International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 3, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Computation of a Minimum Average Distance Tree on Circular-arc Graphs

Biswanath Jana
Department of Applied Mathematics with Oceanology and Computer Programming, Vidyasagar University, Midnapore - 721102, India.

Sukumar Mondal
Department of Mathematics, Raja N. L. Khan Women's College, Gope Palace, Paschim Medinipur - 721102, India.

Madhumangal Pal
Department of Applied Mathematics with Oceanology and Computer Programming, Vidyasagar University, Midnapore - 721102, India.

Abstract **– The average distance of a finite graph $G = (V, E)$ is the average of the distances over all unordered pairs of vertices. 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. In this paper, we present an efficient algorithm to compute a minimum average distance spanning tree on circular-arc graph in $O(n^2)$ time, where n is the number of vertices of the graph.**

Keywords **– Algorithms With Complexity, Circular-Arc Graphs, MAD Tree, Spanning Tree.**

I. INTRODUCTION

A graph $G = (V, E)$ is called an *intersection graph* for a finite family F of a non-empty set if there is a one-to-one correspondence between F and V such that two sets in F have non-empty intersection if and only if their corresponding vertices in V are adjacent to each other. We call F an intersection model of G. For an intersection model F, we use $G(F)$ to denote the intersection graph for G.

Intersection graphs have received much attention in the study of algorithmic graph theory and their applications [1]. Well-known special classes of intersection graphs include interval graphs, chordal graphs, circular-arc graphs, permutation graphs, circle graphs and so on. If F is a family of arcs on a circle, then G is called a circular-arc graph for F and F is called a circular-arc model of G. If F is a family of line segments on real line, then G is called an interval graph for F.

Let $S = \{C_1, C_2, ..., C_n\}$ be a family of n arcs on a circle C. Each endpoint of the arcs in assigned to a positive integer, called a coordinate. The endpoints of each arc are located on the circumference of C in the ascending order of the values of the coordinates in the clockwise direction. For convenience, each arc C_i, $i = 1, 2, ..., n$, is represented as (h_i,t_i), where h_i (the head) and t_i (the tail) denote, respectively that starting and ending points of the arc when it is traversed in counter clockwise manner, starting with an arbitrary chosen point on C which is not an endpoint of any arc in S. Without loss of generality, we assume the following:

(i) no single arc in S covers the entire circle C by itself,
(ii) no two arcs share a common endpoint,
(iii) $\bigcup_{i=1}^{n} C_i = C$ (otherwise, the problem becomes one on interval graph),
(iv) the arcs are sorted in increasing values of t_i's, i.e.,$t_i > t_j$ for $i > j$.

Also the family of arcs S is said to be canonical if h_i's and t_i's for $i = \{1, 2, ..., n\}$ are all distinct integers.

A path of a graph G is an alternating sequence of distinct vertices and edges, beginning and ending with vertices. The length of a path is the number of edges in the path. A path from vertex i to j is a shortest path from i to j with lower length. The shortest distance (i.e., the length of the shortest path) between the vertices i and j is denoted by $d_G(i, j)$.

Alternatively, a circular-arc graph can be defined as follows:

An undirected graph $G = (V, E)$ is a circular-arc graph if and only if

(i) its vertices circularly indexed as $v_1, v_2, ...,v_n$ and
(ii) $(v_i,v_j) \in E$, provided C_i and C_j intersect with each other, where v_i and v_j are the vertices in the graph G corresponding to the arcs C_i and C_j respectively.

It may be noted that the arc C_i and the vertex v_i or i are one and the same thing.

Circular-arc graphs have applications in genetic research [2], traffic control [3], etc. This class of graphs admits some interesting subclasses:

(1) Proper circular-arc graphs: a graph G is a proper circular-arc (PCA) graph if there is a circular-arc representation of G such that no arc is properly contained in any other.

(2) Unit circular-arc graphs: a graph G is a unit circular-arc (UCA) graph if there is a circular-arc representation of G such that all arcs are of the same length.

Clearly, it can be easily proved that UCA PCA. In [4], the author showed that this inclusion is strict. An example of a PCA graph which is not a UCA graph has also been given by Golumbic [1].

(3) Helly circular-arc graphs: A family of subsets S satisfies the Helly property when every subfamily of it consisting of pairwise intersecting subsets has a common element. So, a graph G is a Helly circular-arc (HCA) graph if there is a circular-arc representation of G such that the arcs satisfy the Helly property.

(4) Clique-Helly circular-arc graphs: a graph G is a clique-Helly circular-arc (CH-CA) graph if G is a circular-arc graph and a clique-Helly graph. A graph is clique-Helly when its cliques satisfy the Helly property.

Tucker [5], proposed an $O(n^3)$ time algorithm for recognizing a circular-arc graph. Deng et al. [6] presented an $O(n+m)$ time algorithm for presentation of circular-arc graph.

