Chapter 5

Computation of a minimum average distance tree and inverse 1-centre location problem on permutation graph*

5.1 Introduction

Suppose $G = (V, E)$ be an UndG , where $V = \{1, 2, \ldots, n\}$, $|V| = n$ and $|E| = m$. Now, the graph G is familiar as a Per if and only if there is a permutation $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ on the set V of nodes such that $\forall\ i_1, i_2 \in V$, $(i_1, i_2) \in E$ iff

$$(i_1 - i_2)(\pi^{-1}(i_1) - \pi^{-1}(i_2)) < 0,$$

where for every $i_1 \in V$, $\pi^{-1}(i_1)$ indicates the location of the number i_1 in π [49]. In this paper, we consider a connected PerG. PerG can be displayed as a subclass of IntG [49]. Furthermore, PerG are the subclass of ComG [86]. Also, PerG can be reveled by the MchD, in which two horizontal parallel lines, named as top line and bottom line are exist.

We place the members of V on the top line, in ascending order and for each $i_1 = 1, 2, \ldots, n$ place the permutation number $\pi(i)$ on the bottom line just under the number i_1 on the top line. After then, we join two i_1's situated on the top line and on the bottom line by drawing a line segment between them, for each $i_1 = 1, 2, \ldots n$ ([49]). We also label a drawn line segment by i_1 if it is drawn by joining two i_1's. Beside these, we create each node of the

*A part of the work presented in this chapter is published in (i) *Annals of Pure and Applied Mathematics*, 2(1) (2012) 74-85, (ii) *Interciencia Journal*, 44 (2019) 118-137.

PerG for each line segment in the MchD. If two line segments i_1 and i_2 cuts each other in the MchD then $(i_1, i_2) \in E$. The converse is true also.

Figure 5.1 represents the PerG G and Figure 5.2 is the corresponding MchD of the PerG G.

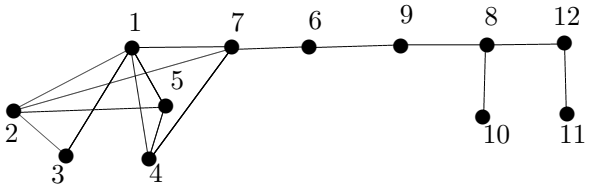

Figure 5.1: A PerG G.

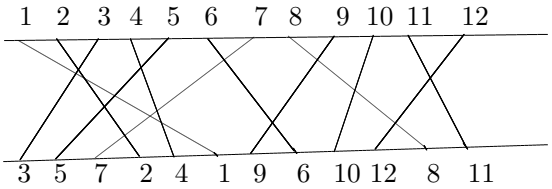

Figure 5.2: MchD of the PerG G of Figure 5.1.

In this chapter, we have designed
(i) an $O(n^2)$ time algorithm to build a MADT for a given PerG G, and
(ii) an $O(n)$ time algorithm to measure Inv1C location problem on weighted PerG where n is the number of nodes of the graph.

5.2 Organization of the chapter

Section 5.3 and its subsections contain the method of the construction of the tree, the MdsT, modified tree of the tree T, an algorithm to get MADT of the PerG and the T-complexity of the unweighted PerG. In Section 5.4 and its subsections presents about weighted PerG, its corresponding MchD, construction of the w-tree, BFS tree, MdsT of the corresponding weighted PerG, an algorithm and its T-complexity for Inv1C of the weighted PerG. In Section 5.5, we give the summary of the chapter.

5.3 MADT for permutation graph

5.3.1 Construction of BFS tree on permutation graph

It is common fact that BFS is a very important graph traversal technique and BFS also constructs a BFS tree. In BFS if we started with node v, then we first scan all edges incident on v and then shift to an adjacent node w. At w we then study all edges incident to w and shift to a node which is adjacent to w. This process is ongoing till all the edges in the graph are scanned [32].

BFS tree can be established on general graphs in $O(x+y)$ time, where x and y represent the number of nodes and number of edges respectively [96]. For construction of this BFS tree on a PerG, we must consider the MchD. At first the line segment x_1 is placed on the zero level and all nodes adjacent to x_1 is placed on the first level. Then we travel through the MchD step by step. At first step we first scan the untraversed line segment which are placed at the right side of line segments x_1 and in the next step we scan all untraversed line segments which are placed at the left side of the line segment x_1. In each iteration, we scan the numbers within an interval on the top channel and then scan permutation number within an interval on the bottom channel alternatively in an order. Mondal et al. [71] recently have structured an algorithm for construction a BFS tree $T^*(i)$ with root as i on TraG in $O(n)$ time, where n is the number of vertices and Barman et al. have constructed another algorithm to construct the BFS tree on a PerG with any vertex x as root in $O(n)$ time [8]. Figure 5.3 is the BFS tree which is constructed by the **Algorithm PBFS** [8].

5.3.2 Computation of minimum diameter spanning tree

Let $T(1)$, $T(\pi(1))$ be two BFS trees constructed by **Algorithm PBFS** [8] of a PerG G. Let T be the least height tree between $T(1)$ and $T(\pi(1))$.

