Chapter 3

Computation of inverse 1-center location problem and minimum average distance tree on interval graphs *

3.1 Introduction

Suppose $G = (V, E)$ be an UndG. Now, G is known to be a InvG, if we create each node of V for each interval in the set of intervals I defined on real line so that $(x, y) \in E$ iff the intervals associated to the nodes x and y have common intersection. We use the symbol I to represent an IR of G and G to represent an $IntG$ of I [50]. Let $I = \{j_1, j_2, \ldots, j_n\}$, where $j_z = [a_z, b_z]$ for $1 \leq z \leq n$, be the IR of the graph G, where a_z represents the left extremity and b_z is the right extremity of the interval i_z. Without loosing the generality, we presume the following notations [50]:

(i) each interval has both its extremities and two or more intervals do not have a same extremely,

(ii) intervals in IR and nodes in InvG are same,

(iii) our assumed InvG is connected & the array of arranged extremeties/endpoints is provided and

*A part of the work presented in this chapter is published in *Int. J. Computing Science and Mathematics*, 8 (2017) 533-541.

(iv) the intervals in I are numbered according to the ascending/growing right extremities.

If two or more intervals contain same extremities we apply the Algorithm CONVERT [82] to convert the given intervals into the intervals with different extremities.

Here we take the weighted InvG, i.e. for each interval j, we assign a positive weight $w_j > 0$.

An InvG and its IR are shown in Figure 3.1 and Figure 3.2 respectively.

Many researchers studied InvG and they used this graph as the mathematical model to solve several real life problems. A brief discussion about InvG was found in [50]. To the best of our knowledge InvG and its different subclass have lots of applications in archeology, protein sequencing [56], works scheduling [18], macro substitution [39], VLSI design, file arrangements [18], psychology, transportation, routing between two points nets [52], circuit routine [57, 79], genetics, molecular biology, sociology, circuit routing etc.

For a graph G, a *walk* can be defined as a finite alternating series of vertices and edges which is beginning and ending with vertices such that each edge is incident with those vertices which are preceding and following it. In a walk,there will be no edge which appears more than once. However, a vertex can appear more than once. A *path* can be defined as an open walk in which there will be no vertex appearing more than once. A *circuit* can be defined as a closed walk in which there will be no vertex (except the initial vertex and the final vertex) appearing more than once. A *tree* T is a graph which is connected, containing no circuits. i.e. a *tree* T is a connected acyclic graph. Clearly there will be one and only one path between each and every pair of vertices of tree T. For an w-tree, there will be a non-negative real number attached with each edge of the tree. For an un-w-tree $T(V_1, E_1)$, where $|E_1| = |V_1| - 1$, the *eccentricity e(x)* of a vertex x can be defined as the distance from x to the vertex which is farthest from $x \in T$, i.e.

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

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

For a w-tree $T(V, E)$, the *eccentricity e(x)* of the vertex x can be defined as the sum of the weights of the edges from x to the vertex which is farthest from $x \in T$, i.e.

$$e(x) = \max\{d_w(x, x_i), \text{ for all } x_i \in T\},$$

where the sum of the weights of the edges on the path between x and x_i is $d_w(x, x_i)$.

A *center* of a tree T can be defined by a vertex with minimum eccentricity i.e. if $e(s) = \min\{e(x), \text{ for all } x \in V\}$, then s is called *1-center*. We know very well that every tree is either monocentric or bicentric.

In a tree T, the eccentricity $e(x)$ of a center can be defined as the *radius* of the tree T which is denoted by $\rho(T)$, i.e.

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

For a tree T , the *diameter* can be defined as the length of the longest path that implies, the diameter is the maximum eccentricity.

For the w-tree T with n vertices and $n-1$ edges of the corresponding weighted InvG, the Inv1C problem on weighted tree T is concerned with modifying parameter, like vertex weight, at minimum total cost within certain modification bounds such that a pre-specified vertex becomes the 1-center. For example, consider the train station of a city which can not be relocated. The mayor could change some parameters in the urban system (e.g., improving streets or urban transportation lines) at minimum cost subject to evident length constraints such that the current location of the train station becomes the center (in a graph theoretic sense) of the city.

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

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

The MADST of the *fuzzy InvG* G is a spanning tree of G with minimum average distance.

In this chapter, we propose an algorithm to compute

(i) Inv1C location problem on weighted InvG in $O(n)$ time, where n is the number of vertices of the weighted InvG, and

(ii) the minimum average distance tree on the fuzzy InvG with $O(n^2)$ time, where n is the number of vertices of the fuzzy InvG.

3.2 Organization of the chapter

In the next section, i.e. Section 3.3, we discuss the Inv1C. In Subsection 3.3.1 present the data structure and construction of the tree T_{IG}. Some notations have also presented in this subsection. In Subsection 3.3.2, we present an algorithm to get Inv1C of the modified vertex weighted interval tree corresponding to the InvG G. The T-complexity is also calculated in this subsection. In Section 3.4, we discuss about MADT on fuzzy InvG and Subsection 3.4.1 presents the definition, basic operations on interval number. In Subsection 3.4.2, we present the data structure and construction of the tree. Also this subsection presents the spanning tree and MdsT and in Subsection 3.4.3, we develop modified spanning tree of the

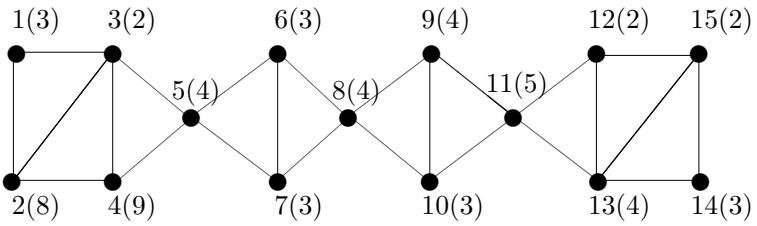

Figure 3.1: Interval graph G.

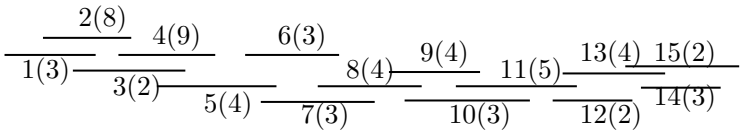

Figure 3.2: Interval matching diagram of the InvG G of Figure 3.1.

tree T. Subsection 3.4.4 presents the average distance of the fuzzy InvG and Subsection 3.4.5 presents an algorithm to get MADT of the InvG. The T-complexity is also calculated in this subsection. Section 3.5 presents the summary.

3.3 Inverse 1-center location problem on interval graphs

3.3.1 Data structure and construction of the tree

Let $G = (V, E)$, $V = \{1, 2, \ldots, n\}, |V| = n, |E| = m$ be a connected InvG where the nodes are given in the sorted order of the right extremity of the interval representation of the graph. Intervals are labeled according to increasing order of their extremities. This labeling is referred to as IG ordering. Let (x, y) or (y, x) denote the existence of an adjacency relation between two vertices x, y. It is assumed that (x, x) is always true, i.e. $(x, x) \in E$. If $[a_x, b_x]$ and $[a_y, b_y]$ are two extremities of the nodes x and y respectively then x, y are adjacent if at least one of the following conditions hold:

(i) $a_y < a_x < b_y$,

(ii) $a_y < b_x < b_y$,

(iii) $a_x < a_y < b_x$,

(iv) $a_x < b_y < b_x$.

So, instead of storing an InvG using adjacency matrix or adjacency list, one can store the interval representation of the InvG using only $2n$ units. The adjacency relation can be tested in $O(1)$ time. This is a major advantage of InvG.

For each vertex $v \in V$, let $H(v)$ be the highest number vertex adjacent to v. If there is no vertex adjacent to v or if the adjacent vertex is not greater than v, then $H(v) = v$, i.e.

$H(v) = max\{u : (u, v) \in E, u \geq v\}$.

