Chapter 6

Computation of inverse 1-center location problem on the weighted trapezoid graphs*

6.1 Introduction

Let the graph $G = (V, E)$ is simple, connected and undirected weighted TraGs. Figure 6.1 represents a TraG and it's TraD is shown in Figure 6.2. The class of TraG includes two well known classes of Int: the PerG and the InvG [50]. The PerG are obtained in the case where $a_i = b_i$ and $c_i = d_i$ for all i and the InvG are obtained in the case where $a_i = c_i$ and $b_i = d_i$ for all i. TraG can be completed in $O(n^2)$ time [62]. The TraG were first studied in [24, 26]. These graphs are superclass of InvG, PerG and subclass of CcoG [60].

6.1.1 Organization of the chapter

Section 6.2 describes the construction of the tree T_{TRP}. In Section 6.3, we discuss the Inv1C and some notations. In Section 6.4, we establish an algorithm to get Inv1C of the modified node weighted tree corresponding to the TraG G. The T-complexity is also calculated in this section. In Section 6.5, we give the summary.

---

*A part of the work presented in this chapter is published in *Missouri Journal of Mathematical Science*, 31 (2019) 1-22.

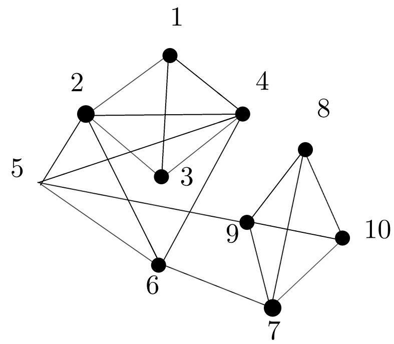

Figure 6.1: A TraG G.

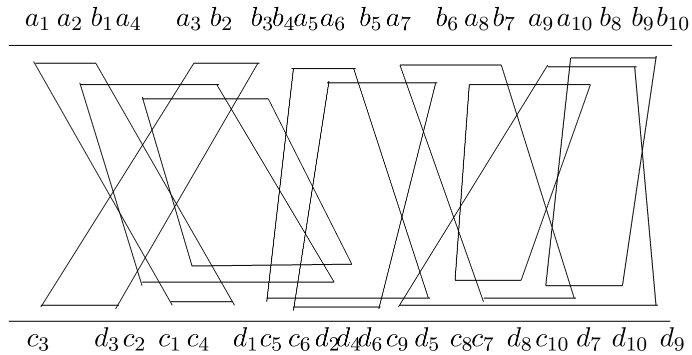

Figure 6.2: A TraD T of the TraG G of Figure 6.1.

6.2 Construction of minimum height tree for trapezoidal graph

Let i be pre-specified node which to be Inv1C. In this section, our aim is to construct a minimum height tree, as root i, with two branches of level difference either zero or one.

Let the node i be the root of the tree. Then we find all adjacent nodes to i correlative to the trapezoid and set them as child (leaves) of i. Next consider the nodes k and j, where $k = max\{b_k \ or \ d_k : (k,i) \in E\}$, $j = max\{b_j \ or \ d_j : (j,i) \in E, \ k \neq j \ and \ b_j < b_k \ or \ d_j < d_k\}$ and set them as a nodes on the main path and marked them. Next find all adjacent trapezoids to the nodes k and j and set them as respective child (leaves). This process is continue until all trapezoids 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_{TRP} is as follows:

Algorithm TRP-TREE

Input: Weighted TraG G with four corner points $[a_i, b_i, c_i, d_i]$, $i = 1, 2, \ldots, n$ and $T = 1, 2, \ldots, n$ be the set of n trapezoids.

Output: The rooted tree T_{TRP} with two branches of the TraG 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 trapezoid, i.e. $i = 1$, then go to Step 3.

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

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

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

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

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

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

$m_2 = max\{b_{m_2} \ or \ d_{m_2} : (m_2,j) \in E, m_2 \in N(j)\}$ and set all unmarked $N(j)$ as the child of j and marked them.

else $m_1' = max\{b_{m_1'} \ or \ d_{m_1'} \in N(k) \cap N(j)\}$ 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 trapezoids are marked.