Suppose P be the main path (longest path) of T of the PerG G and it is denoted by $u_0^* \to u_1^* \to u_2^* \to \cdots \to u_k^*$, where $k \leq (n-1)$ and u_0^* is either the node 1 or the node corresponding to $\pi(1)$ of G. The nodes of the path P, i.e. u_i^*, $i = 0, 1, 2, 3, \ldots, k$ specified as *internal nodes*.

We have the following terms and conditions:

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

After that we define the *level* of the vertex u as the distance of u from the root i of the

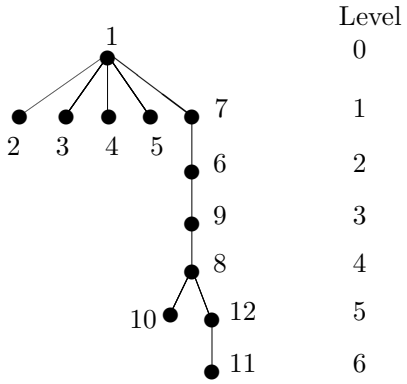

Figure 5.3: BFS tree T rooted at vertex 1 of the PerG G shown in Figure 5.1.

BFS tree $T(i)$ and indicate it by *level(u)*, $u \in V$ and take the level of the root i as 0. The level of each node on BFS tree $T(i), i \in V$ can be allocate by the BFS algorithm of Chen and Das [21]. The level of each vertex of the tree T can be calculated in $O(n)$ time.

According to the construction of BFS tree by the algorithm **PBFS** [8], we must have to pay attention about the result in BFS tree T.

In PerD, there are two intersecting line segments of *V-type* or *inverted V-type* defined as the *scissors type line segments* of PerG. Figure 5.4 gives the illustration of scissors type line segments.

Lemma 5.3.1 *Least number of scissors type line segments with maximum spread of the PerD covering the whole region.*

Proof. Let $\{1, 2, \ldots, n\}$ be the set of numbers and $\{\pi(1), \pi(2), \ldots, \pi(n)\}$ be the permutation. Covering from 1 to n, i.e. whole region, there are two types of scissors, indicated as dotted lines (V-type) and continued lines (inverted V-type), which is shown in the Figure 5.4 and Figure 5.5.

At first, if we select first dotted line segment $(1, \pi^{-1}(1))$ on the top line, then we consider such line segment $(i, \pi^{-1}(i))$, which is maximum spread among all adjacent to the corresponding first line segment 1 and we marked all the line segment adjacent to 1. After that, we must consider such line segment $(j, \pi^{-1}(j))$ which is maximum spread among all unmarked line segment adjacent to i, continuing this process until all the line segments are covered, i.e. marked. Assuming this path be of the type $top \to bottom \to top \to bottom \to \ldots$.

Similar idea must be followed from the bottom line. After following this manner, we obtain the another path of the form $bottom \to top \to bottom \to top \to \ldots$. In this way entire region is covered by the both paths.

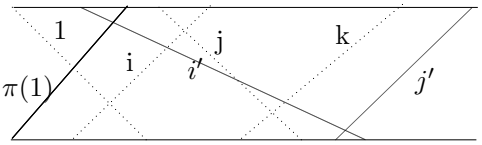

Figure 5.4: An Illustration.

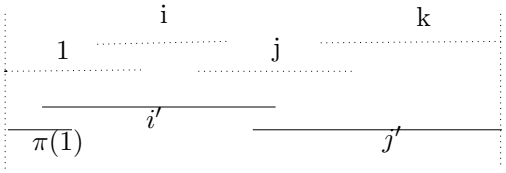

Figure 5.5: Region covered by the projection of minimum number of line segments.

Next, we would like to consider minimum path between them which will be the shortest path to covered all the line segments. Therefore, the scissors type lines segment of the PerG covered the entire region. □

Lemma 5.3.2 *[8] Two crossing line segments of the PerG of the MchD are allowed on the same level or adjacent levels.*

Lemma 5.3.3 *If $x, y \in V$ and $|level(x) - level(y)| > 1$ in T, then an edge does not exist between the nodes x and y in G.*

Proof. Let $|level(x) - level(y)| > 1$ and $(x, y) \in E$, so by Lemma 5.2.2 either $level(x) = level(y)$ or $|level(x) - level(y)| = 1$ which is contradict to $|level(x) - level(y)| > 1$. Hence $(x, y) \notin E$, i.e. the edge does not exist between the nodes x and y. □

The T-complexity of the algorithm PBFS is given below.

Theorem 5.3.1 *[8] BFS tree can be completed in $O(n)$ time for a PerG containing n nodes.*

So this BFS tree is a spanning tree.

Next we prove that this spanning tree is a MdsT..

Lemma 5.3.4 *The tree $\boldsymbol{T}$ is a MdsT.*

Proof. As stated by the erection the BFS tree, the longest path of the tree T is the greatest path, i.e. diameter of T. The longest path consists least number of scissors type line segments (by Lemma 5.2.1). This diameter is least, as T is the least heighted tree among $T(1), T(\pi(1))$. Again, T is a spanning tree. Hence T is a MdsT. □

In the next subsection we modify the tree T.

