Computation of inverse 1-center location problem on the weighted Permutation Graphs

Biswanath Jana[†], Sukumar Mondal[‡] and Madhumangal Pal[†]
[†]Department of Applied Mathematics with Oceanology and Computer Programming, Vidyasagar University, Midnapore – 721102, India.
[‡]Department of Mathematics, Raja N. L. Khan Women's College, Gope Palace, Paschim Medinipur – 721102, India.
E-mail:biswanathjana2012@gmail.com, sm5971@rediffmail.com, mmpalvu@gmail.com

Abstract

Let T_{PER} be the tree corresponding to the weighted permutation graph $G = (V, E)$. The eccentricity $e(v)$ of the vertex v is defined as the sum of the weights of the vertices from v to the vertex farthest from $v \in T_{PER}$. A vertex with minimum eccentricity in the tree T_{PER} is called the 1-center *of that tree. In an inverse 1-center location problem the parameter of the tree T_{PER} corresponding to the weighted permutation graph $G = (V, E)$, like vertex weights have to be modified at minimum total cost such that a pre-specified vertex $s \in V$ becomes the 1-center of the permutation graph G. In this paper, we present an optimal algorithm to find an inverse 1-center location on the tree T_{PER} corresponding to the permutation graph $G = (V, E)$, where the vertex weights can be changed within certain bounds. The time complexity of our proposed algorithm is $O(n)$, where n is the number of vertices of the weighted permutation graph G.*

Keywords: *Tree-networks, center location, 1-center location, inverse 1-center location, inverse optimization, tree, permutation graph.*

1 Introduction

An undirected graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$ is said to be a *permutation graph* iff there exists a permutation $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ on $\{1, 2, \ldots, n\}$ such that for all $i, j \in V$, $(i, j) \in E$ iff

$$(i - j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0,$$

where for each $i \in V$, $\pi^{-1}(i)$ denotes the position of the number i in π [22]. We assume that the graph is connected. Permutation graphs can be visualized as a class of intersection graphs [22]. Pnueli et al. [43] showed that permutation graphs are exactly those comparability graphs (transitively orientable graphs) whose complements are also comparability graphs. If the permutation graph is given in the form of an adjacent matrix or an adjacent list, we can construct the permutation representation (diagram) in $O(n^2)$

time [22, 40]. Henceforth, we assume that a permutation representation is given for the input graph. We assume that the permutation is stored in the array $\pi(i), i = 1, 2, \ldots, n$ and the inverse permutation of π is stored in array $\pi^{-1}(i), i = 1, 2, \ldots, n$. The array $\pi^{-1}(i)$ can be computed in $O(n)$ time from the array $\pi(i)$. Figure 1 represents the permutation graph G and Figure 2 is the corresponding matching diagram of the permutation graph G.

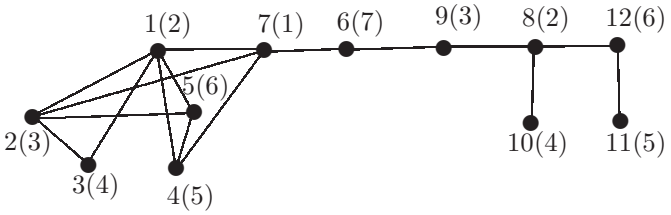

Figure 1: A permutation graph G.

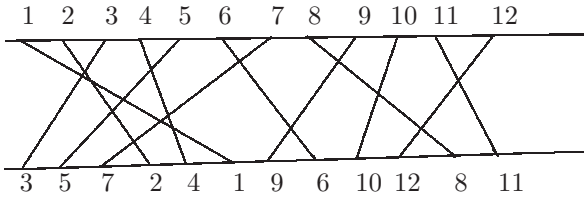

Figure 2: Matching diagram of the permutation graph G of Figure 1.

Permutation graphs is a class of perfect graphs and can be used to solve many real world problems like the problem of airline routes connection with various cities such that all scheduled to be utilized at the same time of the day, the problem of shifted intervals etc.

Breadth-first-search (BFS) is a strategy for searching in a graph when search is limited to essential two operations: (a) visit and inspect a node of a graph; (b) gain access to visit the nodes that neighbour the currently visited node. The BFS begins at a root node and inspect all the neighbouring nodes. Then for each of those neighbour nodes in turn, it inspects their neighbour nodes which were unvisited, and so on.

BFS is a uniformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. In other words, it exhaustively searches the entire graph or sequence without considering the goal until it finds it.

