of trees generated by $A$ for an instance $(G, \text{cost}, R)$ of DST-$N$. Given any
   $M, \ell \in \mathbb{Z}^+$ there exists an instance $(G, \text{cost}, R)$ of DST-$N$, and an integer $j \in \mathbb{Z}^+$ such that
   \[
   \frac{\text{cost}(T_i)}{\text{cost}(T_{opt})} \geq M
   \]
   (12)
   for $j < i \leq j + \ell$ independent of the number of terminal nodes at step $i$. 

   8
Proof: Let graph $G$ contain cycle $C_{M+2}$ with $M + 2$ additional nodes, let each edge in $C_{M+2}$ have cost 1, and let every node in the cycle be connected to a distinct node not in the cycle by an edge of cost $\varepsilon$. Let the set $R$ consist of an initial sequence of $M + 2$ add requests, one for each node in $C_{M+2}$. Let the next $M$ steps remove each node of degree 2 in $T_{M+1}$ to create $T_j$, where $j = 2M + 1$. Thus, the cost of $T_j$ is $M + 1$ while an optimal solution has a cost of 1. For the remaining steps we alternately add and remove one of the nodes connected to a leaf node of $T_j$. We assume that the value of $\varepsilon$ is sufficiently small so that (12) holds.

If we relax our definition of DST-N so that the solution at each step need not be a tree, Theorem 3.3 no longer holds. However, $1 + \frac{1}{2} \lfloor \lg(n_i - 1) \rfloor$ is still a lower worst case bound on the error ratio even with this modification.

4. Rearrangeable Case

For a given graph $G = (V, E)$ and a cost function cost, we can define a complete graph $G' = (V', E')$ with cost' where $V = V'$, $E = \{ (u, v) | u, v \in V \}$ and cost'(u, v) = dist(u, v; G). The optimum Steiner tree for $G'$ is the same as that for $G$, and vice versa. Further cost function cost' satisfies the triangle inequality, that is, cost'(u, v) ≤ cost'(u, w) + cost'(w, v) for any $u, v, w \in V$. In this section in order to simplify the explanations we assume the input network $G$ and cost is a complete network satisfying the triangle inequality. We call this graph a distance graph. The results obtained remain valid even without this assumption.

4.1. Edge-Bounded Tree

If we do not consider the number of rearrangements, problem DST-R is treated as ST for each instance $(G, cost, S_k)$, which is an NP-hard problem. As a starting point, we apply the Minimum Spanning Tree Approximation Algorithm(MSTA) for ST [7]. MSTA is one of the most well known heuristics for the Steiner Tree Problem, because, in spite of its simplicity, it has the best worst case behavior among all the known heuristics; the minimum spanning tree $T_{MST}$ produced by MSTA has a cost that is never more than twice optimal. $T_{MST}$ is the minimum spanning tree for the subgraph induced by $S_k$.

However if we apply MSTA to DST-R, the number of rearrangements would be very large. For example, in the case of the graph shown in Fig 4, the number of rearrangements for each Step $k$ is $k$, which implies we may have to change every edge.

While MSTA generates $T_{MST}$ as a solution, the algorithm proposed is based on a $\delta$ edge-bounded tree defined below.
Definition 1 Let \( u \) and \( v \) be nodes in \( T \). If \( u \) and \( v \) satisfy the following condition, they are called a \( \delta \) edge-bounded pair.

\[
\text{For all } e \in p(u,v; T), \quad \text{cost}(e) \leq \delta \cdot \text{cost}(u,v),
\]

(13)

where \( p(u,v; T) \) is the set of edges on the path between \( u \) and \( v \) in \( T \).

Further if every pair of nodes in \( T \) is \( \delta \) edge-bounded, \( T \) is called a \( \delta \) edge-bounded tree.

A tree generated by \( \text{MSTA} \) is a \( \delta \) edge-bounded tree with \( \delta = 1 \). Thus, the \( \delta \) edge-bounded tree can be viewed as a generalization of \( T_{\text{MST}} \).

To explain our algorithm, another definition needs to be introduced.

Definition 2 For a node set \( V \), if tree \( T_e = (V_e, E_e) \) satisfies the following conditions, \( T_e \) is called "an extension tree of \( V \".