5.3.3 Modification of the spanning tree T

It is noticed that T is not necessarily a MADT. So, modification is necessary. By the following way we modify the tree T:

We first work out $N(x_i^*) \in G$ for each internal node $x_i^* \in T$. If there are any familiar neighbouring nodes of two successive internal nodes of P, i.e. $N(x_i^*) \cap N(x_{i+1}^*) \neq \phi$ then by the following way we can move them.

i) In G, if w be the familiar neighbouring node of x_i^* and x_{i+1}^*, i.e. $N(x_i^*) \cap N(x_{i+1}^*) \neq \phi$, then we count the number of nodes to the upper side (if exist) and lower side of the internal node x_i^* in T with x_i^* as origin. Next count their difference, say, d_1.

ii) Again, count the number of nodes to the upper part and lower part (if exist) of the internal node x_{i+1}^* in T with x_{i+1}^* as origin (ignoring the node y). Next count their difference, say, d_2.

iii) If $d_1 - d_2 \leq 0$, then unchanged, i.e. w is finally adjacent of x_i^* in T and if $d_1 - d_2 > 0$, then the neighbouring node w of the node x_i^* is shifted to the node x_{i+1}^*, i.e. y is finally adjacent of x_{i+1}^* in T.

Same concept is used for pair of any two successive internal nodes on the main path P. Under these conditions we establish the spanning tree T and it is denoted by $T^{'}$. Figure 5.6 is the modified BFS tree $T^{'}$ of the BFS tree T shown in Figure 5.3.

Lemma 5.3.5 *If $T^{'}$ is a BFS tree, then $d_{T'}(x, y)$ is the distance between the nodes x and y in T' is given by*

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

Proof. Let x is the root of $T^{'}$. There exists a shortest path $x \to z_1 \to z_2 \cdots \to z_{p-1} \to y$ from x to any node $y \in G$, with x as parent of z_1 and z_i as parent of z_{i+1} and so on for each $i = 1, 2, \ldots, p-2$ and z_{p-1} as parent of y. Since every node of this path is directly linked with the next one, hence the length of this path is $p = level(y)$. Thus $d(x, y) \leq p$.

Now we show that $d(x, y) \not< p$. If possible, let $d(x, y) = q < p$. Then there is a path $x \to y_1 \to y_2 \cdots \to y_{q-1} \to y$ from x to any node $y \in G$. As every node of is directly linked with the next one using Lemma 5.3.2, $level(y_1)$ is either 0 or 1 since $level(x) = 0$. By same Lemma 5.3.2, $level(y_{k+1})$ is either $level(y_k)$ or $level(y_k) + 1$ or $level(y_k) - 1$. Thus $level(y_2)$ is 0 or 1 or 2, $level(y_3)$ is 0 or 1 or 2 or 3 and so on. Therefore $level(y)$ is 0 or 1 or 2 or...or q. So contradiction arises since $level(y) = p$ and $p > q$. Hence $d(x, y) \not\leq p$ which implies that

$d(x, y) = p$, i.e. $d(x, y) = level(y)$.

If $x = x^*$, i.e. the internal vertex, then as per rule of construction of BFS, there is a shortest path $x \to z_1^{'} \to z_2^{'} \cdots \to y$. Here $z_1^{'}$ is at next level of x^*, $z_2^{'}$ is at the next level of $z_1^{'}$ and so on up to v. So $d(x^*, z_1^{'}) = 1 = (i+1)-i$, $d(x^*, z_2^{'}) = d(x^*, z_1^{'})+d(z_1^{'}, z_2^{'}) = 1+1 = 2 = (i+2)-i$. If $d(x^*, z_k^{'}) = k = (i+k)-i$, then $d(x^*, z_{k+1}^{'}) = d(x^*, z_k^{'})+d(z_k^{'}, z_{k+1}^{'}) = k+1 = (i+k+1)-i$. When x is any leaf, then there exist a path from x to y via the parent of x. If $level(x) = i$, then $level(parent(x)) = i-1$ and $parent(x)$ is an internal node (as per construction of BFS rooted as 1 or $\pi(1)$). Therefore $d(x, y) = d(x, parent(x)) + d(parent(x), y) = 1 + level(y) - level(parent(x))$. □

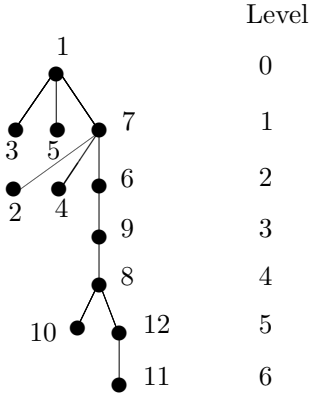

Figure 5.6: Modified BFS tree T' of the tree T shown in Figure 5.3.

5.3.4 Algorithm and its complexity

In this section, we establish the algorithm to build MADT for a given PerG. Also, the T-complexity are introduced here.