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

In an un-weighted tree $T = (V, E^{'})$, where $|E^{'}| = |V| - 1$, the *eccentricity e(v)* of the vertex v is defined as the distance from v to the vertex farthest from $v \in T$, i.e.,

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

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

In a weighted tree $T = (V, E^{'})$, where $|E^{'}| = |V| - 1$, the *eccentricity e(v)* of the vertex v is defined as sum of the weights of the vertices from v to the vertex farthest from $v \in T$, i.e.,

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

where $d(v, v_i)$ is the sum of the weights of the vertices on the shortest path between v and v_i.

A vertex with minimum eccentricity in the tree T is called a *center* of that tree T, i.e., if $e(s) = \min\{e(v), \text{for all } v \in V\}$, then s is the *1-center*. It is clear that every tree has either one or two centers.

The eccentricity of a center in a tree is defined as the *radius* of the tree and is denoted by $\rho(T)$, i.e.,

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

The *diameter* of a tree T is defined as the length of the longest path in T i.e., the maximum eccentricity is the diameter. A spanning tree with minimum diameter is defined as *minimum diameter spanning tree*.

For the weighted tree T with n vertices and $n-1$ edges of the corresponding weighted permutation graphs, the inverse 1-center 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.

Our problem is to design an optimal algorithm to compute the inverse 1-center location problem on weighted permutation graphs.

2 Survey of the related work

As shown in [9, 50, 51], the inverse problems of many combinatorial/network optimization problems can be solved by strongly or weakly polynomial algorithms. In fact in [48], it is shown that for a large class of combinatorial/network optimization problems, if the original problem can be solved in polynomial times, then its inverse problem can be solved in polynomial time by a quite uniform methodology. A detailed survey on inverse optimization problems has been compiled by Heuberger [24]. In the context of location problems Cai et al. [10] proved that the inverse 1-center location problem with edge length modification on general un-weighted directed graphs is *NP*-hard, while the underlying center location problem is solvable in polynomial time. In 2004, Burkard et al. [5] consider inverse p-median location problems can be solved in polynomial time, when p is fixed and not an input parameter. They proposed a greedy like $O(n \log n)$ time algorithm for the inverse 1-median problem with vertex weight modifications on tree networks. Galavi [21] showed later that this problem can actually be solved in $O(n)$ time. Moreover, Burkard et al. [5] proved that the inverse 1-median problem on the plane under Manhattan (or Chebyshev) norm can be solved in $O(n \log n)$ time. Later the same authors [6] investigated the inverse 1-median problem with vertex weight modification and unit cost on a cycle. They showed that this problem can be solved in $O(n^2)$ time by using methods from computational geometry. In 2007, Gassner [20] suggested an efficient $O(n \log n)$ time solution method for the inverse 1-maxian problem with edge length modifications on tree networks. The inverse Fermat-Weber problem was studied by Burkard et al. [7, 8].

The authors derived a combinatorial approach which solves the problem in $O(n \log n)$ time for unit cost and under the assumption that the pre-specified point that should become a 1-median does not coincide with a given point in the plane. Galavii [21] showed that the 1-median on a path with positve/negative weights lies in one of the vertices with positive weights or lies in one of the end points of the path. This property allows to solve the inverse 1-median problem on a path with negative weights in $O(n)$ time. Gassner [20] consider an inverse version of the convex ordered median problem and showed that this problem is *NP*-hard on general graphs, even on trees. Further, it was shown that the problem remains *NP*-hard for unit weights or if the underlying problem is a K-centrum problem (but not, if both of these conditions hold). The inverse unit-weight K-centrum problem with unit cost coefficients on a tree can be solved in $O(n^3k^2)$ time. Recently, Yang and Zhang [49] proposed an $O(n^2 \log n)$ time solution method for the inverse vertex center problem on a tree provided that the modified edge lengths always remain positive. Dropping this condition, they mention that the general problem can be solved in $O(n^3 \log n)$ time. Some papers on circular-arc graphs which helps to complete this paper are Jana et al. [26], Mondal et al. [37, 38], Saha et al. [44, 45, 46], Swagata et al. [30, 31, 32, 33], Satyabrata et al. [42]. Recently, Alizadeh et al. [1] have designed an algorithm for inverse 1-center location problem with edge length augmentation on trees in $O(n \log n)$ time, using a set of suitably extended AVL-search trees. In [2], Alizadeh et al. have designed a combinatorial algorithm for inverse absolute on trees in $O(n^2)$ time when topology not allowed and $O(n^2r)$ time when topology allowed.