- \( V_e \supseteq V \).
- For any node \( v \) in \( V_e - V \), the degree, in \( T_e \), of \( v \) is larger than 2.
Lemma 4.1 If \( T = (V, E) \) is a \( \delta \) edge-bounded tree then
\[
\text{cost}(T) \leq \delta \cdot \text{cost}(T_{MST}) \leq 2\delta \cdot \text{cost}(T_{OPT})
\]
(14)
where \( T_{MST} \) is the tree generated by MST\( A \) and \( T_{OPT} \) is the optimal Steiner tree for \( V \).

Proof: In (14) the right inequality is valid from [7]. Since \( T \) and \( T_{MST} \) are trees and their node sets are the same, the cardinality of \( E \) is equal to the cardinality of the edge set of \( T_{MST} \), which is denoted by \( E_{MST} \). Let us assume that there exists a one to one mapping function \( f \) from \( E_{MST} \) to \( E \) such that if \( f(e') = e \) then edge \( e \) is contained in \( p(u, v; T) \) where \( u \) and \( v \) are the two endpoints of \( e' \) in \( T_{MST} \). Then from the definition of a \( \delta \) edge-bounded tree, \( \text{cost}(f(e')) \leq \delta \cdot \text{cost}(e') \). Thus,
\[
w(T) = \sum_{e \in E} \text{cost}(e) = \sum_{e' \in E_{MST}} \text{cost}(f(e')) \leq \sum_{e' \in E_{MST}} \delta \cdot \text{cost}(e') = \delta \cdot w(T_{MST})
\]
(15)
We complete this proof by showing the existence of \( f \).

For an edge \( e' = (u, v) \in E_{MST} \), let \( \Gamma(e') \) be the set of edges on the path between \( u \) and \( v \) in \( T \), that is, \( p(u, v; T) \). From P.Hall's Theorem [9, p.45 Th.5.1.1], there exists a one-to-one mapping function \( f \) if and only if
\[
|\Gamma(S)| \geq |S|
\]
(16)
for any subset \( S \subseteq E_{MST} \).

where \( \Gamma(S) = \bigcup_{e' \in S} \Gamma(e') \).

Let \( S \) be an arbitrary subset of \( E_{MST} \). Consider the graphs \( G_0 = (V(S), S) \) and \( G_1 = (V(\Gamma(S)), \Gamma(S)) \) where \( V(S) \) is the set of nodes incident with an edge in \( S \). Let \( \alpha_0 \) and \( \alpha_1 \) be the number of connected components of \( G_0 \) and \( G_1 \), respectively. As \( G_0 \) and \( G_1 \) are forests, the number of edges, \( |S| \) and \( |\Gamma(S)| \), are related to the number of nodes and the number of connected components as follows:
\[
|S| = |V(S)| - \alpha_0 \quad \text{and} \quad |\Gamma(S)| = |V(\Gamma(S))| - \alpha_1
\]
It is clear that \( V(S) \subseteq V(\Gamma(S)) \) and \( \alpha_0 \geq \alpha_1 \) from the definitions of \( G_0 \), \( G_1 \) and \( \Gamma(e) \). Thus
\[
|S| = |V(S)| - \alpha_0 \leq |V(\Gamma(S))| - \alpha_0 \leq |V(\Gamma(S))| - \alpha_1 = |\Gamma(S)|
\]
Therefore, (16) is valid.

Lemma 4.2 If \( T_e = (V_e, E_e) \) is a \( \delta \) edge-bounded extension tree tree for \( V \),
\[
\text{cost}(T_e) \leq 2\delta \cdot \text{cost}(T_{MST}) \leq 4\delta \cdot \text{cost}(T_{OPT})
\]
(17)
where \( T_{MST} \) is the tree generated by MST\( A \) and \( T_{OPT} \) is the optimal Steiner tree for \( V \).

Proof: Let \( T_{MST} \) be \( (V, E_{MST}) \). Let us assume that there exists a mapping function \( g \) from \( E_{MST} \) to power set \( 2^E \), such that
Condition 1  If \( e \in g((u, v)) \), \( e \) is contained in \( p(u, v; T_e) \).

Condition 2  For all \( e \in E_{MST} \), \(|g(e)| \leq 2\).

