Chapter 4

Computation of a minimum average distance tree and inverse 1-center location problem on the circular-arc graphs*

4.1 Introduction

A graph $G(V, E)$ is said to be an Int for a finite family of set F which is non-empty if and only if there is one-to-one resemblance between the non-empty set F and vertex set V such that two sets in non-empty set F have a non-empty intersection if their corresponding vertices in vertex set V are adjacent to each other. We can say the non-empty set F an *intersection model* of G. We use $G(F)$ for denoting the intersection graph for G for an intersection model F. IntGs are very important in the study of algorithms of graph theory and its applications [49]. Some special sub-classes of IntGs are TraGs, CorGs, InvGs, CirGs, PerGs and so on. G is called a CirG for non-empty set F, if F is a family of arcs on a circle and F is said to be a *circular-arc model* of G. G is called an InvG for F, if on real line F is a family of line segments.

Let on a circle C, $S = \{C_1, C_2, \ldots, C_n\}$ be a family of n arcs . The *coordinate* is a positive integer which is assigned in each endpoint of the arcs. Each arc's endpoints are located on the perimeter of a circle C in the increasing order of values of the coordinates in clockwise direction. For resemblance, each arc C_i, $i = 1, 2, 3, \ldots, n$, is denoted by (h_i, t_i), where h_i

*A part of the work presented in this chapter is published in *International Journal of Electronics Communication and Computer Engineering*, 6(3) (2015) 384-390.

is head and t_i is tail respectively which represent the starting and the ending points of the arc when it is traversed in clockwise direction, starting with an random chosen point on C. Without loss of generality, we assume the following:
(i) no single arc covers the entire circle C by itself,
(ii) no two arcs can share a common endpoint,
(iii) $\cup_{i=1}^{n} C_i = C$ (otherwise, the problem becomes one on InvG),
(iv) the arcs are sorted in increasing values of t_i's, i.e. $t_i > t_j$ for $i > j$.

Also if h_i's and t_i's for $i = 1, 2, \ldots, n$ are all distinct integers, the family of the arcs S is called *canonical*.

A path of a graph is an alternating sequence of distinct nodes and edges and it begins and ends with nodes. The length of a path is total number of edges in the path. A path from the vertex i to the vertex j is a *shortest path* from i to j with shortest length. The shortest distance between the nodes i and j is denoted by $d_G(i, j)$.
Alternatively, we can define a CirG as follows:
A graph $G(V, E)$ is a CirG iff
(i) its nodes are circularly indexed as $v_1, v_2, \ldots, v_n$, 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 nodes in the graph G corresponding to the arcs C_i and C_j respectively.
It is noted that the arc C_i and the node v_i or i are the same thing.

The CirGs have several applications in traffic control, genetic research [95], compiler design, etc. This graph admits some interesting sub-classes:

(i) Proper Circular-Arc Graphs: A graph G is a Proper Circular-Arc (PCA) graph if there exists a circular-arc representation of graph G so that no arc is properly contained in any other arc.

(ii) Unit Circular-Arc Graphs: A graph G is a Unit Circular-Arc (UCA) graph if there exists a Circular-Arc representation of G such that all arcs are of same length. It can be proved that $UCA \subseteq PCA$. In [98], the author exposed that this inclusion is controlled. An example of a PCA graph but not a UCA graph has also been described by Golumbic [49].

(iii) Helly Circular-Arc Graphs: When each subfamily of it consisting of pairwise intersecting subsets has a common element, a family of subsets S gratifies the Helly property. So, if there is a Circular-Arc representation of G such that the arcs satisfy the Helly property, a graph $G(V, E)$ is a Helly Circular-Arc (HCA) graph.

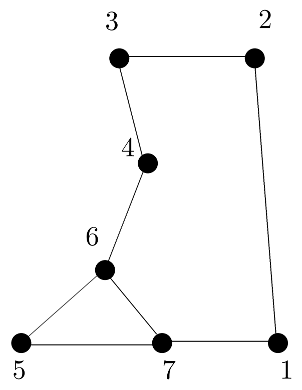

Figure 4.1: A CirG G.

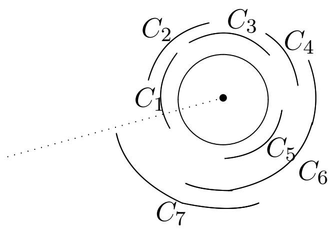

Figure 4.2: CirD of the CirG G of Figure 4.1.

(iv) Clique-Helly Circular-Arc graphs: If a graph $G(V,E)$ is a CirG and a Clique-Helly graph, then the graph G is a Clique-Helly Circular-Arc (CH-CA) graph . A graph G is Clique-Helly when its cliques satisfy the Helly property.

Tucker [99] discussed an $O(n^3)$ time algorithm to recognize a CirG. Also, Deng et al. [31] proposed an $O(n+m)$ time algorithms to recognize the PCA graphs and *proper interval graphs*. Several many other algorithms have been designed for circular-arc graphs [74, 75, 90, 92, 93].

Breadth-First-Search (BFS) is an approach to search in a graph where search is limited to two essential operations: (a) visit and review a vertex of a graph; (b) get access to visit the nodes that neighbour the currently visited vertex. Begining at a root node, BFS inspect all the neighbouring vertices. Then for every of those neighbour vertices, it traverses their neighbour vertices that are unvisited and so on.

Breadth First Search is an uniformed search technique that is aimed to expand every vertices of a graph or combination of sequences by systematically searching through every solution. It exhaustively searches the entire graph without considering the target until it finds a BFS Tree.

Let $G(V,E)$ be a undirected connected graph, v be a vertex of graph G and T is the spanning tree obtained by the BFS on graph G starting from the node v. An appropriate rooted tree $T(v) = (V, E^{'}) \subseteq G$ let us say a *Breadth-First-Search Tree* (BFS tree, in short) with the root v, the edges of graph G that do not appear in the BFS tree, we call them as non-tree edges.

The *average distance* $\mu(G)$ of graph $G(V, E)$ is the average over all unordered pairs of nodes of the distances,

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

The MADST of a CirG G is a spanning tree of graph G having minimum average distance. MADTs which are also referred to as minimum routing cost spanning trees, are of interest in design of the communication networks [55]. One is interested in designing a tree sub-network of a given network, such that on average, one can reach every node from every other node as fast as possible. In this chapter, we have designed
(i) an $O(n^2)$ time algorithm to construct a MADT for a given CirG G, and
(ii) an algorithm to compute Inv1C location problem on weighted CirGs in $O(n)$ time, where n is the number of vertices of the weighted CirG.

4.2 Organization of the chapter

In the next section, i.e. in Section 4.3 and its subsections we present the construction of the tree, MdsT, modified spanning tree of the tree T, an algorithm of MADT and its T-complexity of the unweighted CirG. In Section 4.4 and its subsections we present some notations, construction of minimum heighted tree, an algorithm of the modified vertex weighted tree corresponding to the CirG G. The T-complexity is also calculated in this section. In Section 4.5, we give the summary.