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 3, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

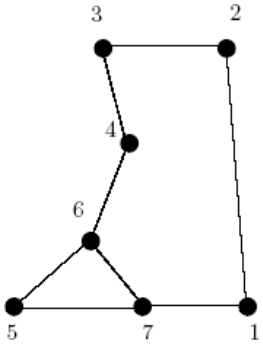


Fig.1. A circular-arc graph G.

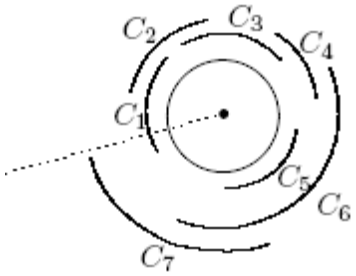


Fig.2. Circular-arc diagram of the circular-arc graph G of Figure 1.

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, for short) 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 $T = (V,E)$, the eccentricity $e(v)$ of the vertex v is defined as the distance from$v$ to a vertex farthest from v in$T$, i.e.,$e(v) = max\ \{d(v, v_i), v_i \in T\}$,

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

In a weighted tree $T = (V, E)$, the eccentricity $e(v)$ of the vertex v is denoted as sum of the weights of the edges from$v$ to a vertex farthest from $v \in T$, i.e.,$e(v) = max\ \{d(v, v_i), v_i \in T\}$,

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 $e(s) = \min\{e(v)$, for all $v \in V\}$, then s is the 1-center. It is fact that every tree has either one or two centers vertices which is classical result for unweighted trees. However, if weighted trees are considered, then this is only true if all edge-weights are strictly positive. Recently, Jana et al. [7] have designed an efficient algorithm to compute inverse 1-center location on the weighted trees which takes $O(n)$ time.

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 denoted 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 circular-arc graph $G = (V, E)$ is the average over all unordered pairs of vertices of the distances,

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

where$d_G(u, v)$denotes the distance betweenthe vertices u and v, i.e., the length of a shortest path joining the vertices u and v.

The minimum average distance spanning tree (MAD tree, for short) of a circular-arc graph G is a spanning tree of G with minimum average distance.

1.1 Survey of the Related Work

In general, the problem of finding a MAD tree is NP-hard [8]. A polynomial time approximation scheme is due to [9]. Hence it is natural to ask for which restricted graph classes a MAD tree can be found in polynomial time. In [10], an algorithm is exhibited that computes a MAD tree of a given distance-hereditary graph in linear time. Recently, Dahlhaus et al. [11], have design 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 neighbourhood a MAD tree of a vertex weighted interval graph. Olario et al. [13], have design optimal parallel algorithms for problems modelled by a family of intervals on interval graphs to compute a MAD tree.

Recently, Jana et al. [14], and Mondal [15] have designed an efficient algorithms to compute minimum average distance spanning tree on permutation graphs and trapezoid graphs respectively in $O(n^2)$ time, where n is the number of vertices of the graph.

1.2 Applications of the problem

MAD trees, also referred to as minimum routing cost spanning trees, are of interest in the design of communication networks [8]. 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.

1.3 Our result

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

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 3, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

1.4 Organization of the paper

In the next section, i.e., Section 2, we present the construction of the tree. In Subsection 2.1, we present BFS tree on circular-arc graph and in Subsection 2.2, we present the minimum diameter spanning tree. In Subsection 2.3, we develop modified spanning tree of the tree T. In Section 3, we discuss about average distance. In subsection 3.1, we present the algorithm to compute average distance. In Section 4, we present an algorithm to get MAD tree of the circular-arc graph. The time complexity is also calculated in this section and Section 6 presents conclusion of the work.

II. CONSTRUCTION OF THE TREE

2.1 Construction of BFS tree on circular-arc 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.

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 [3]. To construct this BFS tree on a circular-arc graph we consider the circular-arc diagram. We first number the vertices of the circular-arc graph successively by *1, 2, 3, ..., n* corresponding to the circular arcs C_1, C_2, ...,C_n according to the order of end pointst_i (the tail) of the circular arcsrespectively when it is traversed in clockwise manner. To construct the rooted tree, we traverse the graph in anticlockwise manner. Firstly, we placed the vertex corresponding to the maximum length of the arcs C_i (i ≠1), which are adjacent to the arc C_1, as a root and put in zero level. Then we traverse all vertices (corresponding to arcs) adjacent to $C_i$and placed them on the first level. Next we traverse the circular-arc diagram step by step until all vertices corresponding to the arcs are traversed. In this way, we get a left BFS tree, denoted by $T_L(i)$. Secondly, we construct BFStree by same manner traversed in clockwise direction of the vertex corresponding to the arc C_1placed the maximum length of the arcs C_j(j ≠ 1), which are adjacent to the arc C_1, as a root and put in zero level and all vertices (corresponding to arcs) adjacent to $C_j$on the first level. Next we traversed the circular-arc diagram step by step. In this way, we get a right BFS tree, denoted by $T_R(j)$. Figure 3(a) and 3(b) represent the left BFS trees $T_L(7)$ rooted at the vertex 7 corresponding to the arc $C_7$and a right BFS tree $T_R(2)$ rooted at the vertex 2 corresponding to the arc C_2, respectively of the circular-arc graph shown in Figure 1. By the following algorithm one can design the BFS tree on circular-arc graphs.