Condition 3  For all \( e \in E_e \), there is some \( e \in E \) such that \( e \in g(e) \).

If \( e \in g(e) \), \( \text{cost}(e) \leq \delta \cdot \text{cost}(e) \) from Condition 1 and the definition of a \( \delta \) edge-bounded tree. Let \( \text{cost}(g(e)) \) be \( \sum_{e \in g(e)} \text{cost}(e) \), then from Condition 2

\[
\text{cost}(g(e)) \leq 2\delta \cdot \text{cost}(e).
\]

Further \( E_e \subseteq \bigcup_{e \in E_{MST}} g(e) \) from Condition 3. Therefore

\[
w(T_e) = \sum_{e \in E_e} \text{cost}(e) \leq \sum_{e \in E_{MST}} \text{cost}(g(e)) \leq \sum_{e \in E_{MST}} 2\delta \cdot \text{cost}(e) = 2\delta \cdot w(T_{MST})
\]

In the remainder of the proof we show the existence of a mapping function \( g \) by induction on \(|V|\), the number of nodes in \( T_{MST} \).

1. \((|V| = 2)\)

\( T_{MST} \) has only one edge, which is denoted by \( e \). It is clear that \( T_e = T_{MST} \). Let \( g(e) = \{e\} \), then it satisfies Condition 1, 2 and 3.

2. \((|V| = n + 1)\)

By the inductive hypothesis for any tree \( T'_{MST} = (V', E'_{MST}) \) with \( n \) nodes and any extension tree \( T'_e \) for \( V' \), there exists a mapping function \( g' \) satisfying the conditions. Let \( T_{MST} = (V, E_{MST}) \) be a tree with \( n + 1 \) nodes and \( T_e = (V_e, E_e) \) be an extension tree for \( V \).

Since \( T_{MST} \) is a tree, there is a node with degree 1, which is denoted by \( u \). Let \( w \) be the node adjacent to \( v \). Consider

\[ T'_{MST} = (V', E'_{MST}) \quad \text{where} \quad V' = V - \{v\} \quad \text{and} \quad E'_{MST} = E - \{(v, w)\}. \]

\( T'_{MST} \) has \( n \) nodes. We will construct an extension tree for \( V' \), \( T'_e = (V'_e, E'_e) \), from \( T_e \) and define a mapping function \( g \) by modifying \( g' : E'_{MST} \rightarrow 2^{E_e} \). There are three cases to consider, depending on the degree of \( v \) in \( T_e \).

Case 1 (degree\((v; T_e) > 2)\)

Every node in \( V_e - V \) has degree larger than 2 in \( T_e \) and the degree of \( v \) is also larger than 2. Thus, every node in \( V_e - V' \) has degree larger than 2, which implies \( T_e \) is an extension graph of \( T'_{MST} \). From the inductive hypothesis, there exists a mapping \( g' : E'_{MST} \rightarrow 2^{E_e} \). Now define the function \( g : E_{MST} \rightarrow 2^{E_e} \) as follows:

\[
g(e) = \begin{cases} 
    g'(e) & \text{if } e \neq (v, w) \\
    \phi & \text{if } e = (v, w)
\end{cases}
\]

This function satisfies Condition 1, 2 and 3.

Case 2 (degree\((v; T_e) = 2)\)
Let \( x \) and \( y \) be the nodes adjacent to \( v \) in \( T_e \) and it can be assumed that the path from \( v \) to \( w \) goes through \( x \). (It is possible that \( x = w \).) Let

\[
T_e' = (V_e', E_e') \quad \text{where} \quad V_e' = V_e - \{v\} \quad \text{and} \quad E_e' = E_e - \{(x,v),(v,y)\} + \{(x,y)\}.
\]

Note that this modification does not change the degree of any node in \( V_e - \{v\} \). The degree of every node in \( V_e' - V' \) is not less than 3 because \( V_e' - V' \subset V_e - V \). Thus, \( T_e' \) is an extension graph of \( T_{MST} \), and there exists a function \( g' : E_{MST} \to 2^{E_e'} \). Now define the function \( g \) as follows:

\[
g(e) = \begin{cases} 
  g'(e) & \text{if } e \neq (v,w) \text{ and } (x,y) \not\in g'(e) \\
  \{(v,x)\} & \text{if } e = (v,w) \\
  g'(e) - \{(x,y)\} + \{(v,y)\} & \text{if } (x,y) \in g'(e)
\end{cases}
\]