4.3 Minimum average distance tree on circular-arc graph

4.3.1 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 [96]. To construct this BFS tree on a CirG we consider the CirD. We first number the vertices of the CirG successively by $1, 2, 3, \ldots, n$ corresponding to the circular arcs $C_1, C_2, \ldots, C_n$ according to the order of ending points t_i (the tail) of the circular arcs respectively 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 $(i \neq 1)$, 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 CirD 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 BFS tree by same manner traversed in clockwise direction of the vertex corresponding to the arc C_1 placed the maximum length of the arcs C_j $(j \neq 1)$, 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 CirD step by step. In this way, we get a right BFS tree, denoted by $T_R(j)$. Figure 4.3(a) and 4.3(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 CirG shown in Figure 4.3. By the following algorithm one can design the BFS tree on CirGs.

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.

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

Algorithm CARBFS-TREE

Input: Sorted arcs C_i, $i = 1, 2, \ldots, n$, with endpoints (h_i, t_i) of the CirD of the CirG $G = (V, E)$.

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 those adjacent arcs in clockwise sense.
Step 2: Choose the vertex $j$ corresponding to the arc $C_j$ as root of the tree.
        Then find the adjacent vertices 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 of adjacent 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.

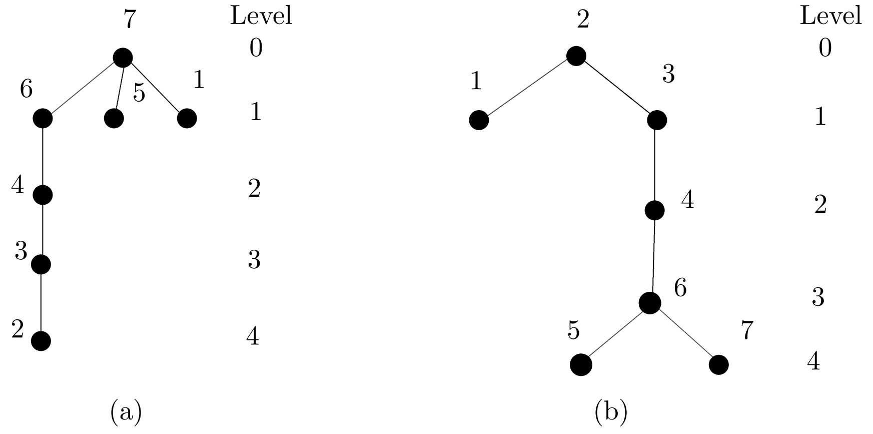

Figure 4.3: BFS trees $T_L(7)$ and $T_R(2)$ rooted at vertices 7 and 2 respectively of the CirG shown in Figure 4.1.

Theorem 4.3.1 *The BFS trees $T_R(j)$ and $T_L(i)$ rooted at any vertex $x \in V$ can be computed in $O(n)$ time for a CirG 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 T-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 CirG with n vertices. So, it is a spanning tree.

4.3.2 Computation of minimum diameter spanning tree

Let $T_L(i)$ and $T_R(j)$ be two BFS trees of a CirG 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 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 CirG G, defined as *main path* and it is denoted by $u_1^* \to u_2^* \to u_3^* \to \cdots \to u_k^*$ corresponding to the arcs $C_1^*, C_2^*, C_3^*, \ldots, C_k^*$ 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 $N(u_i^*) = \{u'' : (u'', u_i^*) \in E\}$ and the *closed neighbourhood* $N[u_i^*] = \{u_i^*\} \cup N(u_i^*)$, where E is the edge set of the given CirG.

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

Lemma 4.3.1 *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 the adjacent 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 CirGs it is evident that there is at least one arc which is extended on both sides of the fixed line (dotted line in Figure 4.2) 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 MdsT.

Lemma 4.3.2 *The spanning tree **T** is a MdsT.*

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 the whole 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 MdsT. □

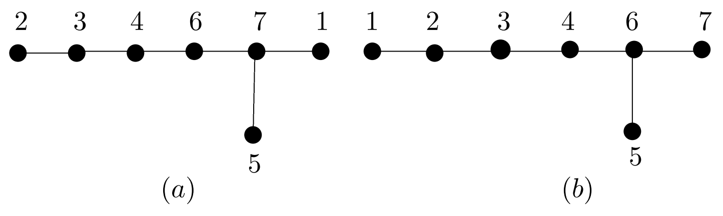

Figure 4.4: Isomorphic trees $T_{1L}(7)$ and $T_{1R}(2)$ of the BFS trees $T_L(7)$ and $T_R(2)$ respectively, where main path is horizontal.

In next subsection, we shall discuss about the modified spanning tree T' of the spanning tree T.

4.3.3 Modification of the spanning tree T

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

First, we draw the tree T_1 (shown in Figure 4.3(a) and Figure 4.3(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 $i = 0, 1, 2, \ldots, k-1$, 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, on the 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_1 along 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_1 along 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 node u_{i+1}^*. i.e. w is finally adjacent to u_{i+1}^* in T_1.

Step IV: Finally represent T_1 as 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 4.5(a) and Figure 4.5(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 4.3(a) and Figure 4.3(b) respectively).

Lemma 4.3.3 If $T^{'}$ is a BFS tree, then the distance $d_{T'}(u,v)$ between the vertices u and v in T' is given by

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

Proof. Case I: If u is a root.

In the tree $T^{'}$, with u as root there exists a unique shortest path $u \to z_1 \to z_2 \cdots \to z_{p-1} \to v$ 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 $i = 1, 2, \ldots, p-2$ 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 $d_{T'}(u,v) \leq p$.

Next we are to show that $d_{T'}(u,v) \nless p$. If possible, let $d_{T'}(u,v) = q < p$. 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, $level(y_1)$ 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 $d_{T'}(u,v) \nless p$ which implies that $d_{T'}(u,v) = p$, i.e. $d_{T'}(u,v) = level(v)$.

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 $u \to z_1^{'} \to z_2^{'} \cdots \to v$. Here $z_1^{'}$ is at next level of u, $z_2^{'}$ is at the next level of $z_1^{'}$ and so on up to v. 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. $d_{T'}(u, v) = |level(parent(u)) - level(v)| + 1$. □

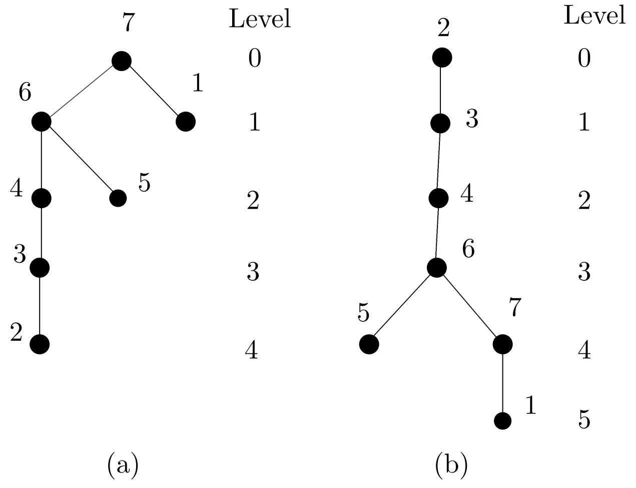

Figure 4.5: Modified BFS trees $T'_L(7)$ and $T'_R(2)$ of the trees $T_L(7)$ and $T_R(2)$ respectively shown in Figure 4.3.