We define, $N(i)$ = the open neighbourhood of $i = \{v : (v, i) \in E\}$,

$$k = max\{b_k : (k, i) \in E\},$$
$$j' = min\{a_{j'} : (j', i) \in E\},$$
$$j = max\{b_j : (j, i) \in E, j \neq k\},$$
$$k' = min\{a_{k'} : (k', i) \in E, k' \neq j'\}.$$

Let i be pre-specified vertex which to be Inv1C. Our target is to form a spanning tree corresponding to the weighted InvG with two branches. Let the vertex i be the root of the tree. Then we find all adjacent vertices to i and set them as child (leaves) of i. To form the spanning trees we have the following two cases:

Cases I: If number of adjacent of i is one, i.e. $deg(i) = 1$, then we can not construct a tree with root i and two branches. Therefore, vertex i is not Inv1C of the weighted IT.

Case II: If number of adjacent vertices to the vertex i are more than one, i.e. $deg(i) > 1$, then three possibilities arises and we try to form a tree with two longest branches.

(a) When i is the starting vertex in G, i.e. $i = 1$:

In this case we find all adjacent vertices to the vertex 1 and set them as child (leaves) of 1 and marked them. Next we consider the vertices k and j whose right end points of the corresponding intervals are maximum and next maximum respectively. Next find all unmarked adjacent vertices to the vertices k and j respectively. If there is no common adjacent vertices to k and j, then find m_1, interval whose right end point is maximum among all adjacent to k and all unmarked adjacent are placed as the child of k and marked them else m_1' as child of j and marked, where $m_1' = max\{b_{m_1'} : m_1' \in N(k) \cap N(j)\}$ and all members of $\{N(k) \cup N(j) - \{m_1'\}\}$ as child of k and marked and find $m_1'' = max\{N(k) \cup N(j) - \{m_1'\}\}$. This process is continued until all intervals right to 1 are marked.

(b) When i is the end vertex in G, i.e. $i = n$:

In this case we find all adjacent vertices to the vertex n and set them as child (leaves) of n and marked them. Next we consider the vertices j' and k' whose left end points of the corresponding intervals are minimum and next minimum respectively. Next find all unmarked adjacent vertices to the vertices j' and k' respectively. If there is no common adjacent vertices to j' and k', then find m_1, interval whose left end point is minimum among all adjacent to j' and unmarked adjacent are placed as the child of j' and marked them else m_1' as child of k', where $m_1' = min\{a_{m_1'} : m_1' \in N(k') \cap N(j')\}$ and all members of $\{N(k') \cup N(j') - \{m_1'\}\}$ as child of j' and marked and find $m_1'' = min\{a_{m_1''} : m_1'' \in \{N(k') \cup N(j') - \{m_1'\}\}\}$. This process is continued until all intervals left to n are marked.

(c) When i is the vertex between 1 and n, i.e. $1 < i < n$:

In this case we find all adjacent vertices to the vertex i and set them as child (leaves) of i and marked them. Next, we consider the vertex k whose right end point of the corresponding interval among all adjacent vertices to i is maximum. Corresponding to the vertex k we find all unmarked vertices adjacent to k and put them as the child of k. Continuing this process on the right side of the interval diagram until all vertices corresponding to the intervals right of i are marked. Similarly, on the left side of i, we find j', the unmarked adjacent to i, and put them as child in left branch. Then same procedure is applied on left side until all intervals left of i are marked.

Now we propose a combinatorial algorithm to construct the tree T_{IG}. Our proposed algorithm is given below:

Algorithm INT-TREE

Input: Weighted InvG G having interval representation $I = [i_1, i_2, \ldots, i_n]$, $i_j = [a_j, b_j]$ and weight w_j, $j = 1, 2, \ldots, n$.

Output: The rooted tree T_{IG} with two branches of the InvG G.

Step 1. Set root $= i$ and compute $N(i) =$ the open neighbourhood of
$i = \{v : (v, i) \in E\}$.

Step 2. If $|N(i)| = 1$, then end.

If $|N(i)| > 1$ and i is starting interval, i.e. $i = 1$, then goto Step 3.

If $|N(i)| > 1$ and i is the end interval, i.e. $i = n$, then goto Step 4.

If $|N(i)| > 1$ and i is an interval between 1 and n, i.e. $1 < i < n$, then goto Step 5.

Step 3. Set $N(i)$ as the child of the root i and marked them.

Step 3.1. Set $k = max\{b_k : (k, i) \in E\}$,
$j = max\{b_j : (j, i) \in E, k \neq j \ \ and \ \ b_j < b_k\}$.

Step 3.2. Find unmarked adjacent of j and k and if $N(j) \bigcap N(k) = \phi$,
then $m_1 = max\{b_{m_1} : (m_1, k) \in E, m_1 \in N(k)\}$ and set all
unmarked $N(k)$ as the child of k and marked them.

else $m_1' = max\{b_{m_1'} : m_1' \in N(k) \cap N(j)\}$ set as child of j and
$\{N(k) \cup N(j) - \{m_1'\}\}$ as child of k and marked and find
$m_1'' = max\{N(k) \cup N(j) - \{m_1'\}\}$.

Step 3.3. This procedure is running till all intervals are traced.

Step 3.4. Compute the interval tree T_{IG}.

Step 4. Set $N(i)$ as the child of the root i and marked them.

Step 4.1. Set $j' = min\{a_{j'} : (j', i) \in E\}$,

$k' = min\{a_{k'} : (k', i) \in E, k' \neq j' \ and \ a'_j < a'_k\}$.

Step 4.2. Find unmarked adjacent of j' and k' and if

$N(j') \bigcap N(k') = \phi$, then $m_1 = min\{a_{m_1} : (m_1, j') \in E, m_1 \in N(j')\}$

and set all unmarked $N(j')$ as the child of j' and marked them.

else $m'_1 = min\{a_{m'_1} : m'_1 \in N(k') \cap N(j')\}$ set as child of k'

and $\{N(k') \cup N(j') - \{m'_1\}\}$ as child of j' and marked and find

$m''_1 = min\{a_{m''_1} : m''_1 \in \{N(k') \cup N(j') - \{m'_1\}\}\}$.

Step 4.3. This procedure is running till all intervals are traced.

Step 4.4. Compute the interval tree T_{IG}.

Step 5. Set $N(i)$ as the child of the root i and marked them.

Step 5.1. Set $p = max\{b_p : (p, i) \in E\}$, $q = min\{a_q : (q, i) \in E\}$

and $p \neq q$.

Step 5.2. Set $p' = max\{b_{p'} : (p', p) \in E, p' \in N(p)\}$ and set all

unmarked $N(p)$ as the child of p and marked.

Step 5.3. Set $q' = min\{a_{q'} : (q', q) \in E, q' \in N(q)\}$ and set all

unmarked $N(q)$ as the child of q and marked.

Step 5.4. This procedure is running till all intervals are traced.

Step 5.5. Compute the interval tree T_{IG}.

Step 6. Put weight w_j to the vertex j in T_{IG} corresponding to the interval j of the InvG G.

end INT-TREE.

