Journal of Scientific Research & Reports
2(2): 598-611, 2013; Article no. JSRR.2013.010

SCIENCEDOMAIN *international*
www.sciencedomain.org

An Efficient Algorithm for Computation of a Minimum Average Distance Tree on Trapezoid Graphs

Sukumar Mondal[1*]

[1]*Department of Mathematics, Raja N. L. Khan Women's College, Gope Palace, Paschim Medinipur, 721 102, West Bengal, India.*

Author's contribution

The only author performed the whole research work. Author AOA wrote the first draft of the paper. Author AOA read and approved the final manuscript.

Research Article

Received 29th April 2013
Accepted 15th August 2013
Published 24th August 2013

ABSTRACT

The average distance $\mu(G)$ of a finite graph G = (V, E) is the average of the distances over all unordered pairs of vertices which can be used as a tool in analytic networks where the performance time is proportional to the distance between any two nodes. A minimum average distance spanning tree of G is a spanning tree of G with minimum average distance. Such a tree is sometimes referred to as a minimum routing cost spanning tree and these are of interest in the design of communication networks. In this paper, I present an efficient algorithm to compute a minimum average distance spanning tree on trapezoid graphs in $O(n^2)$ time, where n is the number of vertices of the graph.

Keywords: MAD tree; spanning tree; BFS tree; algorithms; complexity; trapezoid graphs.

AMS Mathematics Subject Classifications (2010): 05C85

**Corresponding author: E-mail: sm5971@rediffmail.com, smnlkhan@gmail.com;*

1. INTRODUCTION

A *trapezoid graph* can be represented in terms of *trapezoid diagram*. A trapezoid diagram consists of two horizontal parallel lines, named as top line and bottom line. Each line contains n intervals. Left end point and right end point of an interval *i* are a_i and $b_i(\geq a_i)$ on the top line and c_i and $d_i(\geq c_i)$ on the bottom line. A *trapezoid i* is defined by four corner points $[a_i, b_i, c_i, d_i]$ in the trapezoid diagram. Let *T* = {1, 2, . . .,*n*}, be the set of *n* trapezoids. Let *G* = (*V*, *E*) be an undirected graph with *n* vertices and *m* edges and let V = {1, 2, . . .,*n*}. *G* is said to be a *trapezoid graph* if it can be represented by a trapezoid diagram such that each trapezoid corresponds to a vertex in *V* and (*i, j*) ∈*E* if and only if the trapezoids *i* and *j* intersect in the trapezoid diagram [1]. Two trapezoids *i* and *j*(>*i*) intersect if and only if either $(a_j - b_i) < 0$ or $(c_j - d_i) < 0$ or both. We assume that the graph *G* = (*V*, *E*) is connected. Without any loss of generality we assume the following:

(a) a trapezoid contains four different corner points and that no two trapezoids share a common end point,
(b) trapezoids in the trapezoid diagram and vertices in the trapezoid graph are one and same thing,

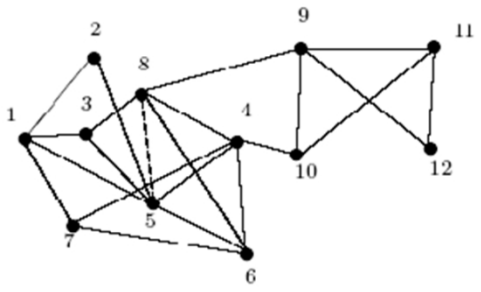

Fig. 1. A trapezoid graph *G*

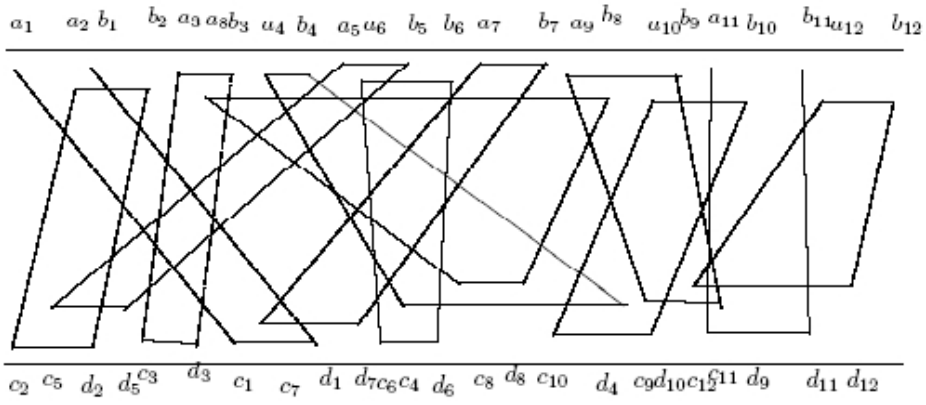

Fig. 2. A trapezoid diagram *T* of the graph *G* of Fig.1

(c) the trapezoids in the trapezoid diagram *T* are indexed by increasing right end points on the top line i.e., if $b_1 < b_2 < \ldots < b_n$ then the trapezoids are indexed by 1, 2, 3, . . ., *n* respectively.