4.3.4 Algorithm and time complexity on minimum average distance tree of circular-arc graph

Depend upon average distance, there are huge works of graphs in the literature [4, 10, 41, 53, 70, 100]. Chung [23] generate a bound of average distance of a graph in terms of independent number. She has presented that $\mu(G) \leq \alpha(G)$, where $\alpha(G)$ and $\mu(G)$ implies commonly the independent number and average distance of the graph G.

In [29], an InvGs average distance with the edges of unit length can be calculated in $O(m)$ time, where the number of edges implies m. Here, we describe about the evaluation of the average distance of a CirG.

In analytic networks, the average distance can be used as a tool, where the time of performance is proportional between any two node's distance . It is the measurement of the time needed in the average case, which opposed to the diameter, which determines the maximum performance time.

First we evaluate $d_G(u, v)$ for each pair u, v $(u \neq v)$, then we evaluate sum of the distances between all pairs of nodes and lastly we multiply it by $2/\{n(n-1)\}$ factor to get average distance. From the above technique, it follows that the time to evaluate the average distance is same as the time to evaluate all-pairs shortest distances.

Here, we propose an efficient algorithm to create MADT for a given CirG. Also, the correctness and the T-complexity of the algorithm are presented here.

Algorithm CAMAD-TREE

```
Input: Sorted endpoints of the arcs $S = \{C_1, C_2, \ldots, C_n\}$ of the CirD of the CirG $G = (V, E)$.
Output: The MADT $T^{'}$ and the average distance $\mu(T')$.
Step 1: Evaluate the MdsT $T$ //Subsection 4.3.2//
        and we calculate $level(u) = d(u^*, u)$, $u^*$ is either the vertex corresponding
        to the arc of maximum length of the arcs, which are adjacent to $C_1$ in
        clockwise manner or the vertex corresponding to the arc of maximum
        length of the arcs, which are adjacent to $C_1$ in anticlockwise manner.
Step 2: Evaluate $N(u_i^*) \forall u_i^* \in T$ and height of the tree $T$ is $k$.
Step 3: //Construction of Modified of the tree $T$//
        Evaluate $N(u_i^*)$ and $N(u_{i+1}^*)$ for $i = 0, 1, 2, 3, \ldots, k-1$ on the main
        path $P$. Let the tree $T_1$ whose main path is horizontal and
        isomorphic to the tree $T$.
    Step 3.1: For $i = 0, 1, \ldots k-1$ 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 number of nodes $k_1, k_2$ respectively, on the
        both sides separately of the node $u_i^*$ (taken as fixed) in $T_1$ along the
        main path. Next, evaluate their difference, say, $d_1 = |k_1 - k_2|$.
        Step 3.1.2: Calculate the number of nodes on the both sides
        separately of the node $u_{i+1}^*$ (taken as fixed) in $T_1$ along the main path
        (ignoring the vertex obtained in Step 1). Next, evaluate the difference,
        say, $d_2$.
        Step 3.1.3: If $(d_1 - d_2) < 0$, then unchanged;
        if $d_1 - d_2 = 0$ 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) and then
        consider $minimum\{D_1, D_2\}$;
        if $d_1 - d_2 > 0$, then the neighbour node $w$ of the vertex $u_i^*$ in the main path $P$ is
        shifted to the node $u_{i+1}^*$ in th main path $P$.
    Step 3.2: Fix $T^{'}$ as modified spanning tree of $T_1$ (isomorphic to $T$) of the
            CirG $G$.
```

Step 4: Calculate $d_{T'}(u,v)$ //Lemma 4.2.3//

where,

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

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

end CAMAD-TREE.

Illustration of the Algorithm CAMAD-TREE: In Figure 4.3(a), $u_i^* = 7$ and $u_{i+1}^* = 6$ are two vertices. 5 is the common adjacent of the vertices $u_i^* = 7$ and $u_{i+1}^* = 6$, i.e. $w = 5$. Taking the vertex 7 as fixed, the number of nodes 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 keeping the vertex 6 as fixed, the number of nodes on the both sides of the node 6 in $T_{1L}(7)$ along the main path is 3 and is 2(ignoring the vertex 5) when the vertex 5 is neighbour of the vertex 6. So, their difference is $d_2 = 3 - 2 = 1$. Therefore $d_1 > d_2$. So, the vertex 5 is shifted to the vertex 6. Now, we have the modification of spanning tree $T_L^{'}(7)$(Figure 4.5(a)).

Next the calculation of average distance $\mu_1(G)$ corresponding to tree $T_L(7)$ (isomorphic to $T_{1L}(7)$). Again the calculation of average distance $\mu_1^{'}(G)$ corresponding to 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 MADT of the CirG G.

In Figure 4.3(b), which is Spanning Tree $T_R(2)$ of the CirG 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 two vertices are $u_{i+1}^* = 7$. 1 is the common neighbour of the nodes $u_i^* = 2$ and $u_{i+1}^* = 7$, i.e. $w = 1$. Keeping the vertex 2 as fixed, number of nodes on 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 keeping the node 7 as fixed, number of nodes on 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 neighbour of the node 7. Therefore, 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 vertex 7. Now, we have the Modified Spanning Tree $T_R^{'}(2)$ (Figure 4.5(b)).

Then we calculate average distance $\mu_2(G)$ corresponding to tree $T_R(2)$ (isomorphic to $T_{1R}(2)$). Again, we calculate average distance $\mu_2^{'}(G)$ corresponding to 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 MADT of the CirG G.

Next, we will show that for any CirG, the tree designed by the **Algorithm CAMAD-TREE** represents a MADT.

Theorem 4.3.2 *For any CirG, the tree designed by the* ***Algorithm CAMAD-TREE*** *is a MADT.*

Proof. Let $G(V, E)$ be a CirG. Then, using Subsection 4.3.2, one can design a MdsT. In this MdsT, shifting (if necessary, under conditions stated in Subsection 4.3.3) 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 nodes decreases. Thus the average distance of the tree decreases. Hence, the tree designed by the **Algorithm CAMAD-TREE** is a MADT for any CirG. This completes the proof. □

Lastly, we discuss the T-complexity of this algorithm.