Illustration of the Algorithm INT-TREE : Let $i = 1$ be the pre-specified vertex which is the root. Next the open neighbourhood of 1 is $N(1) = \{2, 3\}$, where the vertices of $N(1)$ as the child of the root 1. Next, 3 has the farthest right extremity b_j of the intervals of $N(1)$ corresponding to the vertices of the graph G and 2 has the next maximum right extremity of $N(1)$ corresponding to the nodes in the graph G. Next the open neighbourhoods of 3 and 2 are $N(3) = \{4, 5\}$ and $N(2) = \{4\}$ respectively, where the vertices of $N(3)$ and $N(2)$ as the child of the roots 3 and 2. Next 5 has the furthest right extremity of the intervals of $N(3)$ corresponding to the nodes of the graph G and 4 contains the least tail among the intervals of $N(2)$ corresponding to the vertices of the graph G. Next the open neighbourhood of 5 is $N(5) = \{6, 7\}$, where the vertices of $N(5)$ as the child of the root 5. Next 7 has the farthest right extremity of the intervals of $N(5)$ corresponding to the nodes in the graph G. Next the open neighbourhood of 7 is $N(7) = \{8\}$, where the vertex of $N(7)$ as the child of the root 7. Next the open neighbourhood of 8 is $N(8) = \{9, 10\}$, where the vertices of

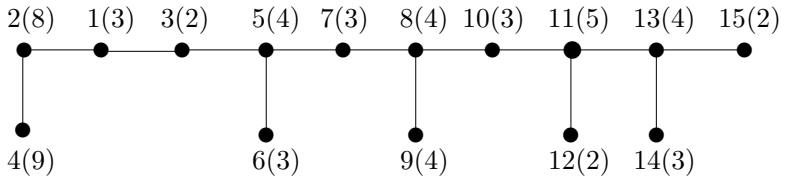

Figure 3.3: Tree T_{IG} of the InvG G with longest branch on both sides by checking adjacency.

$N(8)$ as the child of the root 8. Next 10 has the furthest right extremity of the intervals of $N(8)$ corresponding to the nodes of the graph G. Next the open neighbourhood of 10 is $N(10) = \{11\}$, where the vertex of $N(10)$ as the child of the root 10. Next 11 has the maximum right end point among the intervals of $N(10)$ corresponding to the vertices of the graph G. Next the open neighbourhood of 11 is $N(11) = \{12, 13\}$, where the vertices of $N(11)$ as the child of the root 11. Next 13 has the furthest right extremity of the intervals of $N(11)$ corresponding to the nodes of the graph G. Next the open neighbourhood of 13 is $N(13) = \{14, 15\}$, where the vertices of $N(13)$ as the child of the root 13. Next 15 has the furthest right extremity of the intervals of $N(13)$ corresponding to the nodes in the graph G. In this way we get longest left path L_i from the vertex 1 to other vertex 4, i.e. the path $1 \to 2 \to 4$ and find longest right path R_i from 1 to the vertex 15 does not contain any vertex of the path L_i except 1, i.e. the path $1 \to 3 \to 5 \to 7 \to 8 \to 10 \to 11 \to 13 \to 15$. Next put the weights $9, 8, 3, 2, 4, 3, 4, 3, 5, 4, 2, 3, 4, 2, 3$ to the vertices $4, 2, 1, 3, 5, 7, 8, 10, 11, 13, 15, 6, 9, 12, 14$ respectively. Finally we construct the rooted tree T_{IG} with root $i = 1$ (Figure 3.3).

We have the following important observation on T_{IG}.

Lemma 3.3.1 *The tree T_{IG} constructed by the Algorithm **INT-TREE** is a spanning tree.*

Proof. As per construction of the tree T_{IG} by end points scanning approach of the intervals we get n vertices and $n - 1$ edges without any circuit. Therefore the tree T_{IG} is a spanning tree.

Hence the result. □

Next we discuss about the execution time of the Algorithm **INT-TREE**.

Theorem 3.3.1 *The execution time of the Algorithm **INT-TREE** is $O(n)$, where n is the cardinality of the vertex set.*

Proof. Step 1 can be finished in $O(n)$ time. Step 2 is completed in $O(n)$ time, since number of intervals is n. Again intersection of two finite sets of n elements (number of vertices) can

be executed in $O(n)$ time. Thus Step 3, or Step 4, or Step 5 can be finished in $O(n)$ time. Finally the execution time of the Algorithm **INT-TREE** is $O(n)$, where n is the cardinality of the vertex set. Since weight of the vertex in tree T_{IG} corresponds the weight of the interval in InvG, so it is one - to - one corresponds, and hence Step 6 can be completed in $O(n)$ time. □

Thus the tree T_{IG} of the InvG is formed. The tree T_{IG} with root as 1 of the InvG G is displayed in Figure 3.3. Next assign the weight.

Now, we introduce some notations for our algorithmic purpose.

R_i	:	Longest weighted path right to the vertex i.
L_i	:	Longest weighted path left to the vertex i does not contain any vertex of the path R_i.
$w(R_i)$	:	total weight of the nodes in the path R_i.
$w(L_i)$	:	total weight of the nodes in the path L_i.
$w^*(R_i)$	:	total weight of the nodes in the path R_i after modification.
$w^*(L_i)$	:	total weight of the nodes in the path L_i after modification.
w_{low}	:	$min\{w(L_i), w(R_i)\}$.
w_{high}	:	$max\{w(L_i), w(R_i)\}$.
k	:	number of vertices in such path between L_i, R_i whose weight is maximum, except the vertex i.
$w_{low}(v)$	:	$min\{w(v) : v \in T_{IG}\} = w'$.
$w_{upp}(v)$	:	$max\{w(v) : v \in T_{IG}\}$.
T_{IG}	:	weighted tree associated to the InvG G.
T'_{IG}	:	Modified tree of T_{IG}.

To find Inv1C we discuss following two cases:

1. If $w(L_i) = w(R_i)$, 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 three cases :

Case-2.1. : When $w_{low} = kw'$, where w' is the minimum weight of the vertex in the graph.

Case-2.2. : When $w_{low} > kw'$, where w' is the minimum weight of the vertex in the graph.

Case-2.3. : When $w_{low} < kw'$, where w' is the minimum weight of the vertex in the graph.

Under above conditions we modify the tree T_{IG} with the help of following non-linear optimization model:

$$\text{Min} \sum_{v_1 \in V_1(T_{IG})} \{c_1^+(w(v_1))x_1(w(v_1)) + c_1^-(w(v_1))y_1(w(v_1))\}$$

subject to

$$\max_{v_1 \in V_1(T_{IG})} d_{\overline{w}}(v_1, i) \leq \max_{v_1 \in V_1(T_{IG})} d_{\overline{w}}(v_1, p), \forall p \in V_1(T_{IG}),$$
$$\overline{w}(v_1) = w(v_1) + x_1\{w(v_1)\} - y_1\{w(v_1)\} \text{ for all } v_1 \in V_1(T_{IG}),$$
$$x_1\{w(v_1)\} \leq w^+\{w(v_1)\}, \forall v_1 \in V_1(T_{IG}),$$
$$y_1\{w(v_1)\} \leq w^-\{w(v_1)\}, \forall v_1 \in V_1(T_{IG}),$$
$$x_1\{w(v_1)\}, y_1\{w(v_1)\} \geq 0, \forall v_1 \in V_1(T_{IG}),$$

where $\overline{w}(v_1)$ be the modified vertex weight, $w^+\{w(v_1)\} = w_{upp}(v_1) - w(v_1)$, $w^-\{w(v_1)\} = w(v_1) - w_{low}(v_1)$ be the highest feasible measurements in which $w(v_1)$ can be increased and reduced respectively, i.e. $w_{low}(v_1) \leq \overline{w}(v_1) \leq w_{upp}(v_1)$, $x_1\{w(v_1)\}$ and $y_1\{w(v_1)\}$ are the measurements by which the vertex weight $w(v_1)$ is increased and reduced respectively, $c_1^+(w(v_1))$ is the non negative cost if $w(v_1)$ is grown by one unit and $c_1^-(w(v_1))$ is the non negative cost if $w(v_1)$ is reduced by one unit. Each feasible solution (x_1, y_1) with $x_1 = \{x_1(w(v_1)) : v_1 \in V_1(T_{IG})\}$ and $y_1 = \{y_1(w(v_1)) : v_1 \in V_1(T_{IG})\}$ is known as feasible modification of Inv1C location problem.

Next, we prove some results.