Now, we define *level* of the vertex *v* as the distance of *v* from the root *i* of the BFS tree *T(i)* and denote it by *level*(*v*), $v \in V$ and take the level of the root *i* as 0. The level of each vertex of the tree *T*can be computed in $O(n)$ time.

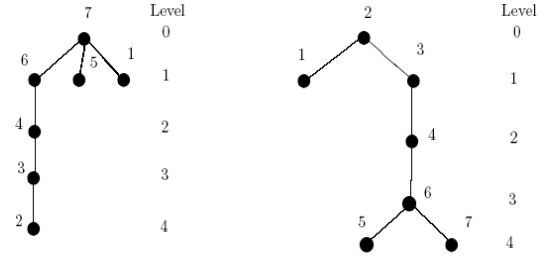


Fig.3. BFS trees $T_L(7)$ and $T_R(2)$ rooted at vertices 7 and 2 respectively of the circular-arc graph shown in Figure 1.

Algorithm CARBFS-TREE

Input: Sorted arcs C_i, *i = 1, 2, ..., n* with endpoints (h_i,t_i) of the circular-arc diagram of thecircular-arc graph *G = (V, E).*

Output: BFS tree with root *j*, $T_R(j)$.

Step 1: Compute the adjacent arcs to the arc $C_1$and select the arc of maximum tails t_j of thoseadjacent arcs in clockwise sense.

Step 2: Choose the vertex *j* corresponding to the arc $C_j$as root of the tree. Then find the adjacentvertices to *j* and placed them as leaves at level 1, and mark them.

Step 3: Let *k* be the vertex corresponding to the arc C_k with maximum tail t_k among the tails ofadjacent arcs to C_j . Then put *k* as node on main path.

Next find all other unmarked adjacent arcs to $C_k$and they are placed as leaves at *level(k)+1*.

Mark them.

Step 4: This process continued until all arcs are marked.

end CARBFS-TREE.

By similar way we can construct the BFS tree with root *i* in anticlockwise manner, i.e. $T_L(i)$. The time complexity of the Algorithm CARBFS-TREE is stated below.

Theorem 1: The BFS trees $T_R(j)$ and $T_L(i)$ rooted at any vertex $x \in V$ can be computed in *O(n)* timefor a circular-arc graph containing *n* vertices.

Proof. Step 1 of the above algorithm can be computed in *O(n)* time, because sorted arcs and adjacent arcs are finite. In Step 2, selection of the root *j* takes *O(1)* time and to mark the vertices adjacent to *j*takes *O(n)* time. Therefore, Step 2 runs in *O(n)* time. Also computation of the vertex *k*corresponding to the maximum tail among the tails of the arcs adjacent to the arc $C_j$takes *O(n)* time. So, Step 3 takes *O(n)* time. Since Step 4 is the checking step, so it takes *O(n)* time. Hence, over all the time complexity of Algorithm **CARBFS-TREE** is *O(n)* time, where *n* is the number of vertices. □

Obviously, two BFS trees, each contains *n* vertices and $(n-1)$ edges corresponding to the given circular-arc graph with *n* vertices. So, it is a spanning tree.

2.2 Computation of minimum diameter spanning tree

Let $T_L(i)$ and $T_R(j)$ be two BFS trees of a circular-arc graph *G*. Next we determine the diameters of the trees $T_L(i)$ and $T_R(j)$. If the diameter of $T_L(i)$ is less than the diameter of $T_R(j)$, then we denote $T_L(i)$ by*T* otherwise $T_R(j)$ by *T*, i.e., *T* is the tree of minimum

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 3, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

diameter between the trees $T_L(i)$ and $T_R(j)$.

Let P be the path in the BFS tree T with maximum length of the circular-arc graph G, defined as main path and it is denoted by $u_1^* \rightarrow u_2^* \rightarrow u_3^* \rightarrow \ldots\ldots \rightarrow u_K^*$ corresponding to the arcs $C_1^*, C_2^*, C_3^*, \ldots\ldots, C_K^*$, where $k \leq n$. Also, u_i^*, $i = 1, 2, 3, \ldots, k$ are nodes in the main path P. Now, we define some more terms below.