Step 3.4. Compute the trapezoid tree T_{TRP}.

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

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

$k' = min\{a_{k'} \ or \ c_{k'} : (k',i) \in E, k' \neq j' \ and \ a_j' < a_k' \ or \ c_j' < c_k'\}$.

Step 4.2. Find unmarked adjacent of j' and k' and if $N(j') \bigcap N(k') = \phi$, then $r_1 = min\{a_{r_1} \ or \ c_{r_1} : (r_1,j') \in E, r_1 \in N(j')\}$ and set all unmarked $N(j')$ as the child of j' and marked them and

$r_2 = min\{a_{r_2} \ or \ c_{r_2} : (r_2,k') \in E, r_2 \in N(k')\}$ and set all unmarked $N(k')$ as the child of k' and marked them.

else $r_1' = min\{a_{r_1'} \ or \ c_{r_1'} : r_1' \in N('k) \cap N(j')\}$ set as child of j' and $\{N(k') \cup N(j') - \{r_1'\}\}$ as child of k' and marked and find

$r_1'' = min\{N(k') \cup N(j') - \{r_1'\}\}$.

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

Step 4.4. Compute the trapezoid tree T_{TRP}.

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

Step 5.1. Set $p = max\{b_p \ or \ d_p : (p,i) \in E\}$,

$q = min\{a_q \ or \ c_q : (q,i) \in E\}$ and $p \neq q$.

Step 5.2. Set $p' = max\{b_{p'} \ or \ d_{p'} : (p', p) \in E, p' \in N(p)\}$ and set all unmarked $N(p)$ as the child of p and marked.

Step 5.3. Set $q' = min\{a_{q'} \ or \ c_{q'} : (q', q) \in E, q' \in N(q)\}$ and set all unmarked $N(q)$ as the child of q and marked.

Step 5.4. This process is continued until all trapezoids are marked.

Step 5.5. Compute the tree T_{TRP}.

Step 6. Put weight $w_j (> 0)$ to the node j in T_{TRP} corresponding to the trapezoid j of the TraG G.

end TRP-TREE.

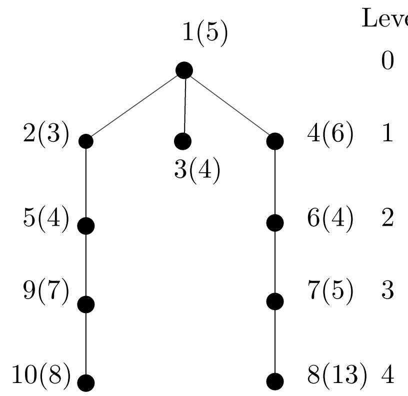

Figure 6.3: Tree T_{TRP} of the TraG G.

Illustration of the Algorithm TRP-TREE : Let $i = 1$ be the pre-specified node which is the root whose level is 0. Next the open neighbourhood of 1 is $N(1) = \{2, 3, 4\}$, where the nodes of $N(1)$ as the child of the root 1 and put them at level 1. Next, 4 has the maximum b_i among the trapezoids of $N(1)$ corresponding to the nodes of the graph G and 2 has the next maximum d_i among the trapezoids of $N(1)$ corresponding to the nodes of the graph G. Next the open neighbourhoods of 4 and 2 are $N(4) = \{5, 6\}$ and $N(2) = \{5, 6\}$ respectively, where the nodes of $N(4)$ and $N(2)$ as the child of the roots 4 and 2 and put them at level 2. Next 6 has the maximum b_i among the trapezoids of $N(4)$ corresponding to the nodes of the graph G and 5 has the next maximum b_i among the trapezoids of $N(2)$ corresponding to the nodes of the graph G. Next the open neighbourhoods of 6 and 5 are $N(6) = \{7\}$ and $N(5) = \{9\}$ respectively, where the nodes of $N(6)$ and $N(5)$ as the child of the roots 6 and 5 and put them at level 3. Next 7 has the maximum d_i among the trapezoids of $N(6)$ corresponding to the nodes of the graph G and 9 has the maximum d_i among the trapezoids of $N(5)$ corresponding to the nodes of the graph G. Next the open neighbourhoods of 7 and 9 are $N(7) = \{8, 10\}$ and $N(9) = \{8, 10\}$ respectively, where the nodes of $N(7)$ and $N(9)$ as the child of the roots 7 and 9 and put them at level 4. Finally we construct the rooted tree T_{TRP} with root $i = 1$ (Figure 6.3).