Theorem 4.3.3 *The MADT of a CirG G with n nodes can be evaluated in $O(n^2)$ time.*

Proof. Step 1 of **Algorithm CAMAD-TREE** takes $O(n)$ time (Theorem 4.3.1). To compute the open neighbourhood of all vertices on the main path in Step 2 can be evaluated in $O(n^2)$ time. $O(n)$ time is needed for computing each Step 3.1.1 and Step 3.1.2. Step 3.1.3 runs in $O(n)$ time. But as, Step 3.1 repeats $(k-1)$ times, the total T-complexity of Step 3.1 is $O(n^2)$, where k is of $O(n)$. Again in Step 3.2, i.e. alteration of the tree can be evaluated in $O(n^2)$ time. Step 4 can be evaluated in $O(n^2)$ time in worst case. Therefore, the overall T-complexity of the proposed algorithm is $O(n^2)$ time. □

In the next section we consider another problem on weighted circular-arc graphs.

4.4 Inverse 1-center location problem on the weighted circular-arc graphs

In this section we discuss about Inv1C. Now, before going to our proposed algorithm we introduce some notations for our algorithmic purpose. Let i be the pre-specified vertex in G.

R_i	:	Longest path right to the vertex i.
L_i	:	Longest path left to the vertex i.
$w(R_i)$	:	Sum of weights of the vertices except the vertex i of the path R_i.
$w(L_i)$	:	Sum of weights of the vertices except the vertex i of the path L_i.
$w_{low}(v)$	:	Minimum weight of the vertex in the graph G.
$w_{upp}(v)$	:	Maximum weight of the vertex in the graph G.
w_{min}	:	$min\{w(L_i), w(R_i)\}$.
w_{max}	:	$max\{w(L_i), w(R_i)\}$.
w_1	:	$min\{w(v), v \in G\}$.
w_2	:	$max\{w(v), v \in G\}$.
k_1	:	The number of vertices in such path between L_i, R_i whose
	:	weight is maximum, except the vertex i.
k_2	:	The number of vertices in such path between L_i, R_i whose
	:	weight is minimum, except the vertex i.
T_{CIR}	:	Weighted tree corresponding to the circular-arc graph G.
T'_{CIR}	:	Modified tree of the tree T_{CIR} corresponding to the circular-arc graph G.
$w^*(R_i)$	:	Sum of weights of nodes except the node i of the path R_i after modification.
$w^*(L_i)$	:	Sum of weights of nodes except the node i of the path L_i after modification.

Figure 4.6 represents the weighted CirG G and Figure 4.7 is the corresponding matching diagram of that CirG G.

4.4.1 Construction of minimum height tree

Let i be pre-specified vertex which to be Inv1c. Here, our aim is to form a minimum height tree, as root i, with two branches of level difference either zero or one.

Let the vertex i be the root of the tree. Then we find all adjacent vertices to i corresponding to the arc C_i and set them as child (leaves) of i. Next consider the vertices j and k whose

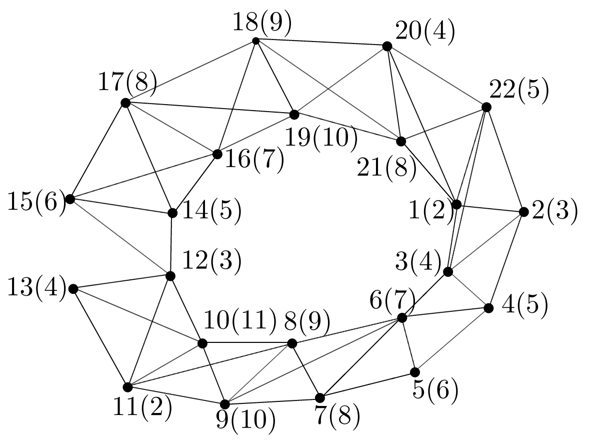

Figure 4.6: A CirG G.

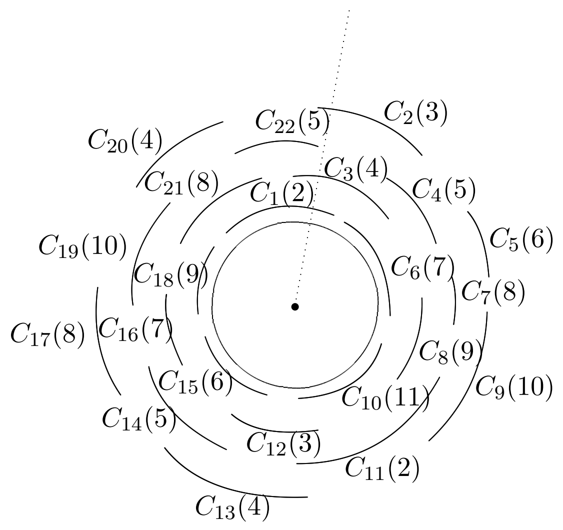

Figure 4.7: CirD of the CirG G of Figure 4.6.

maximum head h_j and minimum tail t_k respectively among the adjacent arcs of C_i and set them as a vertices on the main path and marked them. Next find all adjacent arcs to the vertices j and k and set them as respective child (leaves). This process is continue until all arcs are marked. In this way we construct a rooted tree with two branches with level difference either zero or one.

The proposed combinatorial algorithm to construct the tree T_{CIR} is as follows:

Algorithm CIR-TREE

Input: Sorted arcs C_i, $i = 1, 2, \ldots, n$, with endpoints (h_i, t_i) of CirD of the CirG $G = (V, E)$ and $w_i > 0$, the weight of the arc C_i,$(i = 1, 2, 3, 4, \ldots, n)$ for each i.

Output: Tree T_{CIR} with root i.

Step 1. Set fixed node i as root, and put level 0.

```
Step 2. Compute the open neighbourhood of $i = N(i) = \{v : (v, i) \in E\}$.
Step 3. Set $N(i)$ as the child of the root $i$ then marked them and put them at
        level 1.
Step 4. Set $j = max\{h_j : (j, i) \in E\}$ and $k = min\{t_k : (k, i) \in E\}$.
Step 5. Set $N(j)$ and $N(k)$ as the child of $j$ and $k$ respectively on both sides
        simultaneously put them at next level, i.e. at level 2 and marked them.
Step 6. This process will be continued until all arcs are marked and checked whether
        level difference is 0 or 1.
Step 7. Put weight $w_j$ to the each vertex $j$, $(j = 1, 2 \ldots, n)$ in $T_{CIR}$.
end CIR-TREE.
```