Lemma 3.3.2 *If $w_{low} = kw'$ in T_{IG}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all nodes except the node i, i.e. root i up to least weight maintaining the bounding condition in the path whose weight is maximum and i is the Inv1C.*

Proof. Since k is the number of nodes in the greatest weighted path L_i or R_i and w' is the least weight of the node among the nodes in T_{IG} as well as L_i or R_i, then there is a scope to reduce weight of each vertex up to w'. As k vertices is there in the path L_i or R_i, so we can reduces at least kw' weight and hence reduced weight of the path L_i or R_i becomes kw'. Again we have $w_{low} = kw'$. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{IG}, say T'_{IG}. Again, since the InvG is an arbitrary, so our assumption is true for any InvG.

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

Lemma 3.3.3 *If $w_{low} > kw'$ in T_{IG}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of some nodes 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 node except the root up to minimum weight of the node in T_{IG}, 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 kw'. Again we have $w_{low} > kw'$. Therefore we can decrease the weights $(w_{high} - w_{low})$ from the vertices except the root i in the path whose weight is maximum using the conditions of non-linear semi-infinite optimization model technique (Subsection 3.3.1). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{IG}, say T'_{IG}. Again, since the InvG is an arbitrary, so our assumption is true for any InvG.

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

Lemma 3.3.4 *If $w_{low} < kw'$ in T_{IG}, 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 vertex up to minimum weight of the vertex in T_{IG}, 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 kw'. Again we have $w_{low} < kw'$. Therefore we can increase the weights $(kw' - w_{low})$ to the vertices except the root in the path whose weight is minimum using the conditions of non-linear semi-infinite optimization model technique (Subsection 3.3.1). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{IG}, say T'_{IG}. Again, since the InvG is an arbitrary, so our assumption is true for any InvG.

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

3.3.2 Algorithm and its complexity

Suppose T_{IG} is the weighted tree of the InvG G with n nodes and $(n-1)$ edges. Let V is the node set and E be the edge set. Let i is any non-pendant specified vertex in the tree T_{IG} which is to be Inv1C. At first we calculate the maximum weighted path from i to any pendant vertex of T_{IG}. Let L and R be the left and right paths from i in which weights are maximum with respect to sides. Let $w(L_i)$, $w(R_i)$ be the total weights of the nodes of the paths L_i, R_i respectively with respect to the interval 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 three cases may arise. In the first case, if $w_{low} = kw'$ in T_{IG}, where $w' = min\{w(v), v \in G\}$, $w_{low} = min\{w(L_i), w(R_i)\}$ and $w_{low} > 0$, 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 conditions (Subsection 3.3.1) in the path whose weight is maximum and i is the Inv1C. In the second case, if $w_{low} > kw'$ in T_{IG}, where $w = min\{w(v), v \in G\}$, $w_{low} = min\{w(L_i), w(R_i)\}$ and $w_{low} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of some vertices except the root i maintaining the bounding conditions (Subsection 3.3.1) in the path whose weight is maximum and i is the Inv1C. In third case, if $w_{low} < kw'$ in T_{IG}, where $w = min\{w(v), v \in G\}$, $w_{low} = min\{w(L_i), w(R_i)\}$ and $w_{low} > 0$, 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 conditions (Subsection 3.3.1) 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.

Our proposed algorithm to the Inv1C location problem of the interval tree corresponding to the InvG G is as follows:

Algorithm 1-INV-INT-LOC-TREE

Input: Weighted InvG G with interval representation $I = [i_1, i_2, \ldots, i_n]$, $i_j = [a_j, b_j]$, $j = 1, 2, \ldots, n$.

Output: Vertex i as Inv1C of the tree T_{IG} and modified tree T'_{IG}.

Step 1. Construction of the tree T_{IG} (as per Subsection 3.3.1) with root i.

Step 2. Compute the weighted path R_i from i to other vertex v_j
on the tree T_{IG}.

Step 3. Next compute weighted path L_i from i to the vertex v_k
does not contain any vertex of the path R except i.

Step 4. Calculate weights of two paths L_i and R_i, i.e. $w(L_i)$ and $w(R_i)$.

Step 5. //Modification of the tee T_{IG}//

Step 5.1. If $w(L_i) = w(R_i)$, then i is the vertex one center as well as inverse 1-center of T_{IG}.

Step 5.2. If $w(L_i) \neq w(R_i)$, then

Step 5.2.1. If $w_{low} = kw'$ in T_{IG}, 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 5.3.

Step 5.2.2. If $w_{low} > kw'$ in T_{IG}, 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 5.3.

Step 5.2.3. If $w_{low} < kw'$ in T_{IG}, 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 5.3.

Step 5.3. T'_{IG} = modified tree of the tree T_{IG}.

end 1-INV-INT-LOC-TREE.

Using above Algorithm **1-INV-INT-LOC-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 to the tree T_{IG} in Figure 3.3:

Let $i = 1$ be the pre-specified vertex of the tree T_{IG} which is to be Inv1C. Next we find the longest left path L_i from the vertex 1 to other vertex 4, i.e. the path $1 \to 2 \to 4$ and find longest right path R_i from 1 to the vertex 15 does not contain any vertex of the path L_i except 1, i.e. the path $1 \to 3 \to 5 \to 7 \to 8 \to 10 \to 11 \to 13 \to 15$. Next calculate the weights of the paths L_i and R_i. Let $w(L_i)$ and $w(R_i)$ be the total weight of the nodes of the paths L_i and R_i respectively. Here $w(L_i) = 20$ and $w(R_i) = 30$. Next calculate the difference of weights of two paths, i.e. calculate $w(R_i) - w(L_i)$. Therefore $w(R_i) - w(L_i) = 30 - 20 = 10$. To get equal weights of $w(L_i)$ and $w(R_i)$ we subtract the weight 2 to the vertex 5, 1 to the vertex 7, 2 to the vertex 8, 1 to the vertex 10, 3 to the vertex 11, 1 to the vertex 13 (using Lemma 3.3.3). After modification we get $w^*(L_i) = \{3 + 8 + 9\} = 20$ and $w^*(R_i) = \{3 + 2 + (4 - 2) + (3 - 1) + (4 - 2) + (3 - 1) + (5 - 3) + (4 - 1) + 2\} = 20$, i.e. $w^*(L_i) = w^*(R_i)$. Therefore the vertex 1 is the Inv1C.

Now we have the modified tree T'_{IG} (Figure 3.4) with modified vertex weight.

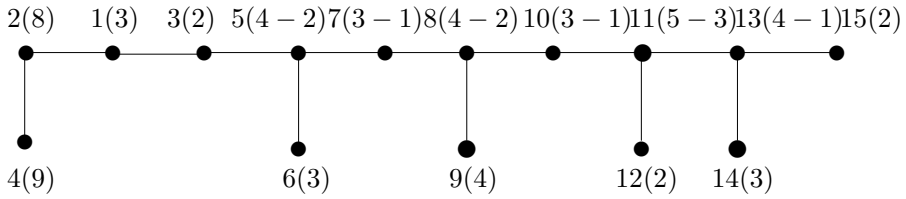

Figure 3.4: Modified tree T'_{IG} of the tree T_{IG}.

Next we prove the important result.

Lemma 3.3.5 *The Algorithm **1-INV-INT-LOC-TREE** correctly computes the Inv1C lo-*

cation on the weighted interval tree.

Proof. Let i be the pre-specified vertex in T_{IG}. We have to prove that i is the Inv1C. At first, by Step 1, we have constructed the tree T_{IG} (as per Subsection 3.3.1) with root i, by Step 2, calculate the weight of the weighted path R_i from i to other vertex v_j on the tree T_{IG}, by Step 3, calculate the weight of the another weighted path L_i from i to the vertex v_k does not contain any vertex of the path R_i except i. In Step 4, Calculate weights of two paths L_i and R_i, i.e. $w(L_i)$ and $w(R_i)$. If $w(L_i) = w(R_i)$, then i is the vertex one center as well as Inv1C of T_{IG} (Step 5.1). But if $w(L_i) \neq w(R_i)$, then modify the tree T_{IG} with the help of non-linear semi-infinite (or nonlinear) optimization model (Step 5.2). By Step 5.3, modify the interval tree T_{IG} 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-INT-LOC-TREE** correctly computes the Inv1C for any vertex w-tree. □

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

Lemma 3.3.6 *The specified vertex i in the modified tree T'_{IG} is the Inv1C.*

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

The following describe the total T-complexity of the algorithm to compute Inv1C problem on the weighted IT.