Algorithm PMAD-TREE

Input: A PerG $G = (V, E)$ with its permutation depiction $i, \pi(i); i = 1, 2, \ldots, n$.

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

Step 1: Construct the MdsT tree T //Section 5.3.2//
and count $level(x) = d(x^*, x), x^* = 1 \,or\, \pi(1)$.

Step 2: Complete $N(x_i^*) \in G$ for all $x_i^* \in T$, and r = height of T = highest level.

Step 3: //Modification of the tree T//
Complete familiar nodes, if any, of two successive internal vertices x_i^* and $x_{i+1}^*, i = 0, 1, 2, \ldots, r-1$ of the main path P may be moved

as leaves to the vertex x^*_{i+1} under the following rules otherwise remains unchanged.

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

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

If $N(x^*_i) \cap N(x^*_{i+1}) \neq \phi$ and let $y \in N(x^*_i) \cap N(x^*_{i+1})$ then,

Step 3.1.1: Count the number of nodes to the upper part and lower part of the internal node x^*_i in T with x^*_i as origin. Next count their difference, d_1.

Step 3.1.2: Count the number of nodes to the upper part and lower part of the internal node x^*_{i+1} with it as origin (neglecting the vertex y). Next count their difference, d_2.

Step 3.1.3: If $d_1 - d_2 \leq 0$, then unchanged and if $d_1 - d_2 > 0$, then the adjacent node y of the internal node x^*_i is shifted to the internal node x^*_{i+1}.

Step 3.2: Set T' as adjusted spanning tree of T on PerG G.

Step 4: Calculate

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

and $\mu(T') = \frac{2}{n(n-1)} \sum_{\{x,y\} \subset V(T')} d_{T'}(x,y)$.

end PMAD-TREE.

5.3.5 Illustration of the algorithm

In Figure 5.3, which is the spanning tree of the PerG G, $x^*_i = 1$ and $x^*_{i+1} = 7$ are two successive nodes. 2 and 4 are the familiar neighbouring of the nodes $x^*_i = 1$ and $x^*_{i+1} = 7$, i.e. $y \in \{2, 4\}$. Taking the node 1 as origin, the number of nodes to the upper side of 1 is 0 and the number of nodes to the lower side of 1 is 7. Hence their difference is $d_1 = 7 - 0 = 7$.

Again taking the node 7 as origin, the number of nodes to the upper side of 7 is 3 (ignoring the nodes 2 and 4) and the number of nodes to the lower side of 7 is 6 when the nodes 2 and 4 are neighbouring with the node 7. Hence their difference is $d_2 = 6 - 3 = 3$. Therefore $d_1 > d_2$. So, the nodes 2 and 4 are moved to the node 7. Then we have the modified spanning tree T'(Figure 5.6).

Next count the average distance $\mu_1(G)$ correlative to the tree T. Again count average distance $\mu_2(G)$ correlative to the tree T'. Here $\mu_1(G) = 224/66 = 3.39$(approx.) and

$\mu_2(G) = 216/66 = 3.27$(approx.).

Clearly, $\mu_1(G) > \mu_2(G)$. Hence $T^{'}$ is the MADT of the PerG G.

Theorem 5.3.2 *The tree formed by* ***Algorithm PMAD-TREE*** *is the MADT.*

Proof. Firstly, we have build the BFS tree T. By nature and method of construction of BFS tree it gives least diameter (Lemma 5.3.4), i.e. there also exist a shortest path by which every other nodes are directly linked with any internal vertices. So, this BFS tree gives the assurance that there is no other least path to link all other nodes directly with the internal vertices. Again, in BFS, the leaves in level difference two or more, are not neighbouring so they may be linked via their parent node, i.e. internal nodes (Lemma 5.3.2 and Lemma 5.3.3). So for getting the minimum average distance, one leaf may be moved to next level, i.e. connected with next internal vertex. Therefore, the sum of the distances among all pair nodes through the main path is least (Lemma 5.3.4). Again by Step 3, we examine the distances among the existing leaves of T. The necessary condition for shifting the leaves to other, is the sum of the distances among the leaves, after modification, is least. Hence total distance among the nodes in T is minimum. Therefore, the tree composed by **Algorithm PMAD-TREE** is a MADT. □

Next we describe the time complexity of the algorithm.

Theorem 5.3.3 *The MADT of a PerG G with n nodes, by* ***Algorithm PMAD-TREE****, can be completed in $O(n^2)$ time.*

Proof. Step 1 takes $O(n)$ time (Theorem 5.3.1). Step 2, i.e. estimation of open neighbourhood of all internal vertices on the main path and add with correlated vertices can be calculated in $O(n^2)$ time. Each Step 3.1.1 and Step 3.1.2 takes $O(n)$ time. Also Step 3.1.3 takes in $O(1)$ time. But Step 3.1 runs $(r-1)$ times, so total T-complexity of Step 3.1 is $O(n^2)$, where r is of $O(n)$. Again Step 3.2 takes $O(n^2)$ time. Step 4 can be completed in $O(n^2)$ time. Hence total T-complexity of our suggested algorithm is $O(n^2)$ time with n nodes of the PerG. □

In the next section we consider another problem on weighted permutation graphs.