If \( e = (v,w) \), Condition 1 is satisfied, since the path between \( v \) and \( w \) goes through \( x \). If path \( p(u,v;T_e) \) contains edge \( (x,y) \), \( p(u,v;T_e) \) contains \( (x,v) \) and \( (v,y) \). Thus, if \( (x,y) \in g'(e) \) Condition 1 holds for \( g(e) \). Clearly Condition 1 holds when \( g(e) = g'(e) \). It is also clear that \( g \) satisfies Conditions 2 and 3.

**Case 3 (degree(v;T_e) = 1)**

Let \( w \) be the nodes adjacent to \( v \) in \( T_e \). We divide this case into the following two subcases:

**Case 3.1 (w \in V' or degree(x;T_e) \geq 4 if x \not\in V')**. We define \( T_e' \) and \( g' \) as follows:

\[
T_e' = (V_e', E_e') \quad \text{where} \quad V_e' = V_e - \{v\} \quad \text{and} \quad E_e' = E_e - \{(v,x)\}.
\]

\[
g(e) = \begin{cases} 
  g'(e) & \text{if } e \neq (v,w) \\
  \{(v,x)\} & \text{if } e = (v,w)
\end{cases}
\]

**Case 3.2 (w \not\in V' and degree(x;T_e) = 3)** Let \( y, z \) be the two adjacent nodes to \( x \) except \( v \) and assume the path between \( v \) and \( w \) goes through \( y \). We define \( T_e' \) and \( g' \) as follows:

\[
T_e' = (V_e', E_e') \quad \text{where} \quad V_e' = V_e - \{v,x\} \quad \text{and} \quad E_e' = E_e - \{(v,x),(x,y),(x,z)\} + \{(y,z)\}.
\]

\[
g(e) = \begin{cases} 
  g'(e) & \text{if } e \neq (v,w) \text{ or } (y,z) \not\in g'(e) \\
  \{(v,x),(x,y)\} & \text{if } e = (v,w) \\
  g'(e) - \{(y,z)\} + \{(x,z)\} & \text{if } (x,y) \in g'(e)
\end{cases}
\]

We can verify that \( g \) satisfies Condition 1, 2 and 3 in a manner similar to that used for Case 2.
4.2. Algorithm

This section proposes an algorithm ($\text{EBA}(\delta)$) generating a sequence $\{T_k\}$ such that $T_k$ is a $\delta$ edge-bounded extension tree for $S_k$.

Figure 5 presents the details of $\text{EBA}(\delta)$. For an add request $(v_k, \text{add})$, $\text{EBA}(\delta)$ joins $v_k$ to $T_{k-1}$ by the shortest edge and investigates whether every pair $v_k$ and $w$ in $T_{k-1}$ is $\delta$ edge-bounded. If not, replace the maximum cost edge in $p(v_k, w; T_{k-1})$ by edge $(v_k, w)$. For a remove request $(v_k, \text{remove})$, if the degree of $v_k$ is larger than 2, $T_k$ is the same as $T_{k-1}$. Otherwise $v_k$ is deleted from $T_k$ and if $T_k - v_k$ is disconnected then the two components are joined by an edge $(u, v)$ such that the cost of the maximum edge in $p(x_0, x_1; T_{k-1} - v_k + (u, v))$ is minimized, where $x_0$ and $x_1$ are the nodes adjacent to $v_k$ in $T_{k-1}$. Further if the resulting graph has a non-terminal node with degree 2 or 1, repeat this step.

It is clear that the generated trees are extension trees for $S_k$. We begin with the following property to show they are $\delta$ edge-bounded trees.

Lemma 4.3 For any $\alpha (\geq 1)$ and $\delta (\geq 1)$, if a pair of nodes $u_0$ and $u_1$ is $\alpha$ edge-bounded in some intermediate tree $T$ generated by $\text{EBA}(\delta)$ then it is also $\alpha$ edge-bounded in any intermediate tree generated after $T$.