Now we have the following important observation on T_{TRP}.

Lemma 6.2.1 *The tree T_{TRP} formed by the **Algorithm TRP-TREE** is a spanning tree.*

Proof. As per construction of the graph T_{TRP} by maximum b_i or d_i, $i = 1, 2 \ldots, n$, TraD we get n nodes and $(n-1)$ edges. Also there is no repetition of the nodes, as we search only unmarked nodes, so this is a graph without any circuit. Therefore the tree T_{TRP} is a spanning tree.

Hence the result. □

Lemma 6.2.2 *The tree T_{TRP} formed by the **Algorithm TRP-TREE** is a BFS tree with minimum height.*

Proof. Actually steps of the algorithm indicates the steps of BFS technique in TraG. Thus the tree formed by the *Algorithm TRP-TREE* is BFS tree. Again we traverse the TraG with respect to maximum b_i or d_i until all unmarked trapezoids are marked. As in each step we move on trapezoid, so, its height to be minimum.

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

Theorem 6.2.1 *The T-complexity of the **Algorithm TRP-TREE** is $O(n)$, where n is the number of nodes of the tree.*

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

Thus the tree T_{TRP} of the TraG is formed. The tree T_{TRP} of the TraG G (Figure 6.1) is shown in Figure 6.3.

6.3 Inverse 1-center location problem for trapezoidal graph

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 node in G.

R_i	:	Longest path to the node i.
L_i	:	Another longest path to the node i.
$w(R_i)$	:	Total weight of the nodes except the node i of the path R_i.
$w(L_i)$	:	Total weight of the nodes except the node i of the path L_i.
$w_{low}(v)$	:	Minimum weight of the node in G.
$w_{upp}(v)$	:	Maximum weight of the node in 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 nodes in such path between L_i, R_i whose weight is maximum, except the node i.
k_2	:	The number of nodes in such path between L_i, R_i whose weight is minimum, except the node i.
T_{TRP}	:	Weighted tree corresponding to the TraG G.
T'_{TRP}	:	Modified tree of the tree T_{TRP} corresponding to the TraG G.
$w^*(R_i)$	:	Total weight of the nodes except the node i of the path R_i after modification.
$w^*(L_i)$	:	Total weight of the nodes except the node i of the path L_i after modification.

To find the Inv1C, we discuss following cases:

1. If total weight of one side of the node i is same as the total weight 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 same as the multiplication of the number of nodes except the node i in the path whose weight is maximum and minimum weight of the node in the graph, i.e. $w_{min} = k_1 w_1$.

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

Case-2.4. : When w_{max} is same as the multiplication of the number of nodes except the node i in the path whose weight is minimum and maximum weight of the node in the graph, i.e. $w_{max} = k_2 w_2$.

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

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

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

subject to

$\max_{v_1 \in v_1(T_{TRP})} d_{\overline{w}}(v_1, i) \leq \max_{v_1 \in v_1(T_{TRP})} d_{\overline{w}}(v_1, q)$, for all $q \in T_{TRP}(\text{or}\, q \in v_1(T_{TRP}))$,

$\overline{w}(v_1) = w(v_1) + x_1\{w(v_1)\} - y_1\{w(v_1)\}$ for all $v_1 \in v_1(T_{TRP})$,