5.4 Inverse 1-centre location problem on weighted permutation graphs

Figure 5.7 perform the weighted PerG G and Figure 5.8 is the correlative MchD of that PerG G.

5.4.1 Construction of BFS tree on weighted permutation graph

In graph theory there are several graph traversal technique exist. For instances Breadth first search, Depth first search, Eulerian tour, Hamiltonian tour, etc. BFS is very popular to the researcher for its simplification and it constructs a BFS tree can be formed on general graphs in $O(n+m)$ time, where n and m are respectively the cardinality of node set and cardinality of edge set [96].

Recently Barman et al. have designed an $O(n)$ time algorithm to form a BFS tree on a PerG [8]. In this chapter we construct a BFS tree T_{PER}, root at 1 by the **Algorithm PBFS** [8]. The Figure 5.9 represents the BFS tree.

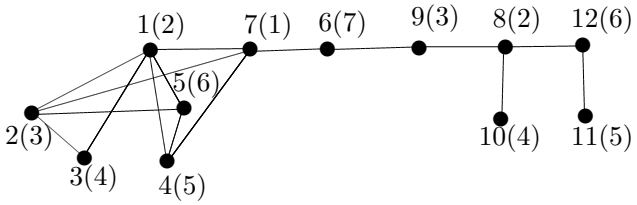

Figure 5.7: A PerG G.

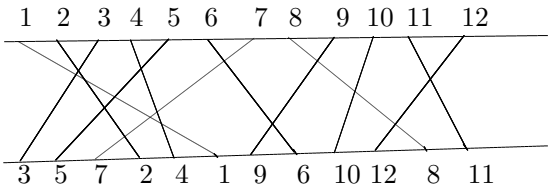

Figure 5.8: MchD of the PerG G of Figure 5.7.

5.4.2 Computation of minimum diameter spanning tree

Let $T(1)$, $T(\pi(1))$ be two BFS trees constructed by **Algorithm PBFS** [8] of a PerG G. Let T be the least heighted tree between $T(1)$ and $T(\pi(1))$.

Suppose P be the main path (longest path) of T of the PerG G and it is denoted by $u_0^* \to u_1^* \to u_2^* \to \cdots \to u_k^*$, where $k \leq (n-1)$ and u_0^* is either the node 1 or the node

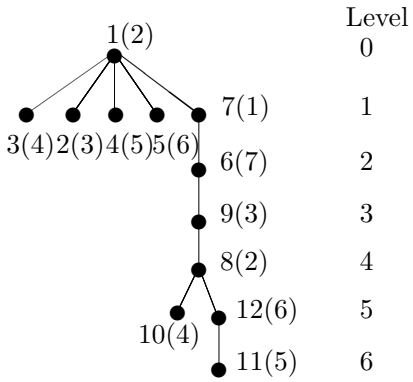

Figure 5.9: BFS tree T_{PER} rooted at vertex 1 of the PerG G shown in Figure 5.7.

corresponding to $\pi(1)$ of G. The nodes of the path P, i.e. u_i^*, $i = 0, 1, 2, 3, \ldots, k$ specified as *internal nodes*.

We have the following terms and conditions:

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

After that we define the *level* of the vertex u as the distance of u from the root i of the BFS tree $T(i)$ and indicate it by *level(u)*, $u \in V$ and take the level of the root i as 0. The level of each node on BFS tree $T(i), i \in V$ can be allocate by the BFS algorithm of Chen and Das [21]. The level of each vertex of the tree T can be calculated in $O(n)$ time.

According to the construction of BFS tree by the algorithm **PBFS** [8], we must have to pay attention about the result in BFS tree T.

In PerD, there are two intersecting line segments of *V-type* or *inverted V-type* defined as the *scissors type line segments* of PerG. Figure 5.4 gives the illustration of scissors type line segments.

Lemma 5.4.1 *Minimum number of scissors type line segments with maximum spread of the PerD cover the whole region.*

Proof. Let $\{1, 2, \ldots, n\}$ be the set of numbers and $\{\pi(1), \pi(2), \ldots, \pi(n)\}$ be the permutation. To cover 1 to n, i.e. whole region, there are two types of scissors, marked as dotted lines (V-type) and continued lines (inverted V-type), shown in the Figure 5.4 and Figure 5.5.

Firstly, if we select first dotted line segment $(1, \pi^{-1}(1))$ on top line. Then we consider such line segment $(i, \pi^{-1}(i))$, which is maximum spread among all adjacent to the corresponding first line segment 1 and we marked all the line segment adjacent to 1. Next, we consider such line segment $(j, \pi^{-1}(j))$ which is maximum spread among all unmarked line segment

adjacent to i, continuing this process until all the line segments are covered, i.e. marked. Let this path be of the type $top \to bottom \to top \to bottom \to \ldots$.