Proof: Let $T_a$ be an intermediate tree. The elementary modifications to $T_a$ in algorithm $\text{EBA}(\delta)$ are divided into the following four cases:

1. add a new node $v$ and join $v$ to $T_a$ by a minimum cost edge $e_a$ between them.
2. remove the maximum cost edge $e_d = (x_0, x_1)$ in $p(v_0, v_1; T_a)$ and join the two components by edge $e_a = (v_0, v_1)$, where cost($e_d$) > $\delta$cost($e_a$).
3. remove node $u$ with degree 1 and the edge incident with $u$.
4. remove node $u$ with degree 2 and the two incident edges, $(x_0, u)$ and $(x_1, u)$, and join the two components by the edge $e_a = (v_0, v_1)$ that minimizes $g(v_0, v_1)$, where function $g(v_0, v_1)$ is the cost of the maximum edge in $p(x_0, x_1; T - u + (u_0, u_1))$.

Let $T_b$ be the tree resulting from the application of one of the above operations to $T_a$. In order to prove the lemma, we show the following property holds for all four cases.

Property If $u_0$ and $u_1$ are a $\alpha$ edge-bounded pair in $T_a$ then they are a $\alpha$ edge-bounded pair in $T_b$.

Case 1: The path between $u_0$ and $u_1$ in $T_b$ is the same as the path in $T_a$ because there is no deletion for edges. Thus $u_0$ and $u_1$ are a $\alpha$ edge-bounded pair in $T_b$ by the assumption of the property.

Case 2: If $p(u_0, u_1; T_a)$ does not contain the deleted edge $e_d$, $p(u_0, u_1; T_a) = p(u_0, u_1; T_b)$ and the property is valid.
EBA(\(\delta\))
\[
\{\quad T_0 := (\{v_0\}, \emptyset), S_0 := \{v_0\} \text{ and } k = 1; \\
\quad \text{do } k \leq K \rightarrow \\
\quad \quad \text{if } r_k \text{ is an add request } \rightarrow \\
\quad \quad \quad T_k := \text{add}(v_k, T_{k-1}) \text{ and } S_k := S_{k-1} \cup \{v_k\}; \\
\quad \quad \quad \text{| } r_k \text{ is a remove request } \rightarrow \\
\quad \quad \quad \quad \text{S}_k := S_{k-1} - \{v_k\} \text{ and } T_k := \text{remove}(T_{k-1}, S_k); \\
\quad \quad \text{fi} \\
\quad \quad k := k + 1; \\
\quad \text{od}\}\]

add(\(v, T\))
\[
\{\quad \text{Let } W \text{ be a set of edges between } v \text{ and } T. \\
\quad \text{Select the minimum cost edge } (v, w_i) \text{ from } W; \\
\quad T := T + (v, w_i) \text{ and } W := W - (v, w_i); \\
\quad \text{do } W \neq \emptyset \rightarrow \\
\quad \quad \text{Select the minimum cost edge } (v, w_i) \text{ from } W \text{ and } W := W - (v, w_i) \\
\quad \quad \text{Find a maximum cost edge, } e, \text{ in } p(v, w_i; T); \\
\quad \quad \text{if } cost(e) > \delta \cdot cost(v, w_i) \rightarrow T := T - e + (v, w_i); \quad \text{fi} \\
\quad \text{od } \text{return}(T)\}\]

remove(\(T, S\))
\[
\{\quad W := V(T) - S \text{ where } V(T) \text{ is the node set of } T; \\
\quad \text{do } W \text{ contains a node of degree 2 or 1 } \rightarrow \\
\quad \quad \text{Let } w \in W \text{ be a node of degree 2 or 1; } \\
\quad \quad \text{if } degree(w) = 1 \rightarrow T := T - w \text{ and } W := W - \{w\}; \\
\quad \quad \text{| } degree(w) = 2 \rightarrow \\
\quad \quad \quad \text{Let } x_0 \text{ and } x_1 \text{ be the nodes adjacent to } w; \\
\quad \quad \quad \text{Let } C_0 \text{ and } C_1 \text{ be the connected components of } T - w; \\
\quad \quad \quad \text{Select two nodes } v_a \in C_0 \text{ and } v_b \in C_1 \text{ which minimizes } g(v_a, v_b), \text{ where } g(u, v) = \max\{cost(e) | e \in p(x_0, x_1; T - w + (u, v))\}; \\
\quad \quad \quad T := T - w + (v_a, v_b) \text{ and } W := W - \{w\}; \\
\quad \text{fi} \\
\quad \text{od } \text{return}(T)\}\]

Figure 5: Edge Bounded Algorithm (EBA(\(\delta\)))
If \( p(u_0, u_1; T_a) \) contains \( e_d \), \( p(u_0, u_1; T_b) \) is a subset of \( p(u_0, u_1; T_a) \cup p(v_0, v_1; T_a) \cup \{ e_a \} \) (see Fig 6).