Fig. 1 represents a trapezoid graph and it's trapezoid representation is shown in Fig. 2. The class of trapezoid graphs includes two well known classes of intersection graphs: the *permutation graphs* and the *interval graphs* [2]. The permutation graphs are obtained in the case where $a_i = b_i$ and $c_i = d_i$ for all *i* and the interval graphs are obtained in the case where $a_i = c_i$ and $b_i = d_i$ for all *i*. Trapezoid graphs can be recognized in $O(n^2)$ time [3]. The trapezoid graphs were first studied in [1,4,5]. These graphs are superclass of interval graphs, permutation graphs and subclass of cocomparability graphs [6].

Breadth- first-search (BFS) is a strategy for searching in a graph when search is limited to essential two operations: (a) visit and inspect a node of a graph; (b) gain access to visit the nodes that neighbour the currently visited node. The BFS begins at a root node and inspect all the neighbouring nodes. Then for each of those neighbour nodes in turn, it inspects their neighbour nodes which were unvisited, and so on.

BFS is a uniformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. In other words, it exhaustively searches the entire graph or sequence without considering the goal until it finds it.

Let *G* be a connected undirected graph, let *v* be a vertex of *G* and let *T* be its spanning tree obtained by the BFS of *G* with the initial vertex *v*. An appropriate rooted tree $T(v) = (V, E') \subseteq G$ let us call a *Breadth-First-Search Tree* (BFS tree, shortly) with the root *v*, the edges of *G* that do not appear in BFS tree let us call non-tree edges.

In an un-weighted tree $T = (V, E')$, where $|E'| = |V| - 1$, the *eccentricity e(v)* of the vertex *v* is defined as the distance from v to the vertex farthest from $v \in T$, i.e.,

$$e(v) = \max\{d(v, v_i), \text{ for all } v_i \in T\},$$

where $d(v, v_i)$ is the number of the edges on the shortest path between *v* and v_i.

In a weighted tree $T = (V, E')$, where $|E'| = |V| - 1$, the *eccentricity e(v)* of the vertex *v* is defined as sum of the weights of the edges from v to the vertex farthest from $v \in T$, i.e.,

$$e(v) = \max\{d(v, v_i), \text{ for all } v_i \in T\},$$

where $d(v, v_i)$ is the sum of the weights of the edges on the shortest path between v and v_i.

A vertex with minimum eccentricity in the tree *T* is called a *center* of that tree *T*, i.e., if $e(s) = \min\{e(v), \text{ for all } v \in V\}$, then s is the *1-center*. It is clear that every tree has either one or two centers.

The eccentricity of a center in a tree is defined as the *radius* of the tree and is denoted by $\rho(T)$, i.e.,

$$\rho(T) = \{\min_{v \in T} e(v)\}.$$

The *diameter* of a tree *T* is defined as the length of the longest path in *T,* i.e., the maximum eccentricity is the diameter. A spanning tree with minimum diameter is defined as *minimum diameter spanning tree*.

The *average distance* $\mu(G)$ of a finite permutation graph $G = (V, E)$ is the average over all unordered pairs of vertices of the distances,

$$\mu(G) = \frac{2}{n(n-1)} \sum_{\{u,v\} \subset V(G)} d_G(u, v),$$

where $d_G(u, v)$ denotes the distance between the vertices u and v, i.e., the length of a shortest path joining the vertices u and v. The average distance can be used as a tool in analytic networks where the performance time is proportional to the distance between any two nodes. It is a measure of the time needed in the average case, as opposed to the diameter, which indicates the maximum performance time.

The *minimum average distance spanning tree* (MAD tree, in short) of a trapezoid graph *G* is a spanning tree of *G* with minimum average distance.