The *open neighbourhood* of the vertex u_i^* in the path P of G, denoted by $N(u_i^*)$ and defined as

$N(u_i^*) = \{u'' : (u'', \quad u_i^*) \in E\}$

and the *closed neighbourhood*

$N[u_i^*] = N(u_i^*) \cup \{u_i^*\}$,

where E is the edge set of the given circular-arc graph.

As per construction of BFS tree we have the following important results in BFS tree T.

Lemma 1: 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, except such $(u, v) \in E$ in which $level(u) = 1$ and $level(v) = k$, where k is the highest level.

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 the idea of breath first search, at any stage if u and v are theadjacent to the previously visited vertex, then u and v to be placed in same level, so $level(u) = level(v)$.But, if u is adjacent to a previously visited vertex then v must be adjacent to next visited vertex,and then v to be placed in the next level in T. So, in this case $|level(u) - level(v)| = 1$. Thus, either$level(u) = level(v)$ or $|level(u) - level(v)| = 1$ implies $(u, v) \in E$, which is contradictory to the assumption $|level(u) - level(v)| > 1, (u, v) \in E$.

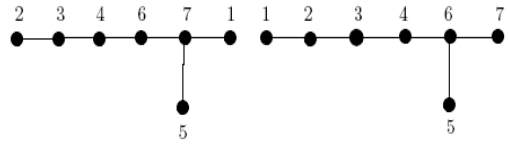
By the process of the ordering of the arcs of the circular-arc graphs it is evident that there is at least one arc which is extended on both sides of the fixed line (dotted line in Figure 2) from which order of the arcs begins. So one end vertex of the edge $(u, v) \in E$ of $G$is at either in first level, i.e., at level 1 or in highest level, i.e., at level k.Hence the result.□

Now, we shall prove that the BFS tree is a minimum diameter spanning tree.

Lemma 2. 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 covers thewhole circle with least number of arcs. This diameter is minimum, because T is the minimum height tree. Also, T is a spanning tree. Hence $T$is a minimum diameter spanning tree. □

In next subsection, we shall discuss about the modified spanning tree T′of the spanning tree T.



(a) (b)

Fig.4. Isomorphic trees $T_{1L}(7)$ and $T_{1R}(2)$ of the BFS trees $T_L(7)$ and $T_R(2)$ respectively, where main path is horizontal.

2.3 Modification of the spanning tree T

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

First, we draw the tree T_1 (shown in Figure 4(a) and Figure 4(b)) whose main path is horizontal and isomorphic to the tree T. Then, we compute $N(u_i^*) \in G$ for each vertices on the main path P. If there are any common adjacent vertices of two nodes $u_i^*$and u_{i+1}^* in the main path P, where $i = 1, 2, \ldots, k-1$ then we can shift them by the following way.

Step I: In G, if any common adjacent vertex w of $u_i^*$and u_{i+1}^*exist, then we calculate the number of vertices k_1, k_2 respectively, onthe both sides separately of the node u_i^*(taken as fixed and leaves which does not lie on main path are not countable) in T_1along the main path.

Next, calculate their difference, say,

$d_1 = |k_1 - k_2|$.

Step II: Again, find the number of vertices on the both sides separately of the node u_{i+1}^*(taken as fixed) in T_1along the main path(ignoring the vertex obtained in Step I).

Next, calculate their difference, say, d_2.

Step III: Case-I: If $d_1 - d_2 < 0$, then unaltered, i.e. w remains adjacent of u_i^*in T_1.

Case-II: If $d_1 = d_2$, then calculate

$D_1 = \sum_{u,v \in V(T_1)} d_{T_1}(u, v)$

(total distance before shifting) and

$D_2 = \sum_{u,v \in V(T_1)} d_{T_1}(u, v)$

(total distance after shifting).

If $D_1 < D_2$then, tree remains unaltered else w is shifted.

Case-III: If $d_1 - d_2 > 0$, then the adjacent vertex w of the node u_i^* is shifted to the nodeu_{i+1}^*,i.e.,$w$is finally adjacent to u_{i+1}^*in T_1.

Step IV: Finally represent T_1as form of tree T'.

Similar idea is used for pair of any two nodes on the main path P. Using this method we construct the modified spanning tree T' starting from T with the help of T_1. Figure 5(a) and Figure 5(b) are the modified BFS trees $T'_L(7)$ and $T'_R(2)$ of the BFS trees $T_L(7)$ and $T_R(2)$ respectively (shown in Figure 3(a) and Figure 3(b) respectively).