Illustration of the Algorithm CIR-TREE : Let $i = 1$ be the pre-specified vertex which is the root whose level is 0. Next the open neighbourhood of 1 is $N(1) = \{2, 3, 20, 21, 22\}$, where the vertices of $N(1)$ as the child of the root 1 and put them at level 1. Next, 3 has the maximum head among the arcs of $N(1)$ corresponding to nodes of graph G and 21 has the minimum tail among the arcs of $N(1)$ corresponding to nodes of graph G. Next the open neighbourhoods of 3 and 21 are $N(3) = \{4, 6\}$ and $N(21) = \{18, 19\}$ respectively, where the vertices of $N(3)$ and $N(21)$ as the child of the roots 3 and 21 and put them at level 2. Next 6 has the maximum head among the arcs of $N(3)$ corresponding to nodes of graph G and 18 has the minimum tail among the arcs of $N(21)$ corresponding to nodes of graph G. Next the open neighbourhoods of 6 and 18 are $N(6) = \{5, 7, 8, 9\}$ and $N(18) = \{16, 17\}$ respectively, where the vertices of $N(6)$ and $N(18)$ as the child of the roots 6 and 18 and put them at level 3. Next 9 has the maximum head among the arcs of $N(6)$ corresponding to nodes of graph G and 16 has the minimum tail among the arcs of $N(18)$ corresponding to the vertices of the graph G. Next the open neighbourhoods of 9 and 16 are $N(9) = \{10, 11\}$ and $N(16) = \{14, 15\}$ respectively, where the vertices of $N(9)$ and $N(16)$ as the child of the roots 9 and 16 and put them at level 4.

Next 11 has the maximum head among the arcs of $N(9)$ corresponding to nodes of the graph G and 15 has the minimum tail among the arcs of $N(16)$ corresponding to the vertices of the graph G. Next the open neighbourhoods of 11 and 15 are $N(11) = \{12, 13\}$ and $N(15) = \{\} = \phi$ respectively, where the vertices of $N(11)$ as the child of the root 11 and put them at level 5. Next 13 has the maximum head among the arcs of $N(11)$ corresponding to nodes of graph G. In this way we get longest left path L_i from the vertex 1 to other vertex 15, i.e. the path $1 \to 21 \to 18 \to 16 \to 15$ and find longest right path R_i from 1 to the vertex 13 does not contain any vertex of the path L_i except 1, i.e. the

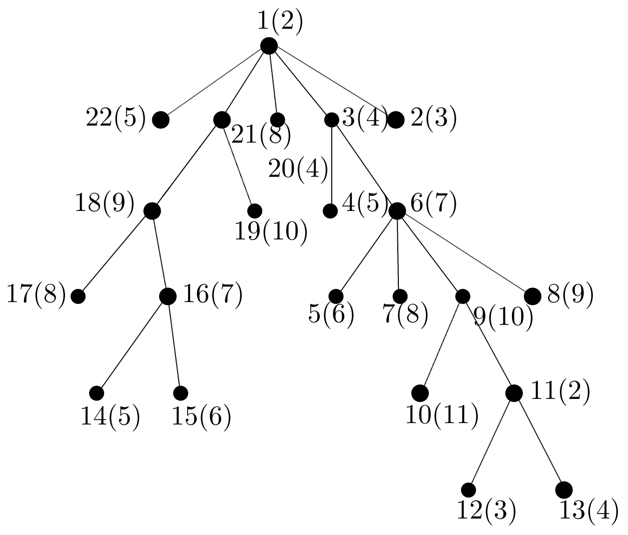

Figure 4.8: Tree T_{CIR} of the CirG G of Figure 4.6.

path $1 \to 3 \to 6 \to 9 \to 11 \to 13$. Hence the level difference of two paths L_i and R_i is 1. Next put the weights $2, 5, 8, 4, 4, 3, 9, 10, 5, 7, 8, 7, 6, 8, 10, 9, 5, 6, 11, 2, 3, 4$ to the vertices $1, 22, 21, 20, 3, 2, 18, 19, 4, 6, 17, 16, 5, 7, 9, 8, 14, 15, 10, 11, 12, 13$ respectively. Finally we construct the rooted tree T_{CIR} with root $i = 1$ (Figure 4.8).
Now we have the following important observation on T_{CIR}.

Lemma 4.4.1 *The tree T_{CIR} formed by the **Algorithm CIR-TREE** is a spanning tree.*

Proof. As per construction of the graph T_{CIR} by right and left end points (h_i, t_i) for $i = 1, 2, 3, 4, \ldots, n$, scanning approach of the arcs in CirD we get n vertices and $(n-1)$ edges. Also there is no repetition of the vertices, as we search only unmarked vertices, so this is a graph without any circuit. Therefore the tree T_{CIR} is a spanning tree.
Hence the result. □

Lemma 4.4.2 *The tree T_{CIR} formed by the **Algorithm CIR-TREE** is a BFS tree with minimum height.*

Proof. Actually steps of the algorithm indicates the steps of BFS technique in CirG. Thus the tree created by the *Algorithm CIR-TREE* is BFS tree. Again, we traverse the CirG both sides simultaneously with respect to maximum head and minimum tail until all unmarked arcs are marked. As in each step we move both sides on circle, so, its height to be minimum.
□

Also the T-complexity of the **Algorithm CIR-TREE** to evaluate the tree T_{CIR} is given below:

Theorem 4.4.1 *The T-complexity of the **Algorithm CIR-TREE** is $O(n)$, where n is number of nodes of the tree.*

Proof. Step 1 and Step 2 each takes $O(n)$ time, since the arcs are already sorted and root is selected from n arcs. Step 3 can be computed in $O(n)$ time, since n is the number of arcs. Since the end points of the arcs are sorted, so the maximum element (vertex) from a set of nodes can be evaluated in $O(n)$ time. Again intersection of two finite sets of n elements (number of vertices) can be executed in $O(n)$ time. Thus Step 4, Step 5 and Step 6 can be evaluated in $O(n)$ time. Since weight of the each vertex in tree T_{CIR} corresponds the weight of arcs in CirG is placed on the corresponding vertex, so, Step 7 can be executed in $O(n)$ time. Therefore, overall T-complexity of the proposed **Algorithm CIR-TREE** is $O(n)$ time, where n is number of the nodes of the weighted CirG. □

Thus the tree T_{CIR} of the CirG is formed. The tree T_{CIR} of CirG G (Figure 4.6) is shown in Figure 4.8.

To find the Inv1C, we discuss following cases:

1. If sum of weights of one side of the vertex i is equal to the sum of weights of other side, i.e. $w(L_i) = w(R_i)$, then i is the center as well as the Inv1C of the graph.

2. If $w(L_i) \neq w(R_i)$, then we have following six cases :

Case-2.1. : When w_{min} is equal to the product of the number of nodes except the vertex i in the path whose weight is maximum and minimum weight of the vertex in the graph, i.e. $w_{min} = k_1 w_1$.

Case-2.2. : When w_{min} is greater than the product of number of the nodes except the vertex i in the path whose weight is maximum and minimum weight of the vertex in the graph, i.e. $w_{min} > k_1 w_1$.

Case-2.3. : When w_{min} is less than the product of the number of nodes except the vertex i in the path whose weight is maximum and minimum weight of the vertex in the graph, i.e. $w_{min} < k_1 w_1$.