2. SURVEY OF THE RELATED WORK

In general, the problem of finding a MAD tree is NP-hard [7]. A polynomial time approximation scheme is due to [8]. Hence, it is natural to ask for which restricted graph classes a MAD tree can be found in polynomial time. In [9], an algorithm is exhibited that computes a MAD tree of a given distance-hereditary graph in linear time. In [10], it is shown that a MAD tree of a given outerplanar graph can be found in polynomial time. Recently, Dahlhaus et al. [11], have designed a linear time algorithm to compute a MAD tree of an interval graph which runs in $O(n)$ time when the left and right boundaries of the intervals are ordered. In [12], Barefoot et al., have shown that if *T* is a MAD tree of a given connected graph *G*, then there exists a vertex *c* in *T* such that every path in *T* starting at *c* is induced in *G*. It remains an open problem to decide whether there exists a polynomial time algorithm to find a MAD tree of a vertex weighted interval graph. Olario et al. [13], have designed optimal parallel algorithms for problems modeled on a family of intervals on interval graphs to compute a MAD tree. Also Mondal et al. [14], have designed an optimal algorithms for computing the average distance on trapezoid graphs. Recently, Jana et. al. [15] have designed an efficient algorithm to compute a MAD tree on permutation graphs in $O(n^2)$ time.

2.1 Applications of the Problem

MAD trees, also referred to as minimum routing cost spanning trees, are of interest in the design of communication networks [6]. One is interested in designing a tree subnetwork of a given network, such that on average, one can reach every node from every other node as fast as possible.

2.2 Our Result

In this paper, I have designed an $O(n^2)$ time efficient algorithm to construct a MAD tree for a given trapezoid graph G with n vertices.

2.3 Organization of the Paper

In the next section, i.e., Section 3, I shall discuss about the construction of the tree. In Subsection 3.1, I discuss BFS tree on trapezoid graph. In Subsection 3.2, I discuss about the minimum diameter spanning tree. In Subsection 3.3, I develop modified spanning tree of the tree *T*. In Section 4, I present an algorithm to get MAD tree of the trapezoid graph. The time complexity is also calculated in this section. Finally Section 5 presents the conclusion part of the work.

3. CONSTRUCTION OF THE TREE

3.1 Construction of BFS Tree on Trapezoid Graph

It is well known that BFS is an important graph traversal technique and also BFS constructs a BFS tree. In BFS, started with vertex *v*, we first scan all edges incident on *v* and then move to an adjacent vertex *w*. At *w* we then scan all edges incident to *w* and move to a vertex which is adjacent of *w*. This process is continued till all the edges in the graph are scanned [16].

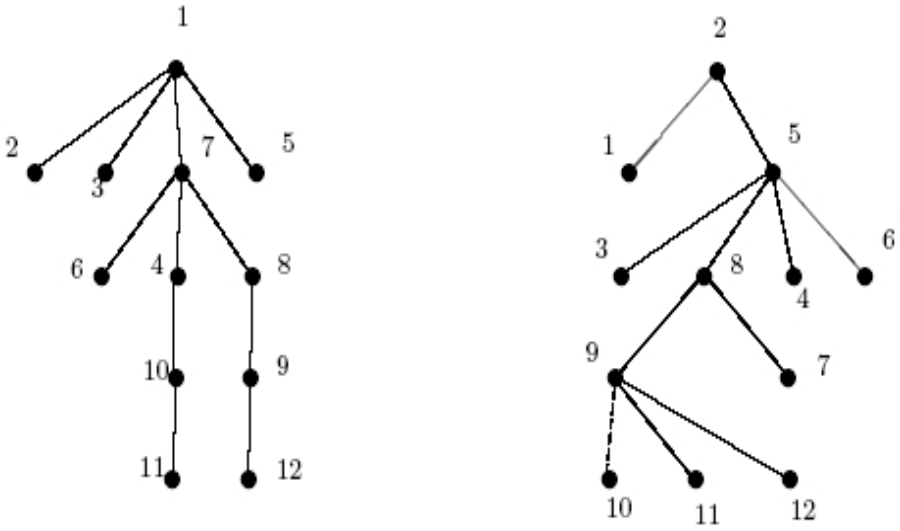

Fig. 3. A BSF tree *T*(1)* and *T*(2)* of the graph *G* of Fig. 2

BFS tree can be constructed on general graphs in $O(n + m)$ time, where *n* and *m* represent respectively the number of vertices and number of edges [17]. To construct this BFS tree on a trapezoid graph we consider the matching diagram. We first placed the trapezoid *x* on the zero level and all vertices adjacent to *x* on the first level. Next we traverse the matching diagram step by step from left to right. In first step we scan the untraversed trapezoids which are situated at the right side of trapezoid *x* and select maximum trapezoid which will be root for next step. In each iteration, we scan numbers within an interval on the top line as well as

the bottom line of the channel and scan trapezoid number within such interval on the bottom channel alternately in an order. Recently Mondal et al. [18], have designed an algorithm to construct a BFS tree $T^*(i)$ with root as i on trapezoid graph in *O*(*n*) time where n is the number of vertices. Fig. 3 is the BFS tree constructed by the Algorithm TBFS [18].