Lemma 3: If $T'$is a BFS tree, then the distance $d_{T'}(u, v)$between the vertices u and v in T′ is given by

$$d_{T'}(u, v) = \begin{cases} level(v), & \text{if } u \text{ is a root and } v \text{ is any vertex,} \\ |level(u) - level(v)|, & \text{if } u \text{and v both are nodes in main path P,} \\ |level(parent(u)) - level(v)| + 1, & \text{if } u \text{ is any leaf and } v \text{ is any node in P.} \end{cases}$$

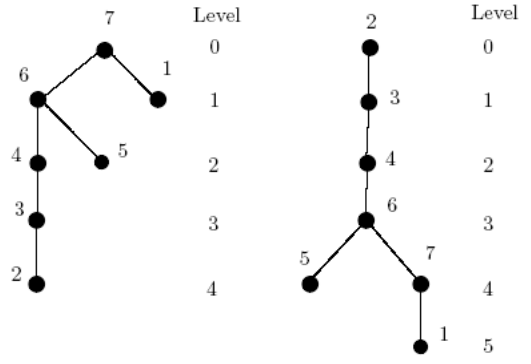
Proof. Case I: If $u$is a root.

In the tree T', with u as root there exists a unique shortest path $u \rightarrow z_1 \rightarrow z_2 \rightarrow \ldots\ldots \rightarrow z_{p-1} \rightarrow v$from u to any vertex $v \in$ G, where u is the parent of z_1 and z_i is the parent of z_{i+1} and so on for each $i = 1, 2, \ldots\ldots, p-2$ and z_{p-1} is the 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 $d(u, v) \leq p$.

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 3, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Next we are to show that $d_{T'}(u,v) \not< p$. If possible, let $d_{T'}(u, v) = q < p$. Then there exist apath $u \rightarrow y_1 \rightarrow y_2 \rightarrow \ldots\ldots \rightarrow y_{p-1} \rightarrow v$from u to any vertex v ∈ G. As each vertex of this path is directly connected with the next one, level(y_1) is either 0 or 1 since level(u) = 0 and 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 $d_{T'}(u,v) \not< p$, which implies $d_{T'}(u,v) = p$, i.e. $d_{T'}(u, v)$= *level(v).*



(a) (b)

Fig.5. Modified BFS trees $T'_L(7)$ and $T'_R(2)$ of the trees $T_L(7)$ and $T_R(2)$ respectively shown in **Fig. 3**.

Case II: If u and $v$both are nodes in the main path P.

If u and v both are the nodes in the main path P, then as per rule of construction of BFS, there is a shortest path$u \rightarrow z_1' \rightarrow z_2' \rightarrow \ldots\ldots \rightarrow z_{p-1}' \rightarrow v$. Here z_1' is at next level of u, z_2' is at the next level of $z_1'$and so on uptov. Let level of u be i, so $d(u, z_1') = 1 = (i + 1) - i = level(z_1') - level(u)$, $d(u, z_2') = d(u, z_1') + d(z_1', z_2') = 1 + 1 = 2 = (i + 2) - i = level(z_2') - level(u)$. If $d(u, z_k') = k = (i + k) - i = level(z_k') - level(u)$, then $d(u, z_{k+1}') = d(u, z_k') + d(z_k', z_{k+1}') = k + 1 = (i + k + 1) - i = level(z_{k+1}') - level(u)$. Hence
$d_{T'}(u, v) = |level(v) - level(u)|$.

Case III: If u is any leaf and v is any node in the main path P.In this case, 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 a node in the main path P (as per construction of BFS rooted at the vertex corresponding to the arc x with maximum length of the arc 1 in anticlockwise manner or the vertex corresponding to the arc y with maximum length of the arc 1 in clockwise manner). Therefore $d_{T'}(u, v)$= *d(u, parent(u)) +d(parent(u), v) = 1 + level(v) - level(parent(u)), i.e.,*$d_{T'}(u, v) = |level(parent(u)) - level(v)| + 1$.□

III. Average Distance

Many works on average distance in graphs are available in literature [16, 17, 18, 19, 20, 21]. Chung [22] give a bound of average distance of a graph in terms of independent number. She has shown that $\mu(G) \leq \alpha(G)$, where $\mu(G)$ and $\alpha(G)$ denote respectively the average distance and the independent number of the graph G. Also in [23], the average distance of an interval graph with edges of unit length can be computed in$O(m)$ time where m is the number of edges. In this section, we discuss about the computation of average distance of a circular-arc graph.

The average distance $\mu(G)$ of a connected circular-arc graph $G = (V, E)$ is the average over all unordered pairs of vertices of the distances,

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

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.

3.1 Algorithm to compute average distance and its computation

At first we compute $d_G(u, v)$ for every pair u, v $(u \neq v)$, then we compute the sum of distance between all pairs of vertices and finally we multiply it by the factor $\frac{2}{n(n-1)}$ to get the average distance. From above procedure it follows that the time to compute the average distance is same as the time to compute all pairs shortest distances. The average distance of a circular-arc graph can be computed in sequential using $O(n^2)$ time, where n is the number of vertices of the graphs.

IV. Algorithm and its Complexity

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

Algorithm CAMAD-TREE

Input: Sorted endpoints of the arcs $S = \{C_1, C_2, \ldots, C_n\}$of the circular-arc graph $G = (V, E)$.