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

Next we consider minimum path between them which will be the shortest path to covered all the line segments. Hence, the scissors type line segments of the PerG covered the whole region. □

Lemma 5.4.2 *[8] Two intersecting line segments of the PerG of the MchD are assigned on the same level or adjacent levels.*

Lemma 5.4.3 *If $u_1, v_1 \in V$ and $|level(u_1) - level(v_1)| > 1$ in T_{PER}, then there is no edge between the nodes u_1 and v_1 in G.*

Proof. If possible let $|level(u_1) - level(v_1)| > 1$ but $(u_1, v_1) \in E$, i.e. u_1 and v_1 are directly connected. Since u_1 and v_1 are directly connected so by Lemma 5.4.2 either $level(u_1) = level(v_1)$ or $|level(u_1) - level(v_1)| = 1$ which is contradictory to the assumption that $|level(u_1) - level(v_1)| > 1$. Hence $(u_1, v_1) \notin E$, i.e. u_1 and v_1 are not directly connected in G. □

The T-complexity of the algorithm PBFS is stated below.

Theorem 5.4.1 *[8] The BFS tree rooted at any vertex $x \in V$ can be computed in $O(n)$ time for a PerG containing n vertices.*

Obviously, this BFS tree is a spanning tree.

Now, we shall prove that this spanning tree is also a MdsT.

Lemma 5.4.4 *The spanning tree T_{PER} is a MdsT.*

Proof. Using the construction the BFS tree, the main path of the tree T_{PER} is the greatest path which is the diameter of T_{PER}. The main path contains minimum number of scissors type line segments (by Lemma 5.4.1). But this diameter is minimum, because T_{PER} is the minimum heighted tree between $T(1), T(\pi(1))$. Also, T_{PER} is a spanning tree. Hence T_{PER} is a MdsT. □

Now, before going to our suggested algorithm we introduce some notations for our algorithmic purpose. Let i be the pre-specified node in G.

R_i	:	Longest path right to the node i.
L_i	:	Longest path left 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 the graph G.
$w_{upp}(v)$	:	Maximum weight of the node in the graph G.
w_{min}	:	$min\{w(L_i), w(R_i)\}$.
w_{max}	:	$max\{w(L_i), w(R_i)\}$.
w_1	:	$min\{w(v), v \in G\}$.
w_2	:	$max\{w(v), v \in G\}$.
k_1	:	The number of 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_{PER}	:	Weighted tree corresponding to the PerG G.
T'_{PER}	:	Modified tree of the tree T_{PER} corresponding to the PerG 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 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, node i is not Inv1C of the weighted tree for the PerG.

Case II: If number of adjacent nodes to the node i are more than one, i.e. $deg(i) > 1$, then we can construct a tree with root i and two branches. To get Inv1C following six possibilities arises:

1. If the total weight of one side of the node i is equal to 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 equal to 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 greater 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 equal to 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 greater 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_{PER} with the help of following non-linear semi-infinite (or nonlinear) optimization model:

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

subject to

$\max_{v_1 \in V(T_{CIR})} d_{\overline{w}}(v_1, i) \leq \max_{v_1 \in V(T_{PER})} d_{\overline{w}}(v_1, q)$, for all $q \in T_{PER}$(or $p \in V(T_{PER})$),

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

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

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

$x_1\{w(v_1)\}, y_1\{w(v_1)\} \geq 0$, for all $v_1 \in V(T_{PER})$,

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 $x_1 = \{x_1(w(v_1)) : v_1 \in V(T_{PER})\}$ and $y_1 = \{y_1(w(v_1)) : v_1 \in V(T_{PER})\}$ is also called a feasible modification of the Inv1C location problem.

Now, we prove the following results.

Lemma 5.4.5 *If $w_{min} = k_1 w_1$ in T_{PER}, 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 is 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_{PER} 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_{PER}, say T'_{PER}. Again, since the PerG is an arbitrary, so our assumption is true for any PerG.

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

Lemma 5.4.6 *If $w_{min} > k_1 w_1$ in T_{PER}, 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_{PER}, 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 (Subsection 5.4.2). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{PER}, say T'_{PER}. Again, since the PerG is an arbitrary, so our assumption is true for any PerG.

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

Lemma 5.4.7 *If $w_{min} < k_1w_1$ in T_{PER}, 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_{PER}, 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_1w_1. Again we have $w_{min} < k_1w_1$. Therefore we can increase the weights $(k_1w_1 - w_{min})$ 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 (Subsection 5.4.2). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{PER}, say T'_{PER}. Again, since the PerG is an arbitrary, so our assumption is true for any PerG.

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

Lemma 5.4.8 *If $w_{max} = k_2w_2$ in T_{PER}, 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 is 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_{PER} 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_2w_2 weight and hence enhanced weight of the path L_i or R_i becomes k_2w_2. Again we have $w_{max} = k_2w_2$. By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{PER}, say T'_{PER}. Again, since the PerG is an arbitrary, so our assumption is true for any PerG.