3.2 Computation of Minimum Diameter Spanning Tree

Let $T^*(1), T^*(1')$, be two BFS trees designed by Algorithm TBFS [18] of a trapezoid graph *G* where 1 is the vertex corresponding to the trapezoid of top right corner b_1 and $1'$ is the vertex corresponding to the trapezoid whose first left corner point is $c_{1'}$ on the bottom line of the diagram.

Case I: Let *T* be the minimum heighted tree between $T^*(1)$ and $T^*(1')$. Let P be the main path (longest path) of *T* of the trapezoid graph *G* and it is denoted by $u_0^* \to u_1^* \to u_2^* \to \cdots \to u_k^*$, where $k \le (n-1)$ and u_0^* is either the vertex 1 or the vertex corresponding to the trapezoid whose leftmost lower corner is c_i of G. The vertices of the path P, i.e., u_i^*, $i = 0, 1, 2, 3, \ldots, k$ defined as *internal nodes*.

Now the following termsare important:

The *open neighbourhood* of the vertex u_i^* in the path P of G, denoted by $N(u_i^*)$ and defined as $N(u_i^*) = \{u : (u, u_i^*) \in E\}$ and the *closed neighbourhood* $N[u_i^*] = \{u_i^*\} \cup N(u_i^*)$, where E is the edge set of the given trapezoid graph.

Next consider the*level* of the vertex u as the distance of u from the root i of the BFS tree $T(i)$ and denote it by *level(u)*, $u \in V$ and take the level of the root i as 0. The level of each vertex on BFS tree $T(i)$, $i \in V$ can be assigned by the BFS algorithm of Chen and Das [19]. The level of each vertex of the tree T can be computed in $O(n)$ time.

Case II: If the heights of the two trees are equal, then choose such BFS tree whose all the adjacent (open neighbourhood) to the root are adjacent to the first internal node. Otherwise select any one BFS tree as *T*. That is we select such tree whose $N(1) = N(u_1^*)$, where 1 and u_1^* are root and first internal node of the tree, otherwise select any one.

As per construction of BFS tree by the algorithm TBFS [18], we have the important result in BFStree *T*.

In trapezoid diagram, two intersecting trapezoids of *V*-type or inverted *V*-type defined as the *scissors type segments* of trapezoid graphs; otherwise *box type* segments.

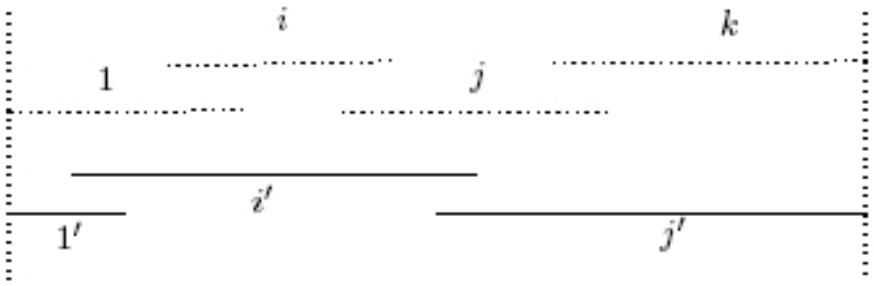

Fig. 4. Region covered by the projection of minimum number of trapezoids

Lemma 1 *Minimum number of scissors type or box type intersecting trapezoids with maximum spread of the trapezoid diagram cover the whole region.*

Proof. Let $\{1, 2, \ldots, n\}$ be the set of numbers represents the vertices number corresponding to the trapezoids $[a_i, b_i, c_i, d_i]$, i = 1, 2, . . .,*n*, respectively. To cover 1 to *n*, i.e., whole region, there are two types of scissors or boxes whose projection marked as dotted lines and continued lines, shown in the Fig. 4.

Firstly, if one select first dotted trapezoid 1 on top line. Then we consider such trapezoid *i*, which is maximum spread with respect to maximum b_i's on top line and maximum d_i's on bottom line among all adjacent to the corresponding first trapezoid 1 and we marked all the line segment adjacent to 1. Next, we consider such trapezoid *j* which is maximum spread among all unmarked trapezoid adjacent to *i,* continuing this process until all the trapezoids are covered, i.e., marked. Let this path be of the type $1 \to i \to j \to k \to \ldots$

Similar manner to be followed from the bottom line. Then we obtain the another path of the form $1' \to i' \to j' \to k' \to \ldots$. In this way whole region is covered by the both paths.