$x_1\{w(v_1)\} \leq w^+\{w(v_1)\}$, for all $v_1 \in v_1(T_{TRP})$,

$y_1\{w(v_1)\} \leq w^-\{w(v_1)\}$, for all $v_1 \in v_1(T_{TRP})$,

$x_1\{w(v_1)\}, y_1\{w(v_1)\} \geq 0$, for all $v_1 \in v_1(T_{TRP})$, where $\overline{w}(v_1)$ be the modified node weight, $w^+\{w(v_1)\} = w_{upp}(v_1) - w(v_1)$ and $w^-\{w(v_1)\} = w(v_1) - w_{low}(v_1)$ are the maximum feasible amounts by 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 maximum amounts by which the node weight $w(v_1)$ is increased and reduced respectively, $c_1^+(w(v_1))$ is the non negative cost if $w(v_1)$ is increased by one unit and $c_1^-(w(v_1))$ is the non negative cost if $w(v_1)$ is reduced by one unit. Every feasible solution (x_1, y_1) with $x_1 = \{x_1(w(v_1)) : v_1 \in v_1(T_{CIR})\}$ and $y_1 = \{y_1(w(v_1)) : e \in v_1(T_{TRP})\}$ is also called a feasible modification of the Inv1C location problem.

Now, we prove the next results.

Lemma 6.3.1 *If $w_{min} = k_1 w_1$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all nodes except the node 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 the node among the nodes in T_{TRP} as well as L_i or R_i, then there is a scope to reduce weight of each node up to w_1. As k_1 nodes 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_{TRP}, say T'_{TRP}. Again, since the TraG is an arbitrary, so our assumption is true for any TraG.

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

Lemma 6.3.2 *If $w_{min} > k_1 w_1$ in T_{TRP}, 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_{TRP}, 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 $k_1 w_1$. Again we have $w_{min} > k_1 w_1$. Therefore we can decrease the weights $(w_{max} - w_{min})$ from the nodes except the root i in the path whose weight is maximum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 6.3). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{TRP}, say T'_{TRP}. Again, since the TraG is an arbitrary, so our assumption is true for any TraG.

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

Lemma 6.3.3 *If $w_{min} < k_1 w_1$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all nodes 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 nodes 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_{TRP}, so we can reduce the weights of the nodes except the root in the path whose weight is maximum in such a way that its least weight of the path becomes $k_1 w_1$. Again we have $w_{min} < k_1 w_1$. Therefore we can increase the weights $(k_1 w_1 - w_{min})$ to the nodes except the root in the path whose weight is minimum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 6.3). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{TRP}, say T'_{TRP}. Again, since the TraG is an arbitrary, so our assumption is true for any TraG.

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

Lemma 6.3.4 *If $w_{max} = k_2 w_2$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all nodes 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 node among the nodes in T_{TRP} as well as L_i or R_i, then there is a scope to increase the weight of each node up to w_2. As k_2 nodes is there in the path L_i or R_i, so we can enhance atmost $k_2 w_2$ weight and hence enhanced weight of the path L_i or R_i becomes $k_2 w_2$. Again we have $w_{max} = k_2 w_2$. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{TRP}, say T'_{TRP}. Again, since the TraG is an arbitrary, so our assumption is true for any TraG.

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

Lemma 6.3.5 *If $w_{max} > k_2 w_2$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all nodes up to maximum weight except the root i maintaining the bounding condition in path whose weight is minimum and reducing the weights of some nodes except the root i in the path whose weight is maximum and i is the Inv1C.*

Proof. Since we can increase the weight of each node up to maximum weight of the node in T_{TRP}, so we can enhance the weights of all nodes 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 nodes except the root in the path whose weight is maximum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 6.3). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{TRP}, say T'_{TRP}. Again, since the TraG is an arbitrary, so our assumption is true for any TraG.

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

Lemma 6.3.6 *If $w_{max} < k_2w_2$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of some nodes 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_{TRP}, so we can enhance the weights of the nodes 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 nodes except the root i in the path whose weight is minimum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 6.3). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{TRP}, say T'_{TRP}. Again, since the TraG is an arbitrary, so our assumption is true for any TraG.