Inverse optimization problems have recently attained significant theoretical interest due to their relevance in practice and for a comprehensive survey on inverse optimization problems see [17, 23, 24, 29].

Network location problems belong to basic optimization models which are concerned with finding the best location of single or multiple new facilities in a network of demand points such that a given function which depends on the distance between the facilities and clients becomes minimum. Depending on the the model under investigation, facilities or clients may either be placed only at vertices or may also lie on edges of the network. For further details on these problems the reader is referred to the books of Daskin [12], Drezner et al. [16], Francis et al. [19], Mirchandani et al. [35] and Nickel et al. [41].

Burton and Toint [4] first investigated an inverse shortest paths problem in 1992. Since then, many problems have been considered by various authors, working at least partly independently. The notation of 'inverse optimization' is always similar, but not the same. Recently, Jana et al. [25, 27, 28] have designed a linear time algorithm to compute inverse 1-center location problem on the edge weighted trees, weighted interval graphs respectively and weighted circular-arc graphs.

2.1 Applications of the problem

For instance, an important application comes from geophysical sciences and concerns predicting the movements of earthquakes. To achieve this aim, geologic zones are discretized into a number of cells. Adjacency relations can be modeled by arcs in a corresponding network (Moser [39]). Although some estimates for the transmission times are known, precise values are hard to obtain. By observing an earthquake and the arrival times of the resulting seismic perturbations at various points and assuming that earthquakes travel along shortest paths, the problem is to refine the estimates of the transmission times between the cells. This is just an inverse shortest path problem.

Another possible application actually changes the real costs: Assume that we are given a road network and some facility in it. The aim is to place the facility in such a way that the maximum distance to the customers is minimum. However we are often faced with the situation that the facility already exists and can not be relocated with reasonable costs. In such a situation, we may want to modify the network as little as possible (improving roads costs), such that the location of the facility becomes optimum (or such that the distances to the customers do not exceed some given bounds). This is an example of the inverse center location problem. When modeling traffic networks, a further option is to impose tolls in order to enforce an efficient use of the network (Dial [15]). The choice of the word 'inverse optimization' was motivated in part by the widespread use of inverse methods in other fields, for instance Marlow [34] and Engl et al. [18].

2.2 Our result

In this paper, we design an algorithm to compute inverse 1-center location problem on weighted permutation graphs in $O(n)$ time, where n is the number of vertices of the graph.

2.3 Organization of the paper

In the next section, i.e., Section 3, we shall discuss about the construction of the tree. In subsection 3.1, we discuss BFS tree on permutation graph. In subsection 3.2, we discuss about the minimum diameter spanning tree. In Section 4, we develop modified spanning tree of the tree T. In Section 5, we present an algorithm to get inverse 1-center of the permutation graph. The time complexity is also calculated in this section. In Section 6, we give a conclusion.

3 Construction of the tree

3.1 Construction of BFS tree on permutation graph

It is well known that BFS is an important graph traversal technique and also BFS constructs a BFS tree. In BFS, started with vertex v, we first scan all edges incident on v and then move to an adjacent vertex w. At w we then scan all edges incident to w and move to a vertex which is adjacent of w. This process is continued till all the edges in the graph are scanned [14].

BFS tree can be constructed on general graphs in $O(n+m)$ time, where n and m represent respectively the number of vertices and number of edges [47]. To construct this BFS tree on a permutation graph we consider the matching diagram. We first placed the line segment x on the zero level and all vertices adjacent to x on the first level. Next we traverse the matching diagram step by step. In first step we scan the untraversed line segment which are situated at the right side of line segments x and in the second step we scan all untraversed line segments which are situated at the left side of the line segment x. In each iteration, we scan numbers within an interval on the top channel and scan permutation number within an interval on the bottom channel alternately in an order. Recently Mondal et al. [36], have designed an algorithm to construct a BFS tree $T^*(i)$ with root as i on trapezoid graph in $O(n)$ time, where n is the number of vertices and Barman et al. have designed another algorithm to construct the BFS tree on a

permutation graph with any vertex x as root in $O(n)$ time [3]. Figure 3 is the BFS tree constructed by the **Algorithm PBFS** [3].