Next consider the minimum path between them which will be the shortest path to cover all the trapezoids. Hence, the scissors type or box type trapezoid segments with maximum spread in the trapezoid diagram corresponding to the trapezoid graph covered the whole region. □

Lemma 2 [20] *Two intersecting trapezoids of the trapezoid graph of the matching diagram are assigned on the same level or adjacent levels.*

Lemma 3. *If* $u, v \in V$ *and* $|level(u) - level(v)| > 1$ *in* T*, then there is no edge between the vertices* u *and* v *in* G*.*

Proof. If possible let $|level(u) - level(v)| > 1$ but $(u, v) \in E$, i.e., u and v are directly connected. Since u and v are directly connected so by Lemma 2 either $level(u) = level(v)$ or $|level(u) - level(v)| = 1$ which is contradictory to the assumption that $|level(u) - level(v)| > 1$. Hence $(u, v) \notin E$, i.e., u and v are not directly connected in G. □

The time complexity of the algorithm TBFS is stated below.

Theorem 1 [18] The BFS tree rooted at any vertex $x \in V$ can be computed in O(n) time for a trapezoid graph containing n vertices.

Obviously, this BFS tree is a spanning tree.

Now, I shall prove that this spanning tree is also a minimum diameter spanning tree.

Lemma 4 *The spanning tree T is a minimum diameter spanning tree.*

Proof. According to the construction the BFS tree, the main path of the tree *T* is the longest path which is the diameter of *T*. The main path contains minimum number of scissors type trapezoids (by Lemma 1). But this diameter is minimum, because *T* is the minimum heighted tree between $T^*(1), T^*(1')$. Also, *T* is a spanning tree. Hence *T* is a minimum diameter spanning tree. □

In next subsection, to compute the MAD tree of the trapezoid graph by modifying the spanning tree*T*.

3.3 Modification of Spanning Tree *T*

It is observed that T is not necessarily a MAD tree. So, modification of T is necessary. One can modify by following way:

Firstly compute $N(u_i^*) \in G$ for every internal nodes (vertices on the longest path) $u_i^* \in T$. If there are any common adjacent vertices of two consecutive internal nodes of P, i.e., $N(u_i^*) \cap N(u_{i+1}^*) \neq \phi$ then by the following way one can shift them.

i) In G, if any common adjacent vertex w of u_i^* and u_{i+1}^* exist, i.e., $N(u_i^*) \cap N(u_{i+1}^*) \neq \phi$ then calculate the number of vertices to the upper side (if exist) and lower side of the internal node u_i^* in T with u_i^* as origin. Next calculate their difference, say, d_1.

ii) Again, find the number of vertices to the upper part and lower part (if exist) of the internal node u_{i+1}^* in T with u_{i+1}^* as origin (ignoring the vertex w). Next calculate their difference, say, d_2.

iii) If $d_1 - d_2 \leq 0$, then unaltered, i.e., w is finally adjacent of u_i^* in T and if $d_1 - d_2 > 0$, then the adjacent vertex w of the node u_i^* is shifted to the node u_{i+1}^*, i.e., w is finally adjacent of u_{i+1}^* in T.

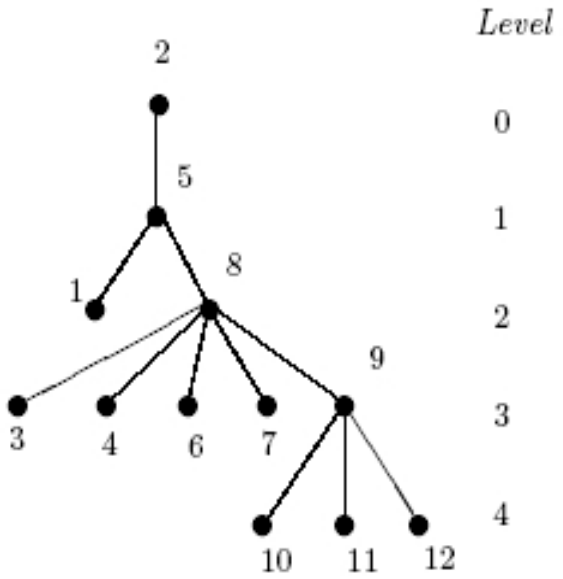

Fig. 5. Modified BSF tree T' of the BSF tree *T*(2)*

Similar idea is used for pair of any two consecutive internal nodes on the main path P. Under these conditions we develop the spanning tree T and it is denoted by T'. Fig. 5 is the modified BFS tree T' of the BFS tree T shown in Fig. 3.