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

Step 1: Compute the minimum diameterspanning tree$T$//by Section 3.2//
and
Calculate $level(u) = d(u', u)$,$u'$is either the vertex corresponding to the left maximum spread arc of the arc C_1 or the vertex corresponding to the right maximum spread arc of the arc C_1.

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

Step 3: //Modification of the tree T //
Compute common vertices, if any, of two 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,

International Journal of Electronics Communication and Computer Engineering
Volume 6, Issue 3, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Step 3.1.1: Calculate the numberof verticesto one side and other side of the internal node u_i in T with u_i as origin. Next calculate theirdifference, d_1.

Step 3.1.2: Calculate the numberof vertices to one sideand other side of theinternal node u_{i+1} in T with u_{i+1}as origin (ignoring the vertex w).
Next calculate their difference, d_2.

Step 3.1.3: If $d_1 - d_2 < 0$, thenunaltered;
if$d_1 - d_2 = 0$,then calculate
$D_1 = \sum d_T(u, v)$ (totaldistance before shifting) and
$D_2 = \sum d_T(u, v)$(total distance aftershifting) and
consider minimum$\{D_1, D_2\}$,
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 circular-arc graph G.

Step 4: Calculate

$$d_{T'}(u,v) = \begin{cases} level(v), & \text{if } u \text{ is a root and } v \text{ is any vertex,} \\ |level(u) - level(v)|, & \text{if } u \text{and v both are nodes in main path P,} \\ |level(parent(u)) - level(v)| + 1, & \text{if } u \text{ is any leaf and } v \text{ is any node in P.} \end{cases}$$

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

end CAMAD-TREE

4.1 Illustration of the algorithm

In Figure 4(a), $u_i^* = 7$ and$u_{i+1}^* = 6$ are two nodes. 5 is the common adjacent of the vertices u_i^* and u_{i+1}^*, i.e., $w = 5$. Taking the vertex 7 as fixed, the number of vertices on the both sides of the node 7 in $T_{1L}(7)$ along the main path are 4 and 1. Hence their difference is $d_1 = 4 - 1 = 3$. Again takingthe vertex 6 as fixed, the number of vertices on the both sides of the node 6 in $T_{1L}(7)$ along the mainpath is 3 and is 2(ignoring the vertex 5) when the vertex 5 is adjacent with the vertex 6. Hence theirdifference is $d_2 = 3 - 2 = 1$. Therefore $d_1 > d_2$. So, the vertex 5 is shifted to the node 6. Then we havethe modified spanning tree $T'_L(7)$(Figure 5(a)).

Next calculate the average distance $\mu_1(G)$ corresponding to the tree $T_L(7)$(isomorphic to $T_{1L}(7)$). Again calculate average distance $\mu'_1(G)$ corresponding to the tree $T'_L(7)$. Here $\mu_1(G) = 52/21$ and $\mu'_1(G) = 50/21$. Clearly, $\mu_1(G) > \mu'_1(G)$. Hence $T'_L(7)$ is a minimum average distance tree of the circular-arc graph G.

In Figure 4(b), which is the spanning tree $T_R(2)$ of the circular-arc graph G with vertex 2 corresponding to the arc C_2 with maximum length adjacent to the arc C_1 as root. $u_i^* = 2$ and $u_{i+1}^* = 7$ are two nodes. 1 is the common adjacent of the vertices u_i^* and u_{i+1}^*, i.e., $w = 1$. Taking the vertex 2 as fixed, the number of vertices on the both sides of the node 2 in $T_{1R}(2)$ along the main path are 0 and 5. Hence their difference is $d_1 = 5 - 0 = 5$. Again taking the vertex 7 as fixed, the number of vertices on the both sides of the node 7 in $T_{1R}(2)$ along the main path are 5 (ignoring the vertex 1) and 0 when the vertex 1 is adjacent with the vertex 7. Hence their difference is $d_2 = 5 - 0 = 5$. Therefore $d_1 = d_2$, i.e., $d_1 - d_2 = 0$. Then calculate $D_1 = 52$ (total distance before shifting) and $D_2 = 50$ (total distance after shifting) and then consider minimum$\{D_1, D_2\} = D_2$. So, the vertex 1 is shifted to the node 7. Then we have the modified spanning tree $T'_R(2)$ (Figure 5(b)).

Next calculate the average distance $\mu_2(G)$ corresponding to the tree $T_R(2)$(isomorphic to $T_{1R}(2)$). Again calculate average distance $\mu'_2(G)$ corresponding to the tree $T'_R(2)$. Here $\mu_2(G) = 52/21$ and $\mu'_2(G) = 50/21$.

Clearly, $\mu_2(G) > \mu'_2(G)$. Hence $T'_R(2)$ is the another minimum average distance tree of the circular-arc graph G.