Theorem 3.3.2 *The T-complexity to find Inv1C problem on a given vertex weighted interval tree T_{IG} is $O(n)$, where n is the number of vertices of the tree.*

Proof. Step 1 takes $O(n)$ time, since the adjacency relation of InvG can be tested in $O(n)$ time. Step 2, i.e. longest weighted path from i to v_i can be computed in $O(n)$ time if T is traversed in a depth-first-search manner. Similarly, Step 3 can be computed in $O(n)$ time. Step 4 takes $O(n)$ time to compute the sum of the weights of the paths. Also, Step 5 takes $O(n)$ time. Since comparing two numbers and distribution of the excess weight takes $O(n)$ time, so, Step 5.1, Step 5.2 and Step 5.3 can be computed $O(n)$ time. Hence overall T-complexity of our proposed Algorithm **1-INV-INT-LOC-TREE** is $O(n)$ time, where n is the number of vertices of the InvG. □

In next section, we solve another interesting problem on fuzzy interval graphs which has a lot of applications in our real life world.

3.4 MADT on fuzzy interval graph

In this section, we consider the weight as interval number of the vertices of the fuzzy InvG corresponding to the interval representation. Here, uncertainty arises due to taking weight as interval number instead of taking single real number weight. So, fuzzy sense occurs in the weight.

Here we consider the fuzzy InvG which is defined in [25, 89]. Corresponding to each interval i, we put a weight $w_i = [a_i, b_i]$ as interval number.

A fuzzy InvG and its interval diagram are shown in Figure 3.5 and Figure 3.6 respectively.

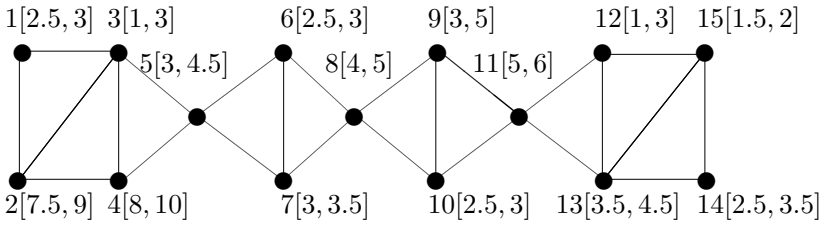

Figure 3.5: An fuzzy interval graph G.

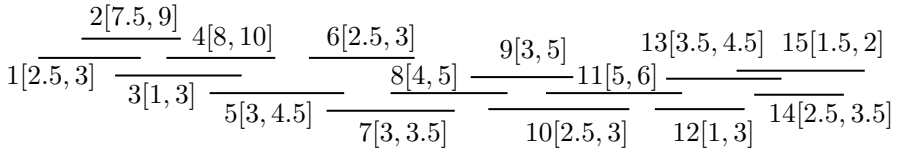

Figure 3.6: Interval matching diagram of the graph G of Figure 3.5.

3.4.1 Interval numbers and their arithmetic

A closed interval $B = [x, y] = b \in R : x \leq b \leq y$ is called an InvNum. The numbers x and y of B are the lower and upper limit respectively .

The number $[b, b]$ is called degenerated InvNum for any real number b. Alternatively B is represented as $B =< me(B), wd(B) >$, where $me(B)$ and $wd(B)$ are the mid-point and half-width of B, i.e.,

$me(B) = (x + y)/2$ and $wd(B) = (y - x)/2$

Next sub section we discuss about InvNum.

Some basic operations on interval number

The arithmetic operations are defined as follows:

(i) $[p_1, q_1] \oplus [p_2, q_2] = [p_1 + p_2, q_1 + q_2]$,

(ii) $[p_1, q_1] \ominus [p_2, q_2] = [p_1 - q_2, q_1 - p_2]$,

(iii) $[p_1, q_1] \odot [p_2, q_2] = [min(p_1p_2, p_1q_2, q_1p_2, q_1q_2), max(p_1p_2, p_1q_2, q_1p_2, q_1q_2)]$,
(iv) $\lambda[p, q] = [\lambda p, \lambda q]$ and $[\lambda q, \lambda p]$ if $\lambda \geq 0$ and $\lambda < 0$ respectively.

Comparison between interval numbers

It is very important to assimilate between two InvNum in interval arithmetic. Several authors [19, 20, 35, 36, 93, 94] are proposed different order relations between two intervals.

Let $P = [x_l, x_r]$ and $Q = [y_l, y_r]$ be two intervals. For $me(P) \geq me(Q)$ and $P > Q$, the AcpI is defined by $A(P > Q) = (me(P) - me(Q))/(wd(Q) + wd(P))$, then P is greater than Q.

The values of the AcpI for different position of means and for different values of widths of the intervals are as follows:
(i) $A(P > Q) \geq 1$ when $me(P) > me(Q)$ and $x_l \geq y_r$,
(ii) $0 < A(P > Q) < 1$ when $me(P) > me(Q)$ and $x_l < y_r$,
(iii) $A(P > Q) = 0$ when $me(P) = me(Q)$.

In the first case, P is greater than Q, where AcpI greater than or equal to 1. In the second case, P is preferred over Q with different grades of satisfaction lying between 0 and 1. In the third case, if $wd(P) = wd(Q)$, then P and Q are the same interval. If $wd(P) \neq wd(Q)$, then the intervals P and Q are not superior to each other, i.e. AcpI becomes insignificant.

3.4.2 Data structure and construction of the tree

Suppose $G = (V, E)$, $|V| = n, |E| = m$ be a connected InvG, where the nodes are given in the sorted order of the right extremity of the interval representation of the graph. According to increasing order of their extremities intervals are labeled. This labeling is referred to as IG ordering. Let (x, y) or (y, x) denote the existence of an adjacency relation among two nodes x, y. It is assumed that (x, x) is always true, i.e. $(x, x) \in E$. If $[a_x, b_x]$ and $[a_y, b_y]$ are two end points of the nodes x and y respectively then x, y are adjacent if at least one of the following restrictions hold:

(i) $a_y < a_x < b_y$,
(ii) $a_y < b_x < b_y$,
(iii) $a_x < a_y < b_x$,
(iv) $a_x < b_y < b_x$.

So, instead of storing an InvG using adjacency matrix or adjacency list, one can store the IR of the InvG using only $2n$ units. The adjacency relation can be tested in $O(1)$ time. This is a major advantage of InvG.

For every node $x \in V$, let $H(x)$ be the greatest number node adjacent to x. If there is no node adjacent to x or if the adjacent node is not greater than x, then $H(x) = x$, i.e. $H(x) = max\{y : (y, x) \in E, y \geq x\}$.

We define, $N(i)$ = open neighbourhood of $i = \{x : (x, i) \in E\}$,

$$k = max\{b_k : (k, i) \in E\},$$
$$j' = min\{a_{j'} : (j', i) \in E\},$$
$$j = max\{b_j : (j, i) \in E, j \neq k\},$$
$$k' = min\{a_{k'} : (k', i) \in E, k' \neq j'\}.$$

Our target is to form a spanning tree corresponding to the fuzzy InvG with two branches. Let the vertex i be the root of the tree. Then we find all adjacent vertices to i and set them as child (leaves) of i. To form the spanning trees we have the following two cases:

Cases I: If number of adjacent of i is one, i.e. $deg(i) = 1$, then we can not construct a tree with root i and two branches.

Case II: If number of adjacent vertices to the vertex i are more than one, i.e. $deg(i) > 1$, then three possibilities arises and we try to form a tree with two longest branches.