Case-2.4. : When w_{max} is equal to the product of the number of nodes except the vertex i in the path whose weight is minimum and maximum weight of the vertex in the graph, i.e. $w_{max} = k_2 w_2$.

Case-2.5. : When w_{max} is greater than the product of number of the nodes except the vertex i in the path whose weight is minimum and maximum weight of the vertex in the graph, i.e. $w_{max} > k_2 w_2$.

Case-2.6. : When w_{max} is less than the product of the number of nodes except the vertex i in the path whose weight is minimum and maximum weight of the vertex in the graph, i.e.

$w_{max} < k_2 w_2$.

Under above conditions we modify the tree T_{CIR} with the help of following non-linear semi-infinite (or nonlinear) optimization model:

$$\text{Minimize} \sum_{v \in V(T_{CIR})} \{c^+(w(v))x(w(v)) + c^-(w(v))y(w(v))\}$$

subject to

$$\begin{aligned}
&\max_{v \in V(T_{CIR})} d_{\overline{w}}(v,i) \leq \max_{v \in V(T_{CIR})} d_{\overline{w}}(v,p), \forall p \in T_{CIR} (\text{or}\, p \in V(T_{CIR})),\\
&\overline{w}(v) = w(v) + x\{w(v)\} - y\{w(v)\}, \forall v \in V(T_{CIR}),\\
&x\{w(v)\} \leq w^+\{w(v)\}, \forall v \in V(T_{CIR}),\\
&y\{w(v)\} \leq w^-\{w(v)\}, \forall v \in V(T_{CIR}),\\
&x\{w(v)\}, y\{w(v)\} \geq 0, \forall v \in V(T_{CIR}),
\end{aligned}$$

where $\overline{w}(v)$ be the modified vertex weight, $w^+\{w(v)\} = w_{upp}(v) - w(v)$ and $w^-\{w(v)\} = w(v) - w_{low}(v)$ are the maximum feasible amounts by which $w(v)$ can be increased and reduced consecutively, i.e. $w_{low}(v) \leq \overline{w}(v) \leq w_{upp}(v)$, $x\{w(v)\}$ and $y\{w(v)\}$ are the maximum amounts by which the vertex weight $w(v)$ is increased and reduced consecutively, $c^+(w(v))$ is the non negative cost if $w(v)$ is increased by one unit and $c^-(w(v))$ is the non negative cost if $w(v)$ is reduced by one unit. Each feasible solution (x, y) with $x = \{x(w(v)) : v \in V(T_{CIR})\}$ and $y = \{y(w(v)) : e \in V(T_{CIR})\}$, is called a feasible modification of the Inv1c location problem.

Now, we prove the results follows below.

Lemma 4.4.3 *If $w_{min} = k_1 w_1$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices except the vertex i, i.e. root i up to minimum weight maintaining the bounding condition in the path whose weight is maximum and i is the Inv1C.*

Proof. If k_1 be the number of nodes in the maximum weighted path L_i or R_i and w_1 be the minimum weight of vertex among the nodes in T_{CIR} as well as L_i or R_i, then there is a scope to reduce weight of each vertex up to w_1. As k_1 vertices is there in the path L_i or R_i, so we can reduces at least $k_1 w_1$ weight and hence reduced weight of the path L_i or R_i becomes $k_1 w_1$. Again we have $w_{min} = k_1 w_1$. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{CIR}, say T'_{CIR}. Again, since the CirG is an arbitrary, so our assumption is true for any CirGs.

Finally in T'_{CIR}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the Inv1C of the given weighted CirG. Hence the result. □

Lemma 4.4.4 *If $w_{min} > k_1w_1$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of some vertices except the root i maintaining the bounding condition in the path whose weight is maximum and i is the Inv1C.*

Proof. Since we can decrease the weight of each vertex except the root up to minimum weight of the vertex in T_{CIR}, so we can reduce the weight in the path whose weight is maximum in such a way that its least weight of the path becomes k_1w_1. Again we have $w_{min} > k_1w_1$. Therefore we can decrease the weights $(w_{max} - w_{min})$ from the vertices except the root i in the path whose weight is maximum using the non-linear semi-infinite (or nonlinear) optimization model technique. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{CIR}, say T'_{CIR}. Again, since the CirG is an arbitrary, so our assumption is true for any CirGs.

Finally in T'_{CIR}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the Inv1C of the given weighted CirG. Hence the result. □

Lemma 4.4.5 *If $w_{min} < k_1w_1$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices up to minimum weight except the root i maintaining the bounding condition in the path whose weight is maximum and enhance the weights of some vertices except the root i in the path whose weight is minimum and i is the Inv1C.*

Proof. Since we can decrease the weight of each node up to minimum weight of the node in T_{CIR}, so we can reduce the weights of the vertices except the root in the path whose weight is maximum in such a way that its least weight of the path becomes k_1w_1. Again we have $w_{min} < k_1w_1$. Therefore we can increase the weights $(k_1w_1 - w_{min})$ to the vertices except the root in the path whose weight is minimum using the non-linear semi-infinite (or nonlinear) optimization model technique. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{CIR}, say T'_{CIR}. Again, since the CirG is an arbitrary, so our assumption is true for any CirGs.

Finally in T'_{CIR}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the Inv1C of the given weighted CirG. Hence the result. □

Lemma 4.4.6 *If $w_{max} = k_2w_2$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding condition in the path whose weight is minimum and i is the Inv1C.*

Proof. If k_2 be the number of nodes in the minimum weighted path L_i or R_i and w_2 be the maximum weight of the vertex among the vertices in T_{CIR} as well as L_i or R_i, then there is a scope to increase the weight of each vertex up to w_2. As k_2 vertices is there in the path L_i or R_i, so we can enhance at most k_2w_2 weight and hence enhanced weight of the path L_i or R_i becomes k_2w_2. Again we have $w_{max} = k_2w_2$. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{CIR}, say T'_{CIR}. Again, since the CirG is an arbitrary, so our assumption is true for any CirGs.

Finally in T'_{CIR}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the Inv1C of the given weighted CirG. Hence the result. □

Lemma 4.4.7 *If $w_{max} > k_2w_2$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding condition in path whose weight is minimum and reducing the weights of some vertices except the root i in the path whose weight is maximum and i is the Inv1C.*

Since we can increase the weight of each node up to maximum weight of the node in T_{CIR}, so we can enhance the weights of all vertices except the root i in the path whose weight is minimum in such a way that its greatest weight of the path becomes k_2w_2. Again we have $w_{max} > k_2w_2$. Therefore we can reduces the weights $(w_{max} - k_2w_2)$ to some vertices except the root in the path whose weight is maximum using the non-linear semi-infinite (or nonlinear) optimization model technique. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{CIR}, say T'_{CIR}. Again, since the CirG is an arbitrary, so our assumption is true for any CirGs.