Next we shall show that for any circular-arc graph, the tree designed by the **Algorithm CAMAD- TREE** represents a MAD tree.

Theorem 2: For any circular-arc graph, the tree designed by the **Algorithm CAMAD-TREE** is a MAD tree.

Proof. Let $G = (V, E)$ be any circular-arc graph. Then, using Section 2.2, one can design a minimum diameter spanning tree. In this minimum diameter spanning tree, shifting (if necessary, under conditions stated in Section 2.3) of the some vertices to its next adjacent node in the main path means that those vertices are placed on such side of the tree, with respect to the fixed node in the main path, in which that side contains maximum number of vertices. As a result, after all possible shifting of the vertices,the sum of total distances over all unordered pair of vertices decreases. Thus the average distance of the tree decreases. Hence, the tree designed by the **Algorithm CAMAD-TREE**is a MAD tree for any circular-arc graph. This completes the proof. □

Lastly, we describe the time complexity of the algorithm.

Theorem 3: The MAD tree of a circular-arc graph G with n vertices can be computed in $O(n^2)$ time.

Proof. Step 1 of **Algorithm CAMAD-TREE** takes $O(n)$ time (Theorem 1). Step 2, i.e. computationof open neighbourhood of all nodes on the main path 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 $O(n)$ time. But Step 3.1 repeats (k -1) times, so the 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.

V. CONCLUSION

In this paper, we proposed an efficient algorithm to compute a minimum average distance spanning tree

on circular-arc graphs which is designed based on BFS technique. The time complexity of this algorithm is of $O(n^2)$, where n is the number of vertices of the circular-arc graph. To the best of our knowledge, the complexity is not optimal, so one can try to improve this algorithm as extensive research work for optimal algorithm.

ACKNOWLEDGMENTS

We are very grateful to two anonymous referees for their many helpful and valuable comments to improve this paper. This work is in part supported by the UGC under the Major Research Project: F. No.41-764/2012 (SR) for second author.

REFERENCES

[1] Golumbic, M. C., Algorithmic graph theory and perfect graphs, Academic Press, New York, 2000.
[2] Stahl F., Circular genetic maps, J. Cell Physiol, 70 (Suppl. 1) (1967) 1 – 12.
[3] Tarjan, R. E., Depth rst search and linear graph algorithm, SIAM J. Comput., 2 (1972) 146 - 160.
[4] Tucker, A., Structure theorems for some circular-arc graphs, Discrete Mathematics, 7 (1974) 167 - 195.
[5] Tucker, A., An efficient test for circular-arc graphs, SIAM J. Comput., 9 (1980) 1 - 24.
[6] Deng, X., Hell, P. and Huang, J., Linear time representation algorithms for proper circular-arc graphs and proper interval graphs, SIAM J. Comput., 25 (1996) 390 - 403.
[7] Jana, B., Mondal, S. and Pal, M., Computation of inverse 1-center location problem on the weighted trees, CiiT International Journal of Networking and Communication Engineering, 4 (2012) 70 - 75.
[8] Johnson, D. S.,Lenstra, J. K. and Rinnooy-Kan, A. H. G., The complexity of the network design problem, Networks, 8 (1978) 279 – 285.
[9] Bang Ye We, Lancia, G., Bafna, V., Chao, K. M., Ravi, R. and Chuang Yi Tang, A polynomial time approximation scheme for minimum routing cost spanning trees, SIAM Journal on Computing, 29 (1999) 761 - 778.
[10] Dahlhaus, E., Dankelmann, P., Goddard, W. and Swart, H. C., MAD trees and distance-hereditary graphs, Discrete Applied Mathematics, 131 (2003) 151 - 167.
[11] Dahlhaus, E., Dankelmann, P. and Ravi, R., A linear time algorithm to compute a MAD tree of an interval graph, Information Processing Letters, 89 (2004) 255 - 259.
[12] Barefoot, C. A., Entringer, R. C. and Szekely, L. A., Extremal values of distances in trees, Discrete Applied Mathematics, 80 (1997) 37 - 56.
[13] Olariu, S., Schwing, J. and Zhang, J., Optimal parallel algorithms for problems modeled by a family of intervals, IEEE Transactions on parallel and Distributed Systems, 3 (1992) 364 - 374.
[14] Jana, B. and Mondal, S., Computation of a Minimum Average Distance Tree on Permutation Graphs, Annals of Pure and Applied Mathematics, 2 (No. 1) (2012) 74 - 85.
[15] Mondal, S., An efficient algorithm for computation of a minimum average distance tree on trapezoid graphs, Journal of Scientific Research and Reports, 2 (2013) 598 - 611.
[16] Althofer, I., Average distance in undirected graphs and the removal of vertices, J. Combin. Theory Ser. B, 48 (1990) 140-142.
[17] Bienstock, D. and Gyon, E., Average distance in graphs with removed elements, J. Graph Theory, 12 (1988) 375 - 390.
[18] Favaron, O., Kouider, M. and Maheo, M., Edge-vulnerability and mean distance, Networks, 19 (1989) 493 - 504.
[19] Hendry, G. R. T., On mean distance in certain classes of graphs, Networks, 19 (1989) 451 - 557.
[20] Mohar, B., Eigenvalues, diameter and mean distance in graphs, Graphs and Combinatories, 7 (1991) 53 - 64.
[21] Winkler, P., Mean distance in a tree, Discrete Applied Math., 27 (1990) 179 – 185.
[22] Chung, F. R. K., The average distance and independence number, J. Graph Theory, 12 (1988) 229- 135.
[23] Dankelmann, P., Computing the average distance of an interval graphs, Information Processing Letters, 48 (1993) 311 - 314.