Lemma 5. *If T' is a BFS rooted tree, then the distance $d_{T'}(u, v)$ between the vertices u and v in T' is given by*

$$d_{T'}(u, v) = \begin{cases} level(v), \text{ if } u \text{ is a root,} \\ level(v) - level(u), \text{ if } u = u^* \text{ (internal node),} \\ |level(v) - level(u)| + 1, \text{ if } u \text{ be any leaf.} \end{cases}$$

Proof. In the tree T', with u as root there exists a unique shortest path $u \to z_1 \to z_2 \cdots \to z_{p-1} \to v$ from u to any vertex $v \in G$, with u as parent of z_1 and z_i as parent of z_{i+1} and so on for each $i = 1, 2, \ldots, p-2$ and z_{p-1} as parent of v. Since each vertex of this path is directly connected with the next one, hence the length of this path is $p = level(v)$. Thus $d(u, v) \le p$.

Next we are to show that $d(u, v) \not< p$. If possible, let $d(u, v) = q < p$. Then there exist a path $u \to y_1 \to y_2 \cdots \to y_{q-1} \to v$ from u to any vertex $v \in G$. As each vertex of this path is directly connected with the next one using Lemma 2, $level(y_1)$ is either 0 or 1 since $level(u) = 0$. By same Lemma 2, $level(y_{k+1})$ is either $level(y_k)$ or $level(y_k) + 1$ or

$level(y_k)-1$. Thus $level(y_2)$ is 0 or 1 or 2, $level(y_3)$ is 0 or 1 or 2 or 3 and so on. Therefore $level(v)$ is 0 or 1 or 2 or...or q. This is a contradiction since $level(v)=p$ and $p>q$. Hence $d(u,\ v)\not\leq p$ which implies that $d(u,\ v)=p$, i.e., $d(u,\ v)=level(v)$.

If $u=u^*$, i.e., the internal node, then as per rule of construction of BFS, there is a shortest path $u\to z_1'\to z_2'\cdots\to v$. Here z_1' is at next level of u^*, z_2' is at the next level of z_1' and so on upto v. So $d(u^*, z_1')= 1 =(i+1) - i$, $d(u^*, z_2') = d(u^*, z_1') + d(z_1', z_2') = 1 + 1 = 2 = (i + 2) - i$. If $d(u^*, z_k') = k = (i + k) - i$, then

$$d(u^*,\ z_{k+1}')=d(u^*,\ z_k')+d(z_k',\ z_{k+1}')=k+1=(i+k+1)-i\ .$$

When u is any leaf, then there is a path from u to v via the parent of u. If $level(u)=i$, then $level(parent(u))=i-1$ and $parent(u)$ is an internal node (as per construction of BFS rooted as 1 or$1'$). Therefore

$$d(u,\ v)=d(u,\ parent(u))+d(parent(u),\ v)=1+level(v)-level(parent(u))\ .\square$$

4. ALGORITHM AND ITS COMPLEXITY

In this section, I present an efficient algorithm to construct MAD tree for a given trapezoid graph. Also, the correctness of the algorithm and its time complexity are presented here.

Algorithm TMAD-TREE

Input: A trapezoid graph G = (V; E) with its trapezoid representation $T_i = [a_i, b_i, c_i, d_i]$ $i = 1, 2, \ldots, n$.

Output: MAD tree T' and *average distance* $\mu(T')$.

Step 1: Compute the minimum diameter spanning tree T //Section 3:2//
and Calculate $level(u)=d(u^*,\ u),\ u^*=1\ or\ 1'$.

Step 2: Compute $N(u_i^*)\in G$ for all $u_i^*\in T$, and k = height of the tree T, i.e., highest level.

Step 3: //Modification of the tree T //
Compute common vertices, if any, of two consecutive internal nodes u_i^* and $u_{i+1}^*,\ i=0,\ 1,\ 2,\ \ldots,\ k-1$ of the main path P may be shifted as leaves to the node u_{i+1}^* under following conditions otherwise remains unaltered.

Step 3.1: For $i=0$ to $k-1$ do
If $N(u_i^*)\cap N(u_{i+1}^*)=\phi$ then go to Step 3.1,
If $N(u_i^*)\cap N(u_{i+1}^*)\neq\phi$ and let $w\in N(u_i^*)\cap N(u_{i+1}^*)$ then,

Step 3.1.1: Calculate the number of vertices to the upper part and lower part of

theinternal node u_i^* in T with u_i^* as origin.

Next calculate their difference, d_1.

Step 3.1.2: Calculate the number of vertices to the upper part and lower part of the internal node u_{i+1}^* with it as origin (ignoring the vertex w).

Next calculate their difference, d_2.

Step 3.1.3: If $d_1 - d_2 \leq 0$, then unaltered and if $d_1 - d_2 > 0$, then the adjacent vertex w of the internal node u_i^* is shifted to the internal node u_{i+1}^*.