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

6.4 Algorithm and its complexity

In this section we suggested a combinatorial algorithm for the Inv1C location problem on the weighted tree T_{TRP}. The main idea of our suggested algorithm is as follows:

Let T_{TRP} be a weighted tree corresponding to the TraG G with n nodes and $(n-1)$ edges. Let V be the node set and E be the edge set. Let i be any non-pendant specified node in the tree T_{TRP} which is to be Inv1C. At first we calculate the path whose weight is maximum from i to any pendant node of T_{TRP}. 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 except the root of the paths L_i, R_i respectively with respect to the node 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_{TRP}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of nodes 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 nodes up to minimum weight except the node i, i.e.,root i maintaining the bounding conditions (Section 6.3) 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_{TRP}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of nodes 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 nodes except the root i maintaining the bounding conditions (Section 6.3) in the path whose weight is maximum and i is the Inv1C. In third case, if $w_{min} < k_1 w_1$ in T_{TRP}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of nodes 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 nodes up to minimum weight except the root i maintaining the bounding conditions (Section 6.3) in the path whose weight is maximum and enhance the weights of some nodes 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_{TRP}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of nodes 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 nodes up to maximum weight except the root i maintaining the bounding conditions (Section 6.3) in the path whose weight is minimum and i is the Inv1C. In fifth case, if $w_{max} > k_2 w_2$ in T_{TRP}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of nodes 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 nodes up to maximum weight except the root i maintaining the bounding conditions (Section 6.3) in path whose weight is minimum and reducing the weights of some nodes except the root i in the path whose weight is maximum and i is the Inv1C. In sixth 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 nodes 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 nodes except the root i maintaining the bounding conditions (Section 6.3) in the path whose weight is minimum and i is the Inv1C.

Our proposed algorithm to the Inv1C location problem of the tree for the TraG G is as follows:

Algorithm 1-INV-TRP-TREE

Input: Weighted TraG $G = (V, E)$ with its TraD $T_i = [a_i, b_i, c_i, d_i]$, $i = 1, 2, \ldots, n$.

Output: Vertex i as the Inv1C of the TraG $G = (V, E)$ with the help of its tree T'_{TRP}.

Step 1. Construction of the tree T_{TRP} with root i //Algorithm TRP-TREE//.

Step 2. Compute the paths R_i and L_i.

Step 3. Calculate $w(L_i)$ and $w(R_i)$.

Step 4. //Modification of the tee T_{TRP}//

Step 4.1. If $w(L_i) = w(R_i)$, then i is the Inv1C of T_{TRP}.

Step 4.2. If $w(L_i) \neq w(R_i)$, then

Step 4.2.1. If $w_{min} = k_1 w_1$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all nodes except the node 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 $w_{min} > k_1 w_1$ in T_{TRP}, 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, then go to Step 4.3.

Step 4.2.3. If $w_{min} < k_1 w_1$ in T_{TRP}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all nodes 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 nodes 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_{TRP}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all nodes 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_{TRP}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all nodes except the root i up to maximum weight maintaining the bounding condition in path whose weight is minimum and reducing the weights of some nodes 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_{TRP}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of some nodes 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'_{TRP} of the tree T_{TRP} with

$w^*(L_i) = w^*(R_i)$, and i is the Inv1C.

end 1-INV-TRP-TREE.

Using above **Algorithm 1-INV-TRP-TREE** we can find out the Inv1C location problem on any weighted tree. Justification of this statement follows the following illustration.

Illustration of the Algorithm 1-INV-TRP-TREE to the tree T_{TRP} in Figure 6.3 : Let $i = 1$ be the pre-specified node of the tree T_{TRP} which is to be Inv1C. Next we find the longest path L_i from the node 1 to other node 10, i.e. the path $1 \to 2 \to 5 \to 9 \to 10$ and find another longest path R_i from 1 to the node 8 does not contain any node of the path L_i except 1, i.e. the path $1 \to 4 \to 6 \to 7 \to 8$.