AUTHOR'S PROFILE

Biswanath Jana

was born in Paschim Medinipur, West Bengal, India on 8th day of January, 1980. He received his Bachelor of Science degree with Honours in Mathematics from Vidyasagar University, West Bengal, India in 2002and he has passed Master of Science (Applied Mathematics) in Mathematics with first class from same University in 2004. He received his Bachelor of Education degree in 2006 from same University.
Currently he is an Assistant Teacher in Mathematics of Jhargram Bikash Bharati Sikshayatan, Jhargram, Paschim Medinipur, West Bengal, India.
Email: biswanathjana2012@gmail.com

Sukumar Mondal

was born in Paschim Medinipur, West Bengal, India on the fifth day of September, 1971. He received his Bachelor of Science degree With Honours in Mathematics(rank second) from Vidyasagar University, West Bengal, India in 1992 and he stood first class second in Master of Science (Applied Mathematics) degree in 1994 from same University. He received his Bachelor of Education degree in 1996 from same University. He joined the department of Applied Mathematics with Oceanology and Computer Programming, Vidyasagar University, Midnapore, West Bengal, India, as a Junior Research Fellow on August 1997 and started research work for his Ph. D. degree. He was also awarded an individual research fellowship by Council of Scientific and Industrial Research, Govt. of India in 1997. He is currently A Ph. D. candidate in the department of Applied Mathematics with Oceanology and Computer Programming, Vidyasagar University, West Bengal, India. He was a Reader of department of Mathematics of Yogada Satsangha Palpara Mahavidyalaya, Palpara, Purba Medinipur, West Bengal, India. He has guided one research scholars for Ph.D. degrees and has published more than 20 articles in international journals and the member of the editorial Boards of the journal 'Annals of Pure and Applied Mathematics'. His specializations include Computational Graph Theory. He is also a reviewer of several international journals. He is the author of the books "Advanced Analytical Statics" published by U. N. Dhaur and Sons, Kolkata and "Advanced Hydrostatics" published by "Amit Publication", Kolkata.
Currently he is Associate Professor of Dept. of Mathematics of Raja N. L. Khan Women's College, Medinipur, West Bengal.
Email: sm5971@rediffmail.com , India.

Prof. Madhumangal Pal

currently Professor of Applied Mathematics, Vidyasagar University, India. He has received Gold and Silver medals from Vidyasagar University for rank first and second in M.Sc. and B.Sc. examinations respectively. Also he received jointly with Prof. G.P. Bhattacherjee, "Computer Division Medal" from Institute of Engineers (India) in 1996 for best research work. He has successfully guided 19 research scholars for Ph.D. degrees and has published more than 150 articles in international journals, 20 articles in national journals, 11 articles in edited book and 26 articles in conference proceedings. His specializations include Computational Graph Theory, Fuzzy Correlation & Regression, Fuzzy Game Theory, Fuzzy Matrices, Genetic Algorithms and Parallel Algorithms. Prof. Pal is the Editor-in-Chief of "Journal of Physical Sciences", "Annals of Pure and Applied Mathematics" and member of the editorial Boards of the journals "International Journal of Fuzzy Systems & Rough Systems", "International Journal of Computer Science, Systems Engineering & Information Technology", "Advanced Modelling and Optimization", "International Journal of Logic and Computation", "ISRN Discrete Mathematics" and "International Journal of Engineering Science, Advanced Computing and Bio-Technology". He isalso a reviewer of several international journals. He is the author of the books "Fortran 77 with Numerical and Statistical Analysis" published by Asian Books, New Delhi, "Numerical Analysis for Scientists and Engineers" and "Classical Mechanics" published by Narosa New Delhi and Alpha Sciences, Oxford, U.K., "Engineering Mathematics –Series", PHI Learning, New Delhi, "C Programs including Numerical and Statistical Methods", Narosa. He organized several national seminars/ conferences/ workshop. Also, participated, delivered invited talks and chaired in national and international seminars/ conferences/ winter school/ refresher course in Indian and Abroad.