3.2 Computation of minimum diameter spanning tree

Let $T(1)$, $T(\pi(1))$ be two BFS trees designed by **Algorithm PBFS** [3] of a permutation graph G. Let T_{PER} be the minimum heighted tree between $T(1)$ and $T(\pi(1))$. Let P be the main path (longest path) of T_{PER} of the permutation graph 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 vertex 1 or the vertex corresponding to $\pi(1)$ of G. The vertices of the path P, i.e., u_i^*, $i = 0, 1, 2, 3, \ldots, k$ defined as *internal nodes*.

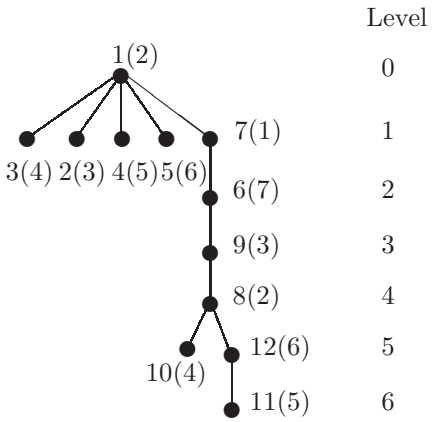

Figure 3: BFS tree T_{PER} rooted at vertex 1 of the permutation graph G shown in Figure 1.

We have the following terms:

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

Next we define *level* of the vertex u as the distance of u from the root i of the BFS tree $T(i)$ and denote it by *level(u)*, $u \in V$ and take the level of the root i as 0. The level of each vertex on BFS tree $T(i), i \in V$ can be assigned by the BFS algorithm of Chen and Das [11]. The level of each vertex of the tree T_{PER} can be computed in $O(n)$ time.

As per construction of BFS tree by the algorithm **PBFS** [3], we have the important result in BFS tree T_{PER}.

In permutation diagram, two intersecting line segments of *V-type* or *inverted V-type* defined as the *scissors type line segments* of permutation graphs. Figure 4 gives the illustration of scissors type line segments.

Lemma 1 *Minimum number of scissors type line segments with maximum spread of the permutation diagram 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 4 and Figure 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 $top \rightarrow bottom \rightarrow top \rightarrow bottom \rightarrow \ldots$.

Similar manner to be followed from the bottom line. Then we obtain the another path of the form $bottom \rightarrow top \rightarrow bottom \rightarrow top \rightarrow \ldots$. 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 lines segment of the permutation graph covered the whole region. □

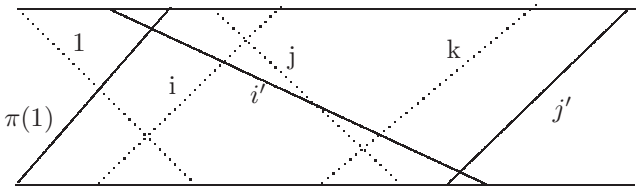

Figure 4: An Illustration.

Lemma 2 *[3] Two intersecting line segments of the permutation graph of the matching diagram are assigned on the same level or adjacent levels.*

Lemma 3 *If $u, v \in V$ and $|level(u) - level(v)| > 1$ in T_{PER}, then there is no edge between the vertices u and v in G.*

Proof. If possible let $|level(u) - level(v)| > 1$ but $(u, v) \in E$, i.e., u and v are directly connected. Since u and v are directly connected so by Lemma 11 either $level(u) = level(v)$ or $|level(u) - level(v)| = 1$ which is contradictory to the assumption that $|level(u) - level(v)| > 1$. Hence $(u, v) \notin E$, i.e., u and v are not directly connected in G. □

The time complexity of the algorithm PBFS is stated below.

Theorem 1 *[3] The BFS tree rooted at any vertex $x \in V$ can be computed in $O(n)$ time for a permutation graph containing n vertices.*

Obviously, this BFS is a spanning tree.

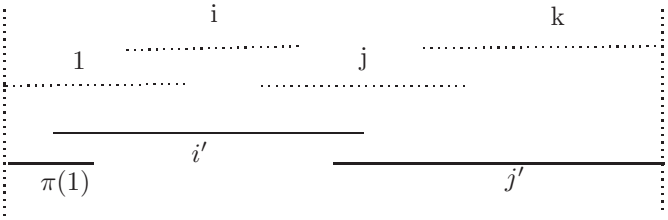

Figure 5: Region covered by the projection of minimum number of line segments.