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

Lemma 5.4.9 *If $w_{max} > k_2w_2$ in T_{PER}, 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 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_{PER}, 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 (Subsection 5.4.2). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{PER}, say T'_{PER}. Again, since the PerG is an arbitrary, so our assumption is true for any PerG.

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

Lemma 5.4.10 *If $w_{max} < k_2w_2$ in T_{PER}, 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_{PER}, 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 (Subsection 5.4.2). By this way we can balance the weights of both paths. So we get the modified tree of the tree T_{PER}, say T'_{PER}. Again, since the PerG is an arbitrary, so our assumption is true for any PerG.

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

5.4.3 Algorithm and its complexity

In this section we choose an algorithm for the Inv1C location problem on the vertex weighted tree T_{PER}. The main idea of our suggested algorithm is as follows:

Let T_{PER} be a weighted tree corresponding to the PerG G with n vertices and $(n-1)$ edges. Let V be the vertex set and E be the edge set. Let i be any non-pendant specified vertex in the tree T_{PER} which is to be Inv1C. At first we calculate the path whose weight is maximum from i to any pendant vertex of T_{PER}. Let L_i and R_i 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 sum of weights of the vertices except the root of the paths L_i, R_i respectively with

respect to the vertex i. If $w(L_i) = w(R_i)$, then i is the center as well as the Inv1C of the graph. If $w(L_i) \neq w(R_i)$, then six cases may arise. In the first case, if $w_{min} = k_1 w_1$ in T_{PER}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of vertices in such path between L_i, R_i whose weight is maximum, except the root i and $w_{min} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices up to minimum weight except the vertex i, i.e.,root i maintaining the bounding condition (Subsection 5.4.2) in the path whose weight is maximum and i is the Inv1C. In the second case, if $w_{min} > k_1 w_1$ in T_{CIR}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of vertices in such path between L_i, R_i whose weight is maximum, except the root i and $w_{min} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of some vertices except the root i maintaining the bounding conditions (Section 5.4.2) in the path whose weight is maximum and i is the Inv1C. In third case, if $w_{min} < k_1 w_1$ in T_{CIR}, where $w_1 = min\{w(v), v \in G\}$, $w_{min} = min\{w(L_i), w(R_i)\}$, k_1 be the number of vertices in such path between L_i, R_i whose weight is maximum, except the root i and $w_{min} > 0$, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices up to minimum weight except the root i maintaining the bounding conditions (Subsection 5.4.2) in the path whose weight is maximum and enhance the weights of some vertices except the root i in the path whose weight is minimum and i is the Inv1C. In fourth case, if $w_{max} = k_2 w_2$ in T_{PER}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of vertices in such path between L_i, R_i whose weight is minimum, except the root i and $w_{max} > 0$, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding conditions (Subsection 5.4.2) in the path whose weight is minimum and i is the Inv1C. In fifth case, if $w_{max} > k_2 w_2$ in T_{PER}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of vertices in such path between L_i, R_i whose weight is minimum, except the root i and $w_{max} > 0$, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding conditions (Section 5.4.2) in path whose weight is minimum and reducing the weights of some vertices except the root i in the path whose weight is maximum and i is the Inv1C. In sixth case, if $w_{max} < k_2 w_2$ in T_{CIR}, where $w_2 = max\{w(v), v \in G\}$, $w_{max} = max\{w(L_i), w(R_i)\}$, k_2 be the number of vertices in such path between L_i, R_i whose weight is minimum, except the root i and $w_{max} > 0$, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of some vertices except the root i maintaining the bounding conditions (Subsection 5.4.2) 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 PerG G is as

follows:

Algorithm 1-INV-PER-TREE

Input: Weighted PerG G with weights w_j to the each vertex v_j, $(j = 1, 2 \ldots, n)$.

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

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

Step 2. Compute the longest paths R_i and L_i from i to the tree T_{PER}.

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

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

Step 4.1. If $w(L_i) = w(R_i)$, then i is the vertex one center as well as Inv1C of T_{PER}.

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_{PER}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices except the vertex i, i.e.,root i up to minimum weight maintaining the bounding condition in the path whose weight is maximum, then go to Step 4.3.

Step 4.2.2. If $w_{min} > k_1 w_1$ in T_{PER}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of some vertices except the root i maintaining the bounding condition in the path whose weight is maximum, then go to Step 4.3.

Step 4.2.3. If $w_{min} < k_1 w_1$ in T_{PER}, then $w^*(L_i) = w^*(R_i)$ by reducing the weights of all vertices except the root i up to minimum weight maintaining the bounding condition in the path whose weight is maximum and enhance the weights of some vertices except the root i in the path whose weight is minimum, then go to Step 4.3.

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

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

Step 4.2.6. If $w_{max} < k_2 w_2$ in T_{PER}, then $w^*(L_i) = w^*(R_i)$ by enhance

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

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

end 1-INV-PER-TREE.