Step 3.2: Set T' as modified spanning tree of T on trapezoid graph G.

Step 4: Calculate $d_{T'}(u,\ v) = \begin{cases} level(v), \text{ if } u \text{ is a root,} \\ level(v) - level(u), \text{ if } u = u^* \text{ (internal node),} \\ |\ level(v) - level(u)\ | + 1, \text{ if } u \text{ be any leaf.} \end{cases}$

and $\mu(T') = \dfrac{2}{n(n-1)} \sum_{\{u,v\} \subset V(T')} d_{T'}(u,\ v)$.

end **TMAD-TREE**

4. 1 Illustration of the Algorithm

In Fig. 3, T* (2) is the required spanning tree of the trapezoid graph *G*, u_i^* = 2 and u_{i+1}^* = 5 are two consecutive nodes. 1 is the common adjacent of the vertices u_i^* = 2 and u_{i+1}^* = 5, i.e., $w \in \{2\}$. Taking the vertex 2 as origin, the number of vertices to the upper side of 2 is 0 and the number of vertices to the lower side of 2 is 10. Hence their difference is d_1 = 10 - 0 = 10.

Again taking the vertex 5 as origin, the number of vertices to the upper side of 5 is 1 (ignoring the vertex 1) and the number of vertices to the lower side of 5 is 6 when the vertex 1 is adjacent to the vertex 5. Hence their difference is d_2 = 6 - 1 = 5. Therefore $d_1 > d_2$. So, the vertex 1 is shifted to the node 5. Next, we consider next two consecutive nodes 5 and 8. 3 and 4 are the common adjacent of the vertices u_{i+1}^* = 5 and u_{i+2}^* = 8, i.e., $w \in \{3,4\}$. Taking the vertex 5 as origin, the number of vertices to the upper side of 5 is 1 and the number of vertices to the lower side of 5 is 6. Hence their difference is d_1 = 6 - 1 = 5.

Again taking the vertex 8 as origin, the number of vertices to the upper side of 8 is 3 (ignoring the vertices 3, 4) and the number of vertices to the lower side of 8 is 4 when the vertices 3, 4 are adjacent to the vertex 8. Hence their difference is d_2 = 4 - 1 = 3. Therefore $d_1 > d_2$. So, the vertices 3 and 4 are shifted to the node 8.

Then we have the modified spanning tree T' (Fig. 5).

Next calculate the average distance $\mu_1(G)$ corresponding to the tree T'. Again

calculateaverage distance $\mu_2(G)$ corresponding to the tree T'. Here $\mu_1(G) = \frac{176}{66}$= 2.67 (approx.) and $\mu_2(G) = \frac{158}{66}$= 2.39 (approx.).

Clearly, $\mu_1(G) > \mu_2(G)$. Hence T' is the minimum average distance tree of the trapezoid graph G.

Theorem 2 *The tree formed by Algorithm TMAD-TREE is the MAD tree*.

Proof. Initially, I have constructed the BFS tree *T*. By nature and method of construction of BFS tree it gives minimum diameter (Lemma 4), i.e., there exists a shortest path by which every other vertices are directly connected with any internal nodes. So, this BFS tree gives the guarantee that there is no other shortest path to connect all other vertices directly with the internal nodes. Again, since the leaves, in level of difference between two or more in BFS, are not adjacent, so they may be connected via their parent vertex, i.e., internal nodes (Lemma 2 and Lemma 3). So to get minimum average distance one leaf may be shifted to immediate level, i.e., adjacent to just next internal node. Therefore sum of the distances among all pair vertices through the main path is minimum (Lemma 5). Again, by Step 3, check the distances among leaves of the tree. The shifting condition of the leaves, as leaves of other internal node (in next level), implies sum of the distances among the leaves is minimum. Hence, sum of the distances among the vertices, in tree *T*' is minimum.Therefore, the tree formed by the Algorithm TMAD-TREE is a MAD tree. □

Next I describe the time complexity of the algorithm.

Theorem 3 *The MAD tree of a trapezoid graphs G with n vertices, by Algorithm TMAD-TREE, can be computed in O(n^2) time*.