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

Lemma 4 *The spanning tree T_{PER} is a minimum diameter spanning tree.*

Proof. According to the construction the BFS tree, the main path of the tree T_{PER} is the longest path which is the diameter of T_{PER}. The main path contains minimum number of scissors type line segments (by lemma 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 minimum diameter spanning tree. □

In next section, we compute the inverse 1-center of the permutation graph by modifying the spanning tree T_{PER}.

4 Inverse 1-center

In this section we discuss about inverse 1-center.
Now, before going to our proposed algorithm we introduce some notations for our algorithmic purpose. Let i be the pre-specified vertex in G.

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

To find the inverse 1-center, we discuss 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 inverse 1-center of the weighted tree corresponding to the permutation graph.

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

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

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

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

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

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

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

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

Case-2.6. : When w_{max} is less than the product of the number of vertices except the vertex i in the path whose weight is minimum and maximum weight of the vertex in the graph, i.e., $w_{max} < k_2 w_2$.

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

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

subject to

$\max_{v\in V(T_{CIR})} d_{\overline{w}}(v,i) \leq \max_{v\in V(T_{PER})} d_{\overline{w}}(v,p)$, for all $p \in T_{PER}$(or $p \in V(T_{PER})$),
$\overline{w}(v) = w(v) + x\{w(v)\} - y\{w(v)\}$ for all $v \in V(T_{PER})$,
$x\{w(v)\} \leq w^{+}\{w(v)\}$, for all $v \in V(T_{PER})$,
$y\{w(v)\} \leq w^{-}\{w(v)\}$, for all $v \in V(T_{PER})$,
$x\{w(v)\}, y\{w(v)\} \geq 0$, for all $v \in V(T_{PER})$,

where $\overline{w}(v)$ be the modified vertex weight, $w^{+}\{w(v)\} = w_{upp}(v) - w(v)$ and $w^{-}\{w(v)\} = w(v) - w_{low}(v)$ are the maximum feasible amounts by which $w(v)$ can be increased and reduced respectively, i.e., $w_{low}(v) \leq \overline{w}(v) \leq w_{upp}(v)$, $x\{w(v)\}$ and $y\{w(v)\}$ are the maximum amounts by which the vertex weight $w(v)$ is increased and reduced respectively, $c^{+}(w(v))$ is the non negative cost if $w(v)$ is increased by one unit and $c^{-}(w(v))$ is the non negative cost if $w(v)$ is reduced by one unit. Every feasible solution (x, y) with $x = \{x(w(v)) : v \in V(T_{PER})\}$ and $y = \{y(w(v)) : e \in V(T_{PER})\}$ is also called a feasible modification of the inverse 1-center location problem.

Now, we prove the following results.

Lemma 5 *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 and i is the inverse 1-center.*

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

Finally in T'_{PER}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the inverse 1-center of the given weighted permutation graph. Hence the result. □

Lemma 6 *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 and i is the inverse 1-center.*

Proof. Since we can decrease the weight of each vertex except the root up to minimum weight of the vertex 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 vertices 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 3). 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 permutation graph is an arbitrary, so our assumption is true for any permutation graphs.

Finally in T'_{PER}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the inverse 1-center of the given weighted permutation graph. Hence the result. □

Lemma 7 *If $w_{min} < k_1w_1$ in T_{PER}, 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 inverse 1-center.*

Proof. Since we can decrease the weight of each vertex up to minimum weight of the vertex in T_{PER}, so we can reduce the weights of the vertices except the root in the path whose weight is maximum in such a way that its least weight of the path becomes k_1w_1. Again we have $w_{min} < k_1w_1$. Therefore we can increase the weights $(k_1w_1 - w_{min})$ to the vertices except the root in the path whose weight is minimum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 3). 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 permutation graph is an arbitrary, so our assumption is true for any permutation graphs.

Finally in T'_{PER}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the inverse 1-center of the given weighted permutation graph. Hence the result. □

Lemma 8 *If $w_{max} = k_2w_2$ in T_{PER}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding condition in the path whose weight is minimum and i is the inverse 1-center.*

Proof. If k_2 be the number of vertices in the minimum weighted path L_i or R_i and w_2 be the maximum weight of the vertex among the vertices in T_{PER} as well as L_i or R_i, then there is a scope to increase the weight of each vertex up to w_2. As k_2 vertices is there in the path L_i or R_i, so we can enhance 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 permutation graph is an arbitrary, so our assumption is true for any permutation graphs.