(a) When i is the starting node in G, i.e. $i = 1$:

In this case we find all adjacent vertices to the vertex 1 and set them as child (leaves) of 1 and marked them. Next we consider the vertices k and j whose right end points of the corresponding intervals are maximum and next maximum respectively. Next find all unmarked adjacent vertices to the vertices k and j respectively. If there is no common adjacent vertices to k and j, then find m_1, interval whose right end point is maximum among all adjacent to k and all unmarked adjacent are placed as the child of k and marked them else m_1' as child of j and marked, where $m_1' = max\{b_{m_1'} : m_1' \in N(k) \cap N(j)\}$ and all members of $\{N(k) \cup N(j) - \{m_1'\}\}$ as child of k and marked and find $m_1'' = max\{N(k) \cup N(j) - \{m_1'\}\}$. This process is continued until all intervals right to 1 are marked.

(b) When i is the end node in G, i.e. $i = n$:

In this case we find all adjacent vertices to the vertex n and set them as child (leaves) of n and marked them. Next we consider the vertices j' and k' whose left end points of the corresponding intervals are minimum and next minimum respectively. Next find all unmarked adjacent vertices to the vertices j' and k' respectively. If there is no common adjacent vertices to j' and k', then find m_1, interval whose left end point is minimum among all adjacent to j' and unmarked adjacent are placed as the child of j' and marked them else m_1' as child of k', where $m_1' = min\{a_{m_1'} : m_1' \in N(k') \cap N(j')\}$ and all members of $\{N(k') \cup N(j') - \{m_1'\}\}$

as child of j' and marked and find $m_1'' = min\{a_{m_1''} : m_1'' \in \{N(k') \cup N(j') - \{m_1'\}\}\}$. This process is continued until all intervals left to n are marked.

(c) When i is the node between 1 and n, i.e. $1 < i < n$:

In this case we find all adjacent vertices to the vertex i and set them as child (leaves) of i and marked them. Next, we consider the vertex k whose right end point of the corresponding interval among all adjacent vertices to i is maximum. Corresponding to the vertex k we find all unmarked vertices adjacent to k and put them as the child of k. Continuing this process on the right side of the interval diagram until all vertices corresponding to the intervals right of i are marked. Similarly, on the left side of i, we find j', the unmarked adjacent to i, and put them as child in left branch. Then same procedure is applied on left side until all intervals left of i are marked.

Now, we propose a combinatorial algorithm to construct the tree T_{IG}. Our proposed algorithm is as follows:

Algorithm INT-TREE

Input: Fuzzy InvG G with interval representation $I = [i_1, i_2, \ldots, i_n]$, $i_j = [a_j, b_j]$ and weight $w_j = [a_j, b_j]$, $j = 1, 2, \ldots, n$ as interval number.

Output: The rooted tree T_{IG} with two branches of the fuzzy InvG G.

Step 1. Set root $= i$ and compute $N(i) =$ the open neighbourhood of $i = \{v : (v, i) \in E\}$.

Step 2. If $|N(i)| = 1$, then end.

If $|N(i)| > 1$ and i is the starting interval, i.e. $i = 1$, then goto Step 3.

If $|N(i)| > 1$ and i is the end interval, i.e. $i = n$, then goto Step 4.

If $|N(i)| > 1$ and i is an interval between 1 and n, i.e. $1 < i < n$, then goto Step 5.

Step 3. Set $N(i)$ as the child of the root i and marked them.

Step 3.1. Set $k = max\{b_k : (k, i) \in E\}$,

$j = max\{b_j : (j, i) \in E, k \neq j \ \ and \ \ b_j < b_k\}$.

Step 3.2. Find unmarked adjacent of j and k and if $N(j) \bigcap N(k) = \phi$, then $m_1 = max\{b_{m_1} : (m_1, k) \in E, m_1 \in N(k)\}$ and set all unmarked $N(k)$ as the child of k and marked them.

else $m_1' = max\{b_{m_1'} : m_1' \in N(k) \cap N(j)\}$ set as child of j and $\{N(k) \cup N(j) - \{m_1'\}\}$ as child of k and marked and find $m_1'' = max\{N(k) \cup N(j) - \{m_1'\}\}$.

Step 3.3. This process is continued until all intervals are marked .

Step 3.4. Compute the interval tree T_{IG}.

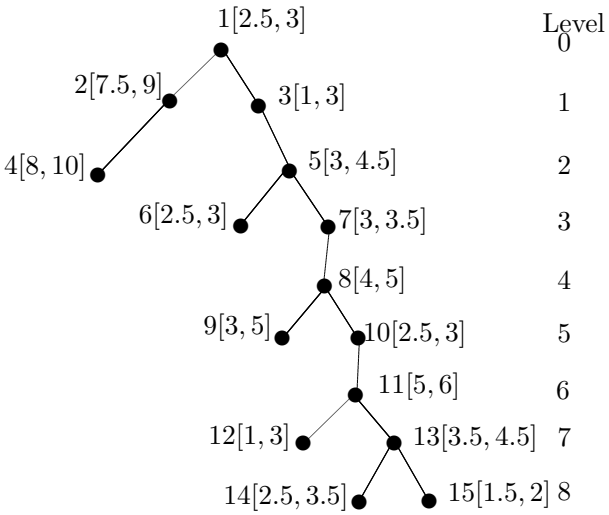

Figure 3.7: Tree T_{IG} of the fuzzy interval graph G.

Step 4. Set $N(i)$ as the child of the root i and marked them.

Step 4.1. Set $j' = min\{a_{j'} : (j', i) \in E\}$,

$k' = min\{a_{k'} : (k', i) \in E, k' \neq j' \ and \ a'_j < a'_k\}$.

Step 4.2. Find unmarked adjacent of j' and k' and if

$N(j') \bigcap N(k') = \phi$, then $m_1 = min\{a_{m_1} : (m_1, j') \in E, m_1 \in N(j')\}$

and set all unmarked $N(j')$ as the child of j' and marked them.

else $m'_1 = min\{a_{m'_1} : m'_1 \in N(k') \cap N(j')\}$ set as child of k'

and $\{N(k') \cup N(j') - \{m'_1\}\}$ as child of j' and marked and find

$m''_1 = min\{a_{m''_1} : m''_1 \in \{N(k') \cup N(j') - \{m'_1\}\}\}$.

Step 4.3. This process is continued until all intervals are marked.

Step 4.4. Compute the interval tree T_{IG}.

Step 5. Set $N(i)$ as the child of the root i and marked them.

Step 5.1. Set $p = max\{b_p : (p, i) \in E\}$, $q = min\{a_q : (q, i) \in E\}$

and $p \neq q$.

Step 5.2. Set $p' = max\{b_{p'} : (p', p) \in E, p' \in N(p)\}$ and set all

unmarked $N(p)$ as the child of p and marked.

Step 5.3. Set $q' = min\{a_{q'} : (q', q) \in E, q' \in N(q)\}$ and set all

unmarked $N(q)$ as the child of q and marked.

Step 5.4. This process is running till all intervals are marked.

Step 5.5. Compute the interval tree T_{IG}.

Step 6. Put weight $w_j = [a_j, b_j]$, $j = 1, 2, \ldots, n$ to the vertex j in T_{IG}

corresponding to the interval j of the fuzzy InvG G.

end INT-TREE.