Proof. Step 1 takes *O*(*n*) time (Theorem 1). Step 2, i.e., computation of open neighbourhood of all internal nodes on the main path and add with corresponding nodes can be computed in *O*(n^2) time. Each Step 3.1.1 and Step 3.1.2 takes *O*(*n*) time. Also Step 3.1.3 runs in *O*(1) time. But Step 3.1 runs (k – 1) times, so total time complexity of Step 3.1 is *O*(n^2), where k is of *O*(*n*). Again Step 3.2, i.e., modification of the tree can be computed in *O*(n^2) time. The last step, i.e., Step 4, in worst case, can be computed in *O*(n^2) time. Hence, overall time complexity of our proposed algorithm is *O*(n^2) time with *n* vertices of the trapezoid graphs. □

5. CONCLUSION

In this paper, I have propose an elegant and efficient algorithm to compute a minimum average distance spanning tree on trapezoid graphs which is based on BFS technique for the minimum diameter of such graphs. The time complexity of this algorithm is of *O*(n^2) time, where *n* is the number of vertices of the trapezoid graph. To the best my knowledge, it is the minimum time algorithm for MAD tree on trapezoid graphs.

ACKNOWLEDGEMENT

I am grateful to four anonymous referees for their many helpful and valuable comments to improve this paper. The author is also grateful to the Teacher-in-Charge, Head of the Department and all other colleagues for their mental support regarding this research project

work. This work was supported in part by the University Grants Commission, Major Research Project grants F. 41-764/ 2012 (SR). It was conducted at Raja N. L. Khan Women's College, Department of Mathematics, Midnapore-721 102, West Bengal, India sponsored by DST (Govt. of India), CPE (U. G. C) and BSR.

COMPETING INTERESTS

Author has declared that no competing interests exist.

REFERENCES

1. Dagan I, Golumbic MC, Pinter RY. Trapezoid graphs and their coloring, Discrete Applied Mathematics. 1988;21:35-46.
2. Golumbic MC. Algorithmic graph theory and perfect graphs, Academic Press, New York; 2004.
3. Ma T, Spinrad JP. An $O(n^2)$ algorithm for 2-chain problem on certain classes of perfect graphs, In: Proc. 2^{nd} ACM-SIAM Symp. on Discrete Algorithms; 1991.
4. Corneil DG, Kamula PA. Extension of permutation and interval graphs, CongressusNumerantium. 1987;58:267-275.
5. Pnueli A, Lempel A,Even S. Transitive orientation of graphs and identification of trapezoid graphs, Canad. J. Math. 1971;23:160-175.
6. Liang YD. Domination in trapezoid graphs, Information Processing Letters. 1994;52:309-315.
7. Johnson DS, Lenstra JK,Rinnooy-Kan AHG. The complexity of the network design problem, Networks. 1978;8:279-285.
8. Bang Ye We, Lancia G, Bafna V, Chao KM, Ravi R,Chuang Yi Tang. A polynomial time approximation scheme for minimum routing cost spanning trees, SIAM Journal on Computing. 1999;29:761-778.
9. Dahlhaus E, Dankelmann P, Goddard W, Swart HC. MAD trees and distance-hereditary graphs, Discrete Applied Mathematics. 2003;131:151-167.
10. Dankelmann P, Slater P. Average distance in outerplanar graphs, Journal of Graph Theory. 2001;38(1);1-17.
11. Dahlhaus E, Dankelmann P, Ravi R. A linear time algorithm to compute a MAD tree of an interval graph, Information Processing Letters. 2004;89:255-259.
12. Barefoot CA, Entringer RC, Szekely LA. Extremal values of distances in trees, Discrete Applied Mathematics. 1997;80:37-56.
13. Olariu, S, Schwing J, Zhang J. Optimal parallel algorithms for problems modeled by a family of intervals, IEEE Transactions on parallel and Distributed Systems. 1992;3:364-374.
14. Mondal S, Pal M, Pal TK. An optimal algorithm to solve the all-pairs shortest paths problem on trapezoid graphs, Journal of Mathematics Modelling and Algorithms. 2003;2:57-65.
15. Jana B, Mondal S. Computation of a minimum average distance tree on permutation graphs, Annals of Pure and Applied Mathematics. 2012;2:74-85.
16. Deo N. Graph theory with applications to engineering and computer science, Prentice Hall of India Private Limited, New Delhi; 1990.
17. Tarjan RE. Depth first search and linear graph algorithm, SIAM J. Comput. 1972;2:146-160.
18. Mondal S, Pal M, Pal TK. An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs, Intern. J. Comput. Engg. Sci. 2002;3:103-116.

19. Chen CCY, Das SK. Breath- first traversal of trees and integer sorting in parallel, Information Processing Letters. 1992;41:39-49.
20. Barman S, Mondal S, Pal M. An efficient algorithm to find next-to-shortest path on trapezoid graphs, Advanced in Applied Mathematical Analysis. 2007;2:97-107.

Peer-review history:
The peer review history for this paper can be accessed here:
http://www.sciencedomain.org/review-history.php?iid=250&id=22&aid=1916