Suffices to show that for every edge \( e \) in \( p(u_0, u_1; T_a) \cup p(v_0, v_1; T_a) \cup \{ e_a \} \),

\[
\text{cost}(e) \leq \alpha \cdot \text{cost}(u_0, u_1). \quad (18)
\]

If \( e \in p(u_0, u_1; T_a) \), (18) holds since the pair \( u_0 \) and \( u_1 \) is \( \alpha \) edge-bounded in \( T_a \). Note that this implies \( \text{cost}(e_d) \leq \alpha \cdot \text{cost}(u_0, u_1) \). If \( e \in p(v_0, v_1; T_a) \), (18) holds since \( e_d \) is the maximum cost edge in \( p(v_0, v_1; T_a) \). Finally, (18) holds for edge \( e_a \) since \( \delta \cdot \text{cost}(e_a) < \text{cost}(e_d) \).

Figure 6: Elementary Modification 2

**Case 3:** The property is valid because \( p(u_0, u_1; T_b) = p(u_0, u_1; T_a) \).

**Case 4:** Let the deleted two edges be \( (x_0, w) \) and \( (w, x_1) \). The two components of \( T - w \) are denoted by \( C_0 \) and \( C_1 \). If nodes \( u_0 \) and \( u_1 \) are contained in the same component, the property is valid because \( p(u_0, u_1; T_b) = p(u_0, u_1; T_a) \).

When \( u_0 \) and \( u_1 \) are contained in the different components, we can assume \( u_0, x_0 \) and \( v_0 \) are contained in \( C_0 \) and \( u_1, x_1 \) and \( v_1 \) are contained in \( C_1 \) without loss of generality (see Fig 7).

From the choice of \( (v_0, v_1) \), the following inequality holds:

\[
\max \{ \text{cost}(e) | e \in p(x_0, x_1; T_b) \} \leq \max \{ \text{cost}(e) | e \in p(x_0, u_0; C_0) \cup \{(u_0, u_1)\} \cup p(u_1, x_1; C_1) \} \quad (19)
\]

As \( u_0 \) and \( u_1 \) are a \( \alpha \) edge-bounded pair in \( T_a \), the right side of Inequality 19 is not larger than \( \alpha \cdot \text{cost}(u_0, u_1) \). Thus

\[
\text{cost}(e) \leq \alpha \cdot \text{cost}(u_0, u_1) \quad \text{for any } e \in p(x_0, x_1; T_b) \cup p(u_0, x_0; C_0) \cup p(x_1, u_1; C_1).
\]
On the other hand,
\[ p(u_0, u_1; T_b) \subseteq p(x_0, x_1; T_b) \cup p(u_0, x_0; C_0) \cup p(x_1, u_1; C_1). \]
Consequently, any edge in \( p(u_0, u_1; T_b) \) is not larger than \( \alpha \cdot \text{cost}(u_0, u_1) \).

This lemma is useful for estimating the number of rearrangements in addition to showing the following theorems. In the following theorems and lemmas, where not stated explicitly, we assume that an instance of DST-R is given with \( R = \{r_0, r_1, \ldots, r_K\} \).

**Theorem 4.1** Any tree \( T_k \), generated by \( \text{EBA}(\delta) \), is a \( \delta \) edge-bounded extension tree for \( S_k \). This implies the following inequalities:
\[
\text{cost}(T_k) \leq 2\delta \cdot \text{cost}(\text{MST}) \leq 4\delta \cdot \text{cost}(\text{OPT}),
\]
for \( 0 \leq k \leq K \).