Finally in T'_{CIR}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the Inv1C of the given weighted CirG. Hence the result. □

Lemma 4.4.8 *If $w_{max} < k_2w_2$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of some vertices except the root i maintaining the bounding condition in the path whose weight is minimum and i is the Inv1C.*

Proof. Since we can increase the weight of each node up to maximum weight of the node in T_{CIR}, so we can enhance the weights of the vertices except the root i in the path whose weight is minimum in such a way that its greatest weight of the path becomes k_2w_2. Again we have $w_{max} < k_2w_2$. Therefore we can increase the weights $(w_{max} - w_{min})$ to some vertices except the root i in the path whose weight is minimum using the non-linear semi-infinite (or nonlinear) optimization model technique. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{CIR}, say T'_{CIR}. Again, since the CirG is an arbitrary, so our assumption is true for any CirGs.

Finally in T'_{CIR}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the Inv1C of the given weighted CirG. Hence the result. □

4.4.2 Algorithm and its complexity

Here, we have proposed a combinatorial algorithm for the Inv1c location problem on the vertex weighted tree T_{CIR}. The main concept of the proposed algorithm is as follows:

Let T_{CIR} be a weighted tree corresponding to the CirG G with $(n-1)$ edges and n vertices. Let V be the set of the vertices and E be the set of the edges. Let i be any non-pendant specified vertex in the tree T_{CIR} which is to be Inv1C. At first we calculate the path whose weight is maximum from i to any pendant vertex of T_{CIR}. Let R and L be the right and left paths from i in which weights are maximum with respect to sides. Let $w(L_i)$, $w(R_i)$ be the sum of the weights of vertices except the root of the paths L_i, R_i respectively with respect to the vertex i. If $w(L_i) = w(R_i)$, then i is the center as well as the Inv1C of the graph. If $w(L_i) \neq w(R_i)$, then six cases may arise. In the first case, if $w_{min} = k_1 w_1$ in T_{CIR}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of vertices in such path between L_i, R_i whose weight is maximum, except the root i and $w_{min} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices up to minimum weight except the vertex i, i.e.,root i maintaining the bounding condition in the path whose weight is maximum and i is the Inv1C. In the second case, if $w_{min} > k_1 w_1$ in T_{CIR}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of vertices in such path between L_i, R_i whose weight is maximum, except the root i and $w_{min} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of some vertices except the root i maintaining the bounding condition in the path whose weight is maximum and i is the Inv1C. In third case, if $w_{min} < k_1 w_1$ in T_{CIR}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of vertices in such path between L_i, R_i whose weight is maximum, except the root i and $w_{min} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices up to minimum weight except the root i maintaining the bounding condition in the path whose weight is maximum and enhance the weights of some vertices except the root i in the path whose weight is minimum and i is the Inv1C. In fourth case, if $w_{max} = k_2 w_2$ in T_{CIR}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of vertices in such path between L_i, R_i whose weight is minimum, except the root i and $w_{max} > 0$, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding condition in the path whose weight is minimum and i is the Inv1C. In fifth case, if $w_{max} > k_2w_2$ in T_{CIR}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of vertices in such path between L_i, R_i whose weight is minimum, except the root i and $w_{max} > 0$, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding condition in path whose weight is minimum and reducing the weights of some vertices except the root i in the path whose weight is maximum and i is the Inv1C. In sixth case, if $w_{max} < k_2w_2$ in T_{CIR}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of vertices in such path between L_i, R_i whose weight is minimum, except the root i and $w_{max} > 0$, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of some vertices except the root i maintaining the bounding condition in the path whose weight is minimum and i is the Inv1C.

Our proposed algorithm to the Inv1c location problem of the tree corresponding to the CirG G is as follows:

```
**Algorithm 1-INV-CIR-ARC-TREE**
Input: Weighted CirG $G$ with arcs $C_i$, $i = 1, 2, \ldots, n$, with endpoints $(h_i, t_i)$ and $w_j$ to the each vertex $j$, $(j = 1, 2 \ldots, n)$ are the weights of the arcs $C_i$.
Output: Vertex $i$ as Inv1C of the tree $T_{CIR}$ and modified tree $T'_{CIR}$.
Step 1. Construction of the tree $T_{CIR}$ with root $i$ //Algorithm CIR-TREE//.
Step 2. Compute the longest paths $R_i$ and $L_i$ from $i$ to the tree $T_{CIR}$.
Step 3. Calculate $w(L_i)$ and $w(R_i)$.
Step 4. //Modification of the tee $T_{CIR}$//
    Step 4.1. If $w(L_i) = w(R_i)$, then $i$ is the vertex one center as well as Inv1C
            of $T_{CIR}$.
    Step 4.2. If $w(L_i) \neq w(R_i)$, then
        Step 4.2.1. If $w_{min} = k_1w_1$ in $T_{CIR}$, then $w^*(L_i) = w^*(R_i)$ by reducing the
                weights of all vertices except the vertex $i$, i.e.,root $i$ up to minimum weight
                maintaining the bounding condition in the path whose weight is
                maximum, then go to Step 4.3.
        Step 4.2.2. If $w_{min} > k_1w_1$ in $T_{CIR}$, then $w^*(L_i) = w^*(R_i)$ by reducing the
                weights of some vertices except the root $i$ maintaining the bounding condition
                in the path whose weight is maximum, then go to Step 4.3.
        Step 4.2.3. If $w_{min} < k_1w_1$ in $T_{CIR}$, then $w^*(L_i) = w^*(R_i)$ by reducing the
                weights of all vertices except the root $i$ up to minimum weight maintaining
```

the bounding condition in the path whose weight is maximum and enhance the weights of some vertices except the root i in the path whose weight is minimum, then go to Step 4.3.

Step 4.2.4. If $w_{max} = k_2 w_2$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices except the root i up to maximum weight maintaining the bounding condition in the path whose weight is minimum, then go to Step 4.3.

Step 4.2.5. If $w_{max} > k_2 w_2$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices except the root i up to maximum weight maintaining the bounding condition in path whose weight is minimum and reducing the weights of some vertices except the root i in the path whose weight is maximum, then go to Step 4.3.

Step 4.2.6. If $w_{max} < k_2 w_2$ in T_{CIR}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of some vertices except the root i maintaining the bounding condition in the path whose weight is minimum, then go to Step 4.3.

Step 4.3. Modified tree T'_{CIR} of the tree T_{CIR} with $w^*(L_i) = w^*(R_i)$, and i is the Inv1C.

end 1-INV-CIR-ARC-TREE.

Using above **Algorithm 1-INV-CIR-ARC-TREE** we can find out the Inv1c location problem on any vertex weighted tree. Justification of this statement follows the following illustration.