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

Illustration of the Algorithm 1-INV-PER-TREE to the tree T_{PER} in Figure 5.9**:** Let $i = 1$ be the pre-specified vertex of the tree T_{PER} which is to be Inv1C. Next we find the longest left path L_i from the vertex 1 to other vertex 3, i.e. the path $1 \to 3$ and find longest right path R_i from 1 to the vertex 11 does not contain any vertex of the path L_i except 1, i.e. the path $1 \to 7 \to 6 \to 9 \to 8 \to 12 \to 11$. Next calculate the weights of the paths L_i and R_i. Let $w(L_i)$ and $w(R_i)$ are the total weights of the vertices except the root $i = 1$ of L_i and R_i respectively. Here $w(L_i) = 4$ and $w(R_i) = 24$. Therefore $w_{min} = w(L_i) = 4$ and $w_{max} = w(R_i) = 24$. Again $k_1 = 6$ and $w_1 = 1$, then $k_1 w_1 = 6$. Therefore $w_{min} < k_1 w_1$. Therefore we reduce the weights of each vertex in R_i except the root $i = 1$ up to minimum weight 1 maintaining the bounding conditions (Subsection 5.4.2) and increase the weight 2 to the vertex 3 in L_i, then we get $w^*(R_i) = \{1+(7-6)+(3-2)+(2-1)+(6-5)+(5-4)\} = 6$. Again $w^*(L_i) = 4+2 = 6$, hence we get $w^*(L_i) = w^*(R_i)$. Therefore the vertex 1 is the Inv1C.

Now we have the modified tree T'_{PER} (Figure 5.10) with modified vertex weight.

Next we shall prove the following important result.

Lemma 5.4.11 *The **Algorithm 1-INV-PER-TREE** correctly computes the Inv1C of the weighted PerG.*

Proof. Let i be the pre-specified vertex in T_{PER}. We have to prove that i is the Inv1C. At first, by Step 1, we have constructed the tree T_{PER} (as per Subsection 5.4.1) with root i, by Step 2, compute the longest paths R_i and L_i from i to the tree T_{PER}, by Step 3, calculate the weight of the paths L_i and R_i from i except i, i.e. $w(L_i)$ and $w(R_i)$. In Step 4, If $w(L_i) = w(R_i)$, then i is the vertex one center as well as Inv1C of T_{PER} (Step 4.1). But if $w(L_i) \neq w(R_i)$, then modify the tree T_{PER} under the conditions of non-linear semi-infinite (or nonlinear) optimization model (Step 4.2). By Step 4.3, modify the tree T_{PER} 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

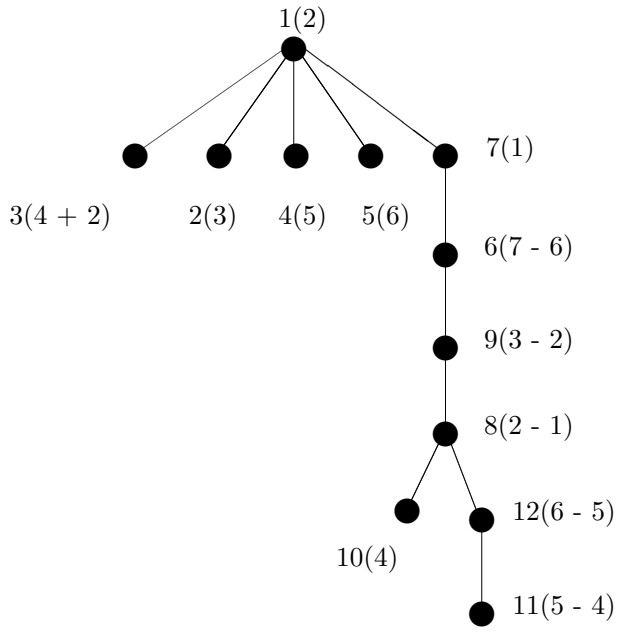

Figure 5.10: Modified tree T'_{PER} of the tree T_{PER}.

the Inv1C. Hence **Algorithm 1-INV-PER-TREE** correctly computes the Inv1C for any w-tree. □

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

Lemma 5.4.12 *The specified vertex i in the modified tree T'_{PER} is the Inv1C.*

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

The following describe the total T-complexity of the algorithm to compute Inv1C problem on the w-tree for the weighted PerG G.

Theorem 5.4.2 *The T-complexity to find Inv1C problem on a given w-tree T'_{PER} for the weighted PerG 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 PerG 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_{PER} is traversed in a depth-first-search manner. Step 3 takes $O(n)$ time to compute the total

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_{PER} contains n nodes and $(n-1)$ edges, so Step 4.3 can be executed in $O(n)$ time. Hence total T-complexity of our suggested **Algorithm 1-INV-PER-TREE** is $O(n)$ time, where n is the number of nodes of the PerG. □

5.5 Summary

In this chapter, we have arranged an $O(n^2)$ time algorithm to construct a MADT for a given PerG G with n nodes. We design the communication networks by the help of this problem.

Also, we investigated the Inv1C location problem with node weights on the tree corresponding to the weighted PerG G. Firstly, we develop minimum heighted tree with two branches of level difference either zero or one of the PerG. 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 PerG G. This idea can be applied to solve the 1-center location problem to other graphs.