**Proof:** It is clear that \( T_k \) is an extension tree for \( S_k \). It will be shown that \( T_k \) is \( \delta \) edge-bounded by induction on \( k \). \( T_1 \) is clearly a \( \delta \) edge-bounded tree. We assume \( T_{k-1} \) is \( \delta \) edge-bounded. From Lemma 4.3, every pair of nodes in \( T_{k-1} \) is \( \delta \) edge-bounded in \( T_k \). The remaining pair to be considered is \( v_k \) and \( w \in T_{k-1} \) when \( r_k = (v_k, \text{add}) \). \( \text{EBA}(\delta) \) examines each pair and if it not \( \delta \) edge-bounded then modifies the tree so that it becomes 1 edge-bounded. This modification maintains the \( \delta \) edge-bounded property for other pairs as Lemma 4.3 shows. Thus \( T_k \) is a \( \delta \) edge-bounded tree and the theorem follows from Lemma 4.2.\[\square\]
Theorem 4.2 If every request $r_k$ is an add operation, every generated tree $T_k$ satisfies
\[ \text{cost}(T_k) \leq \delta \cdot \text{cost}(T_{MST}) \leq 2\delta \cdot \text{cost}(T_{OPT}). \] (21)

Proof: If every request is an add request, the set of nodes of $T_k$ is the same as $S_k$. $T_k$ is $\delta$ edge-bounded. Thus the theorem is valid from Lemma 4.1. \hfill \Box

4.3. Total Number of Changes

In this section we estimate the total number of rearrangements for $\text{EBA}(\delta)$ in case of $\delta \geq 2$. Note that if an edge $(u, v)$ is contained in $T_i$, then the pair $u$ and $v$ is a 1 edge-bounded pair in any $T_j$ ($j \geq i$) from Lemma 4.3. We begin by finding other 1 edge-bounded pairs.

If $r_i$ is an add request, let $L_i$ be the set of endpoints for the edges added at Step $i$. The cardinality of $L_i$ is one more than the number of edges added since $v_i$ is one of the endpoints for each edge. Therefore the number of rearrangements at Step $i$, $\alpha_i$, is $|L_i| - 2$ because the number of edges deleted is one less than the number of the edges added. If $r_i$ is a remove request, let $L_i = \emptyset$.

Lemma 4.4 Every pair of nodes in $L_i$ is 1 edge-bounded in $T_j$ ($i \leq j \leq K$).

Proof: Let $L_k = \{v_k, w_1, w_2, \ldots, w_\alpha\}$. Since $T_k$ contains edge $(v_k, w_j)$ for any $w_j \in L_k$, a pair of $v_k$ and $w_j$ is 1 edge-bounded which implies it is 1 edge-bounded in any $T_j$ ($j \geq k$) from Lemma 4.3. We consider a pair of $w_i$ and $w_j$. Without loss of generality we can assume that
\[ \text{cost}(w_i, v_k) \leq \text{cost}(w_j, v_k). \] (22)

Consider the substep at which $(w_j, v_k)$ is added to the tree and let $T_a$ and $T_b$ be the intermediate trees just before and after this substep. Thus
\[ T_b = T_a - (x_0, x_1) + (w_j, v_k) \]
where $(x_0, x_1)$ is the edge deleted at this substep. Note that $T_a$ and $T_b$ contain an edge $(w_i, v_k)$.

Since a pair of $w_i$ and $w_j$ is $\delta$ edge-bounded in $T_a$ and $(x_0, x_1)$ is an edge in $p(w_i, w_j; T_a)$, $\text{cost}(x_0, x_1) \leq \delta \text{cost}(w_i, w_j)$. Since $(x_0, x_1)$ is replaced with $(w_j, v_k)$, $\text{cost}(x_0, x_1) > \delta \text{cost}(w_j, v_k)$. Thus $\text{cost}(w_j, v_k) < \text{cost}(w_i, w_j)$.