Illustration of the Algorithm 1-INV-CIR-ARC-TREE to the tree T_{CIR} in Figure 4.8 **:** Let $i = 1$ be the pre-specified vertex of the tree T_{CIR} which is to be Inv1C. Next we find the longer left path L_i from the node 1 to other node 15, i.e. the path $1 \to 21 \to 18 \to 16 \to 15$ and find longest right path R_i from 1 to the vertex 13 does not contain any vertex of the path L_i except 1, i.e. the path $1 \to 3 \to 6 \to 9 \to 11 \to 13$. Next calculate the weights of the paths L_i and R_i. Let $w(L_i)$ and $w(R_i)$ are the sum of weights of nodes except the root $i = 1$ of the paths L_i and R_i respectively. Here $w(L_i) = 30$ and $w(R_i) = 27$. Therefore $w_{min} = w(R_i) = 27$ and $w_{max} = w(L_i) = 30$. Again $k_1 = 4$ and $w_1 = 2$, then $k_1 w_1 = 8$. Therefore $w_{min} > k_1 w_1$. Next calculate $(w_{max} - w_{min})$. Therefore $(w_{max} - w_{min}) = (30 - 27) = 3$. Therefore we can decrease the weights $(w_{max} - w_{min})$ from the vertices except the root i in the path whose weight is maximum using the non-linear semi-infinite (or nonlinear) optimization model technique. Now we subtract the weight 3 from the weight of the vertex 21 in L_i, then we get $w^*(L_i) = \{(8 - 3) + 9 + 7 + 6\} = 27$.

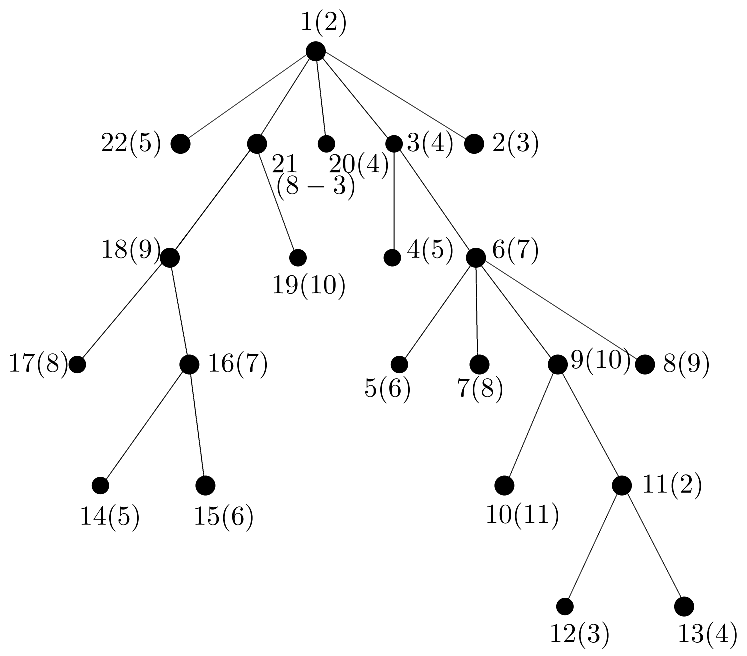

Figure 4.9: Modified tree T'_{CIR} of the tree T_{CIR}.

Again $w^*(R_i) = w_{min} = w(R_i) = 27$, hence we get $w^*(L_i) = w^*(R_i)$. Therefore the vertex 1 is the Inv1C.

Now we have the modified tree T'_{CIR} (Figure 4.9) with modified vertex weight.

Next we shall prove the following important result.

Lemma 4.4.9 *The* ***Algorithm 1-INV-CIR-ARC-TREE*** *correctly computes the Inv1C of the weighted CirG.*

Proof. Let i be the pre-specified vertex in T_{CIR}. We have to prove that i is the Inv1C. At first, by Step 1, we have constructed the tree T_{CIR} (as per Subsection 4.4.1) with root i, by Step 2, compute the longest paths R_i and L_i from i to the tree T_{CIR}, by Step 3, calculate the weight of the paths L_i and R_i from i except i, i.e. $w(L_i)$ and $w(R_i)$. In Step 4, If $w(L_i) = w(R_i)$, then i is the vertex one center as well as Inv1C of T_{CIR} (Step 4.1). But if $w(L_i) \neq w(R_i)$, then modify the tree T_{CIR} under the conditions of non-linear semi-infinite (or nonlinear) optimization model (Step 4.2). By Step 4.3, modify the tree T_{CIR} we get the weights $w^*(L_i)$ and $w^*(R_i)$ of both sides of i and we get $w^*(L_i) = w^*(R_i)$. Therefore i is the Inv1C. Hence **Algorithm 1-INV-CIR-ARC-TREE** correctly computes the Inv1C for any weighted tree. □

We have another important observation in the tree T'_{CIR} given by the **Algorithm 1-INV-CIR-ARC-TREE**.

Lemma 4.4.10 *The specified vertex i in the modified tree T'_{CIR} is the Inv1C.*

Proof. By **Algorithm 1-INV-CIR-ARC-TREE**, finally we get $w^*(L_i) = w^*(R_i)$ in the modified tree T'_{CIR}. Therefore the specified vertex i in modified tree T'_{CIR} is the Inv1C. □

The following theorem describe the total T-complexity of the algorithm to compute Inv1c problem on weighted tree corresponding to the weighted CirG G.

Theorem 4.4.2 *The T-complexity to find Inv1C problem on a given weighted tree T'_{CIR} corresponding to the weighted CirG G is $O(n)$, where n is number of vertices of the graph.*

Proof. $O(n)$ time is needed to compute Step 1 , since the adjacency relation of CirG can be tested in $O(1)$ time. Step 2, i.e. longest weighted path from i to v_i can be computed in $O(n)$ time if T_{CIR} is traversed in a Depth-First-Search(DFS) manner. Step 3 takes $O(n)$ time to compute the sum of the weights of the paths. Also, Step 4.1 takes $O(1)$ time. Computation of k_1 and k_2, i.e. number of vertices in R_i and L_i needs $O(n)$ time, so each Step 4.2 takes $O(n)$ time (since comparison of two numbers and distribution of the excess weight takes $O(n)$ time, so, each Step 4.2.1 to 4.2.6 can compute in $O(n)$ time). The modification of weights in either R_i or L_i just takes $O(n)$ time as T_{CIR} contains $(n-1)$ edges and n vertices, so, Step 4.3 can be executed in $O(n)$ time. Hence, the overall T-complexity of our proposed **Algorithm 1-INV-CIR-ARC-TREE** is $O(n)$ time, where n is number of vertices of the CirG. □

4.5 Summary

In this chapter, we proposed an efficient algorithm for computing a MADT on CirGs which is designed based on BFS technique. The T-complexity of this algorithm is of $O(n^2)$, where n is number of vertices of the CirG. According to our knowledge, the complexity is not optimal, so one can try to improve this algorithm as extensive research work for optimal algorithm. Also, we investigated the Inv1c location problem with weights of the vertex on the tree corresponding to the weighted CirG G. We developed exact combinatorial solution algorithm for the *tree* with fast running time $O(n)$, where n is number of vertices of the weighted CirG.