Next calculate the weights of the paths L_i and R_i. Let $w(L_i)$ and $w(R_i)$ be the total weights of the nodes except the root $i = 1$ of the paths L_i and R_i respectively. Here $w(L_i) = 22$ and $w(R_i) = 28$. Therefore, $w(L_i) \neq w(R_i)$. Therefore $w_{min} = w(L_i) = 22$ and $w_{max} = w(R_i) = 28$. Again $k_1 = 4$ and $w_1 = 3$, then $k_1w_1 = 12$. Therefore $w_{min} > k_1w_1$. Next calculate $(w_{max} - w_{min})$. Therefore $(w_{max} - w_{min}) = (28 - 22) = 6$. Therefore we can decrease the weights $(w_{max} - w_{min})$ from the nodes except the root i in the path whose weight is maximum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 6.3). Now we subtract the weight 3 from the weight of the node 4 in R_i, again we subtract the weights 1, 2 from the weights of the nodes 6, 7 respectively in R_i, then we get $w^*(R_i) = \{(6-3)+(4-1)+(5-2)+13\} = 22$. Again $w^*(L_i) = w_{min} = w(L_i) = 22$, hence we get $w^*(L_i) = w^*(R_i)$. Therefore the node 1 is the Inv1C.

Now we have the modified tree T'_{TRP} (Figure 6.4) with modified node weight.

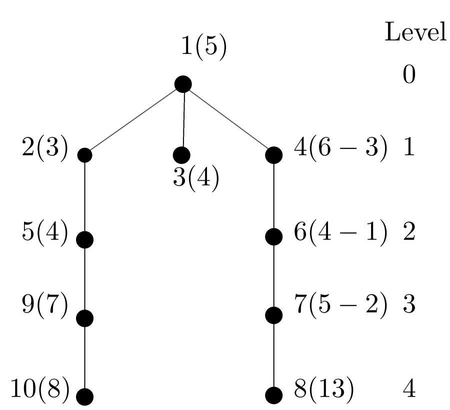

Figure 6.4: Modified tree T'_{TRP} of the tree T_{TRP}.

Next we shall prove the following important result.

Lemma 6.4.1 *The* ***Algorithm 1-INV-TRP-TREE*** *correctly computes the Inv1C of the weighted TraG.*

Proof. Let i be the pre-specified node in T_{TRP}. We have to prove that i is the Inv1C. At first, by Step 1, we have constructed the tree T_{TRP} (as per section 6.3) with root i, by Step 2, compute the longest paths R_i and L_i from i to the tree T_{TRP}, 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 node one center as well as Inv1C of T_{TRP} (Step 4.1). But if $w(L_i) \neq w(R_i)$, then modify the tree T_{TRP} under the conditions of non-linear semi-infinite (or nonlinear) optimization model (Step 4.2). By Step 4.3, modify the circular-arc tree T_{TRP} 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-TRP-TREE** correctly computes the Inv1C for any node weighted tree. □

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

Lemma 6.4.2 *The specified node i in the modified tree T'_{TRP} is the Inv1C.*

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

The following describe the total T-complexity of the algorithm to compute Inv1C problem on the weighted tree corresponding to the weighted TraG G.

Theorem 6.4.1 *The T-complexity to find Inv1C problem on a given weighted tree T'_{TRP} corresponding to the weighted TraG G is $O(n)$, where n is the number of nodes of the graph.*

Proof. Step 1 takes $O(n)$ time, since the adjacency relation of TraG 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_{TRP} is traversed in a depth-first-search 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 nodes in R_i and L_i takes $O(n)$ 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 be computed $O(n)$ time). Also, modification of weights in either R_i or L_i just takes $O(n)$ time as T_{TRP} contains n nodes and $(n-1)$ edges, so Step 4.3 can be executed in $O(n)$ time. So total T-complexity of our suggested **Algorithm 1-INV-TRP-TREE** is $O(n)$ time, where n be the number of nodes of the TraG. □

6.5 Summary

Here we investigated the Inv1C location problem with node weights on the weighted TraG G. Firstly, we develop minimum heighted tree with two branches of level difference either zero or one of the TraG. Secondly, we modified the tree maintaining the bounding conditions to get Inv1C. The T-complexity of our suggested algorithm is $O(n)$, where n is the number of nodes of the TraG G.