Illustration of the Algorithm INT-TREE : Let $i = 1$ be the pre-specified node which is the root. Next the open neighbourhood of 1 is $N(1) = \{2, 3\}$, where the nodes of $N(1)$ as the child of the root 1. Next, 3 has the maximum right extremity b_j of the intervals of $N(1)$ correlative to the nodes of the graph G and 2 has the next maximum right extremity of $N(1)$ correlative to the nodes of the graph G. Next the open neighbourhoods of 3 and 2 are $N(3) = \{4, 5\}$ and $N(2) = \{4\}$ respectively, where the nodes of $N(3)$ and $N(2)$ as the child of the roots 3 and 2. Next 5 has the maximum right extremity of the intervals of $N(3)$ correlative to the nodes of the graph G and 4 has the minimum tail of the intervals of $N(2)$ correlative to the nodes of the graph G. Next the open neighbourhood of 5 is $N(5) = \{6, 7\}$, where the nodes of $N(5)$ as the child of the root 5. Next 7 has the maximum right extremity of the intervals of $N(5)$ correlative to the nodes of the graph G. Next the open neighbourhood of 7 is $N(7) = \{8\}$, where the vertex of $N(7)$ as the child of the root 7. Next the open neighbourhood of 8 is $N(8) = \{9, 10\}$, where the vertices of $N(8)$ as the child of the root 8. Next 10 has the maximum right extremity of the intervals of $N(8)$ correlative to the vertices of the graph G. Next the open neighbourhood of 10 is $N(10) = \{11\}$, where the vertex of $N(10)$ as the child of the root 10. Next 11 has the maximum right extremity of the intervals of $N(10)$ correlative to the nodes of the graph G. Next the open neighbourhood of 11 is $N(11) = \{12, 13\}$, where the nodes of $N(11)$ as the child of the root 11. Next 13 has the maximum right end point among the intervals of $N(11)$ correlative to the vertices of the graph G. Next the open neighbourhood of 13 is $N(13) = \{14, 15\}$, where the nodes of $N(13)$ as the child of the root 13. Next 15 has the maximum right extremity of the intervals of $N(13)$ correlative to the nodes of the graph G. In this way we get longest left path L_i from the vertex 1 to other vertex 4, i.e. the path $1 \to 2 \to 4$ and find longest right path R_i from 1 to the vertex 15 does not contain any vertex of the path L_i except 1, i.e. the path $1 \to 3 \to 5 \to 7 \to 8 \to 10 \to 11 \to 13 \to 15$. Next put the weights $[8, 10], [7.5, 9], [2.5, 3], [1, 3], [3, 4.5], [3, 3.5], [4, 5], [2.5, 3], [5, 6], [3.5, 4.5], [1.5, 2],$
$[2.5, 3], [3, 5], [1, 3], [2.5, 3.5]$ to the vertices $4, 2, 1, 3, 5, 7, 8, 10, 11, 13, 15, 6, 9, 12, 14$ respectively. Finally we construct the tree T_{IG} with root $i = 1$ (Figure 3.7).

We have the following important observation on T_{IG}.

Lemma 3.4.1 *The tree T_{IG} is constructed by the Algorithm **INT-TREE** is a spanning tree.*

Proof. As per construction of the tree T_{IG} by end points scanning approach of the intervals we get n vertices and $(n-1)$ edges without any circuit. Therefore the tree T_{IG} is a spanning tree. Hence the result. □

Theorem 3.4.1 *The Algorithm **INT-TREE** is finished in $O(n)$ time.*

Proof. Step 1 runs $O(n)$ time, as the intervals are sorted and the root is selected from n intervals. Step 2 is completed in $O(n)$ time, since number of intervals is n. Since the extremities of the intervals are sorted, so the maximum element (node) from a set of nodes can be completed 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 3, or Step 4, or Step 5 can be finished in $O(n)$ time. So overall time complexity of the Algorithm **INT-TREE** is $O(n)$ time, where n is the number of nodes of the weighted InvG. Since weight as interval number of the vertex in tree T_{IG} corresponds the weight of the interval in fuzzy InvG, so it is one to one corresponds, and hence Step 6 runs in $O(n)$ time. □

Thus, the tree T_{IG} of the fuzzy InvG is formed. The tree T_{IG} with root as 1 of the InvG G (Figure 3.7). Next we put the weight as interval number of interval diagram to the corresponding vertex in the tree T_{IG}.

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

Lemma 3.4.2 *If $x, y \in V$ and $|level(x) - level(y)| > 1$ in T_{IG}, then there is no edge between the nodes x and y in G, except such $(x, y) \in E$ in which $level(x) = 1$ and $level(y) = k$, where k is the highest level.*

Proof. Let $|level(x) - level(y)| > 1$ but $(x, y) \in E$, i.e. x and y are connected directly. So by the idea of breath first search, at any stage if x, y are the adjacent to the previously visited node, then x and y to be placed in same level, so $level(x) = level(y)$. But, if x is adjacent to a previously visited node then y must be adjacent to next visited node, then y to be placed in the next level in T_{IG}. So, in this case $|level(x) - level(y)| = 1$. Thus, either $level(x) = level(y)$ or $|level(x) - level(y)| = 1$ implies $(x, y) \in E$, which is reverse to the assumption $|level(x) - level(y)| > 1$, $(x, y) \in E$.Hence the result. □

Now, we shall prove that the BFS tree is a MdsT.

Lemma 3.4.3 *The spanning tree T_{IG} is a MdsT.*

Proof. According to the construction the BFS tree, the main path of the tree T_{IG} is the greatest path called diameter. The main path covers the whole interval with least number of intervals. This diameter is minimum, because T_{IG} is the minimum height tree. Also, T_{IG} is a spanning tree. Hence T_{IG} is a MdsT. □

In next subsection, we shall discuss about the modified spanning tree T'_{IG} of the spanning tree T_{IG}.

3.4.3 Modification of the spanning tree T_{IG}

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

Firstly, 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_{IG} 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_{IG} 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_{IG}.

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_{IG}.

Similar idea is used for pair of any two nodes on the main path P. Using this method we construct the modified spanning tree $T_{IG}^{'}$ starting from T_{IG}. The tree $T_{IG}^{'}$ (Figure 3.8) is the modified BFS tree of the BFS tree T_{IG}.

Lemma 3.4.4 If $T_{IG}^{'}$ is a BFS tree, then the distance $d_{T_{IG}^{'}}(u, v)$ between the vertices u and v in $T_{IG}^{'}$ is given by

$$d_{T_{IG}^{'}}(u,v) = \begin{cases} level(v), \text{if } u \text{ is a root and } v \text{ is any vertex,} \\ |level(v) - level(u)|, \text{if } u \text{ and } v \text{ both are nodes in the main path } P, \\ |level(parent(u)) - level(v)| + 1, \text{if } u \text{ is any leaf and } v \text{ is any node} \\ \text{in the main path P.} \end{cases}$$

Proof. Case I: If u is a root.

In the tree $T_{IG}^{'}$, 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

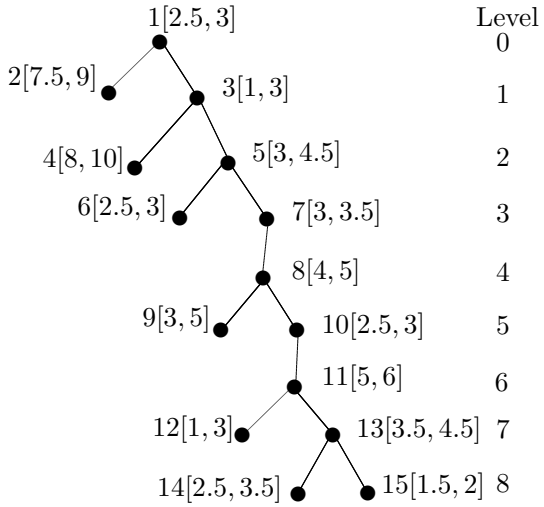

Figure 3.8: Modified BFS tree $T_{IG}^{'}$ of the tree T_{IG}.

is directly connected with the next one, hence the length of this path is $p = level(v)$. Thus $d_{T_{IG}^{'}}(u, v) \leq p$. Next we are to show that $d_{T_{IG}^{'}}(u, v) \not< p$. If possible, let $d_{T_{IG}^{'}}(u, v) = q < p$.

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 $d_{T_{IG}^{'}}(u, v) \not< p$ which implies that $d_{T_{IG}^{'}}(u, v) = p$, i.e. $d_{T_{IG}^{'}}(u, v) = level(v)$.