From inequality 22, $\text{cost}(w_i, v_k) \leq \text{cost}(w_j, v_k) < \text{cost}(w_i, w_j)$. Therefore, the pair of nodes $w_i$ and $w_j$ is 1 edge-bounded in $T_a$ and in any $T_j$ ($j \geq k$). \hfill \Box

Lemma 4.5 If $\delta \geq 2$,
\[ |L_i \cap L_j| \leq 1 \quad \text{for all } i \text{ and } j, \quad 0 < i < j \leq K. \] (23)
Proof: We assume $|L_i \cap L_j| \geq 2$ and derive a contradiction. Let $w_1$ and $w_2$ be nodes in $L_i \cap L_j$ and assume $j > i$. Note that edges $(v_j, w_1)$ and $(v_j, w_2)$ are added at Step j, where $r_j = (v_j, add)$. Without loss of generality we can assume

$$\text{cost}(v_j, w_1) \leq \text{cost}(v_j, w_2) \tag{24}$$

From the triangle inequality,

$$\text{cost}(w_1, w_2) \leq \text{cost}(w_1, v_j) + \text{cost}(v_j, w_2) \leq 2\text{cost}(v_j, w_2).$$

Consider the substep at which $(v_j, w_2)$ is added to an intermediate tree and let $T_a$ be the intermediate tree just before this substep. It is clear that

$$p(v_j, w_2; T_a) = \{(v_j, w_1)\} \cup p(w_1, w_2; T_a) \tag{25}$$

Since $w_1, w_2 \in L_i$, a pair $w_1$ and $w_2$ is 1 edge-bounded by Lemma 4.4. Thus

$$\text{cost}(e) \leq \text{cost}(w_1, w_2) \leq 2\text{cost}(v_j, w_2) \text{ for any } e \in p(w_1, w_2; T_a) \tag{26}$$

From (24), (25) and (26), pair $v_j$ and $w_2$ is 2 edge-bounded, which contradicts that $(v_j, w_2)$ is added at this substep. \hfill \blacksquare

Let $K_a$ be the number of add requests in $R$, and $K_r$ be the number of remove requests. We can derive the following theorem by using Lemma 4.4.

**Theorem 4.3** For any instance $(G, \text{cost}, R)$, if $\delta = 2$ then the total number of rearrangements in $\text{EBA}(\delta)$ is bounded by the following inequality:

$$\sum_{i=0}^{K} \alpha_i \leq \frac{1}{2}K_a(\sqrt{4K_a - 3} - 1) + K_r. \tag{27}$$

Proof: Let $R_a$ and $R_r$ be the sets of add and remove requests in $R$, respectively. Consider Step $i$ for which the request, $r_i$, is a remove request. The number of rearrangements $\alpha_i$ is not larger than the number of deleted nodes at this step. A deleted node is not contained in $S_k$, which implies there is a request to remove it from a connection. Thus the total number of rearrangements for $R_r$, $\sum_{r_i \in R_r} \alpha_i$, is not larger than the number of remove requests, $K_r(=|R_r|)$.

Consider the bipartite graph $G_b = (V_1, V_2; E)$ where $V_1 = R_a$, $V_2 = \cup_{r_i \in R_a} L_i$ and $E = \{(r_i, v_2)|v_2 \in L_i, v_2 \in V_2, r_i \in V_1\}$. If there are multiple requests to add the same node, we consider them as different nodes. It is clear that $|V_1| = |V_2| = K_a$. From Lemma 4.5, this graph does not contain a complete bipartite graph $K_{2,2}$ (that is, a cycle with 4 edges) as a subgraph. The following inequality is shown in [8, p.74 Th.10]:

$$z(n, n, 2, 2) \leq \frac{1}{2}n(1 + \sqrt{4n - 3})$$

19
where \( z(n, n, 2, 2) \) is the maximum number of edges in a bipartite graph whose two node sets both have \( n \) nodes and that does not include \( K_{2,2} \). Since the number of edges in \( G_i \) is equal to \( \sum L_i \) and \( \alpha_i \) is equal to \( |L_i| - 2 \),

\[
\sum_{r_i \in R_a} \alpha_i \leq \frac{1}{2} K_a(\sqrt{4K_a - 3} - 1).
\] (28)

We do not know if the \( K \sqrt{K} \) growth permitted by Theorem 4.3 can actually be achieved. We conjecture that the total number of rearrangements is not larger than \( K'(= K_a + K_b) \).

One of the interesting questions which still remains regarding DST is whether there exist algorithms for which both the worst case error ratio and the maximum number of rearrangements at each step are bounded by a pair of constants.

Other areas of interest include issues such as average case performance of algorithms for DST, distributed implementation of algorithms for DST, and the application of these algorithms to multipoint communication networks.

References