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 $d_{T_{IG}^{'}}(u, v) = |level(v) - level(u)|$.

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. Therefore $d_{T_{IG}^{'}}(u, v) = d(u, parent(u)) + d(parent(u), v) = 1 + level(v) - level(parent(u))$,

i.e. $d_{T_{IG}^{'}}(u, v) = |level(parent(u)) - level(v)| + 1$. □

3.4.4 Average distance of interval graph

Many works on average distance in graphs are available in literature [4, 10, 41, 53, 70, 100]. Chung [23] 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 [29], the average distance of an InvG 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 fuzzy InvG.

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.

3.4.5 Algorithm and its complexity on average distance

At first we compute $d_G(u, v)$ for every pair u, v $(u \neq v)$, then we compute the sum of distances between all pairs of vertices and finally we multiply it by the factor $2/\{n(n-1)\}$ 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.

Here we discuss an algorithm to construct MADT for a given fuzzy InvG. Also time complexity are presented here.

Algorithm INMAD-TREE

Input: Sorted endpoints of the intervals $I = [i_1, i_2, \ldots, i_n]$, $i_j = [a_j, b_j]$ of the interval diagram of the fuzzy InvG $G = (V, E)$.

Output: MADT T'_{IG} and average distance $\mu(T'_{IG})$.

Step 1: Work out the MdsT T_{IG} //Subsection 3.4.2//
and determine $level(u) = d(u^*, u)$, u^* is either the vertex corresponding to the interval of maximum right end points of the intervals, which are adjacent to i_1.

Step 2: Work out $N(u_i^*)$ for all $u_i^* \in T$ and k = height of the tree T_{IG}.

Step 3: //Modification of the tree T_{IG}//
Work out $N(u_i^*)$ and $N(u_{i+1}^*)$ for $i = 0, 1, 2, \ldots, k-1$ on the main path P.

Step 3.1: For $i = 0$ to $k - 1$ do

If $N(u_i^*) \cap N(u_{i+1}^*) = \phi$ then go to Step 3.1,

If $N(u_i^*) \cap N(u_{i+1}^*) \neq \phi$ and let $w \in N(u_i^*) \cap N(u_{i+1}^*)$ then,

Step 3.1.1: Determine the number of vertices k_1, k_2 respectively, on the both sides separately of the node u_i^* (taken as fixed) in T_{IG} along the main path.

Next, determine their difference, say, $d_1 = |k_1 - k_2|$.

Step 3.1.2: Determine the number of vertices on the both sides separately of the node u_{i+1}^* (taken as fixed) in T_{IG} along the main path (ignoring the vertex obtained in Step I).

Next determine their difference, say, d_2.

Step 3.1.3: If $d_1 - d_2 < 0$, then unaltered;

if $d_1 - d_2 = 0$, then determine $D_1 = \sum_{u,v \in V(T_1)} d_{T_1}(u, v)$ (total distance before shifting) and $D_2 = \sum_{u,v \in V(T_{IG})} d_{T_1}(u, v)$ (total distance after shifting) and then consider $minimum\{D_1, D_2\}$;

if $d_1 - d_2 > 0$, then the adjacent vertex w of the node u_i^* in the main path P is shifted to the node u_{i+1}^* in the main path P.

Step 3.2: Set $T_{IG}^{'}$ as modified spanning tree of T_{IG} of the fuzzy InvG G.

Step 4: Determine $d_{T_{IG}^{'}}(u, v)$ //Lemma 3.4.4//

Where,

$$d_{T_{IG}^{'}}(u,v) = \begin{cases} level(v), \text{if u } \textit{is a root and} \text{ v } \textit{is any vertex}, \\ |level(v) - level(u)|, \text{if u } \textit{and} \text{ v } \textit{both are nodes in the main path} \text{ P}, \\ |level(parent(u)) - level(v)| + 1, \text{if u } \textit{is any leaf and} \text{ v } \textit{is any node in the main path P}. \end{cases}$$

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

end INMAD-TREE.

Illustration of the Algorithm INMAD-TREE :

In Figure 3.7, which is the spanning tree of the fuzzy InvG G, $u_i^* = 2$ and $u_{i+1}^* = 3$ are two successive nodes. 4 is the similar adjacent of the nodes $u_i^* = 2$ and $u_{i+1}^* = 3$, i.e. $w = 4$. Taking the vertex 2 as origin, the number of nodes of one side of 2 is 0 and the number of nodes of other side of 2 is 13. So their variation is $d_1 = 13 - 0 = 13$.

Now considering the node 3 as origin, the number of nodes of one side of 3 is 2 (ignoring the node 4) and the number of nodes of other side of 3 is 11 when the node 4 is adjacent with the node 3. Hence their variation is $d_2 = 11 - 2 = 9$. Therefore $d_1 > d_2$. So, the node 4 is agitate to the node 3. Then we have the modified spanning tree $T_{IG}^{'}$(Figure 3.8).

Next determine the average distance $\mu_1(G)$ equivalent to the tree T_{IG}. Again determine

average distance $\mu_2(G)$ equivalent to the tree T'_{IG}. Here $\mu_1(G) = [x_l, x_r] = [18.85, 22.25]$ and $\mu_2(G) = [y_l, y_r] = [15.78, 20.88]$. Let $m_1 = mean(\mu_1(G))$, $m_2 = mean(\mu_2(G))$, $w_1 = width(\mu_1(G))$ and $w_2 = width(\mu_2(G))$. Here, $m_1 = 20.55, m_2 = 18.33, w_1 = 1.7$ and $w_2 = 2.55$.

Therefore, $A(\mu_1(G) > \mu_2(G)) = 2.22/4.25 = 0.52235$. Hence $0 < A(\mu_1(G) > \mu_2(G)) < 1, m_1 > m_2$ and $x_l < y_r$.

Clearly, $\mu_1(G) > \mu_2(G)$. Hence T'_{IG} is the minimum average distance tree of fuzzy InvG G.

Next we shall show that for any fuzzy InvG, the tree designed by the **Algorithm INMAD-TREE** represents a MADT.

Theorem 3.4.2 *For any fuzzy InvG, the tree designed by the* ***Algorithm INMAD-TREE*** *is a MADT.*

Proof. Let $G = (V, E)$ be any fuzzy InvG. Then, using Subsection 3.4.2, one can design spanning tree with minimum diameter. In this tree, shifting (if necessary, under conditions stated in Subsection 3.4.3) of the some nodes to its next adjacent node in the main path means that those nodes 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 nodes. As a result, after all possible shifting of the nodes, 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 INMAD-TREE** is a MADT for any fuzzy InvG. This completes the proof. □

Now we discuss the T-complexity of the **Algorithm INMAD-TREE**.

Theorem 3.4.3 *The MADT of the fuzzy InvG G with n nodes can be completed in $O(n^2)$ time.*

Proof. In **Algorithm INMAD-TREE** Step 1 takes $O(n)$ time. Step 2 can be finished in $O(n^2)$ time. $O(n)$ time required to complete each Step 3.1.1 and 3.1.2. Also Step 3.1.3 takes the time $O(n)$. Total T-complexity of Step 3.1 is $O(n^2)$ as the Step 3.1 repeats $(k-1)$ times, where k is of $O(n)$. Again $O(n^2)$ time is taken to complete the Step 3.2. The last step, i.e. Step 4, in worst case, can be computed in $O(n^2)$ time. Finally the T-complexity of our proposed algorithm is $O(n^2)$. □

3.5 Summary

In that chapter, we investigated the Inv1C location problem with different vertex weights on the tree for the weighted InvG G. We developed an combinatorial algorithm to construct

the *tree* of InvG in $O(n)$ time, where $|V| = n$. Also in this chapter, we designed MADT on fuzzy InvG. Also, we investigated the MADT on the fuzzy InvG which is designed based on BFS technique. The algorithm is completed in $O(n^2)$ time, where $|V| = n$.