Finally in T'_{PER}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the inverse 1-center of the given weighted permutation graph. Hence the result. □

Lemma 9 *If $w_{max} > k_2w_2$ in T_{PER}, then $w^*(L_i) = w^*(R_i)$ by enhance the weights of all vertices up to maximum weight except the root i maintaining the bounding condition in path whose weight is minimum and reducing the weights of some vertices except the root i in the path whose weight is maximum and i is the inverse 1-center.*

Proof. Since we can increase the weight of each vertex up to maximum weight of the vertex in T_{PER}, so we can enhance the weights of all vertices except the root i in the path whose weight is minimum in such a way that its greatest weight of the path becomes k_2w_2. Again we have $w_{max} > k_2w_2$. Therefore we can reduces the weights $(w_{max} - k_2w_2)$ to some vertices except the root in the path whose weight is maximum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique

(Section 3). 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 permutation graph is an arbitrary, so our assumption is true for any permutation graphs.

Finally in T'_{PER}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the inverse 1-center of the given weighted permutation graph. Hence the result. □

Lemma 10 *If $w_{max} < k_2w_2$ 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 and i is the inverse 1-center.*

Proof. Since we can increase the weight of each vertex up to maximum weight of the vertex in T_{PER}, so we can enhance the weights of the vertices except the root i in the path whose weight is minimum in such a way that its greatest weight of the path becomes k_2w_2. Again we have $w_{max} < k_2w_2$. Therefore we can increase the weights $(w_{max} - w_{min})$ to some vertices except the root i in the path whose weight is minimum using the conditions of non-linear semi-infinite (or nonlinear) optimization model technique (Section 3). 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 permutation graph is an arbitrary, so our assumption is true for any permutation graphs.

Finally in T'_{PER}, we have $w^*(L_i) = w^*(R_i)$, which implies that i is the inverse 1-center of the given weighted permutation graph. Hence the result. □

5 Algorithm and its complexity

In this section we propose a combinatorial algorithm for the inverse 1-center location problem on the vertex weighted tree T_{PER}. The main idea of our proposed algorithm is as follows:

Let T_{PER} be a weighted tree corresponding to the permutation graph 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 inverse 1-center. At first we calculate the path whose weight is maximum from i to any pendant vertex of T_{PER}. 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 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 inverse 1-center of the graph. If $w(L_i) \neq w(R_i)$, then six cases may arise. In the first case, if $w_{min} = k_1w_1$ 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 conditions (Section 3) in the path whose weight is maximum and i is the inverse 1-center. In the second case, if $w_{min} > k_1w_1$ 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 3) in the path whose weight is maximum and i is the inverse 1-center. In third case, if $w_{min} < k_1w_1$ 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 (Section 3) 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 inverse 1-center. 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 (Section 3) in the path whose weight is minimum and i is the inverse 1-center. 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 3) 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 inverse 1-center. 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 (Section 3) in the path whose weight is minimum and i is the inverse 1-center.

Our proposed algorithm to the inverse 1-center location problem of the tree corresponding to the permutation graph G is as follows:

Algorithm 1-INV-PER-TREE

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

Output: Vertex i as inverse 1-center 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 inverse 1-center 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 $w_{max} = k_2w_2$ 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 $w_{max} > k_2w_2$ 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 $w_{max} < k_2w_2$ 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 inverse 1-center.

end 1-INV-PER-TREE.

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

Illustration of the Algorithm 1-INV-PER-TREE to the tree T_{PER} in Figure 3 : Let $i = 1$ be the pre-specified vertex of the tree T_{PER} which is to be inverse 1-center. 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 $w(R_i)$ be the sum of the weights of the vertices except the root $i = 1$ of the paths 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 $k_1w_1 = 6$. Therefore $w_{min} < k_1w_1$. 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 (Section 3) 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 inverse one center.

The above illustration gives the following table.

Pre-specified vertex of the tree T_{PER}	$i = 1$
Longest left path L_i from the vertex 1 to the vertex 3	$1 \rightarrow 3$
Longest right path R_i from the vertex 1 to the vertex 11	$1 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 8 \rightarrow 12 \rightarrow 11$
Weight of the path L_i except the vertex $i = 1$	$w(L_i) = 4$
Weight of the path R_i except the vertex $i = 1$	$w(R_i) = 24$
$w_{min} = w(L_i)$	4
$w_{max} = w(R_i)$	24
Number of vertices in R_i except $i = 1$	$k_1 = 6$
Minimum weight of the vertex in the graph	$w_1 = 1$
$k_1 w_1$	6
w_{min}	$< k_1 w_1$
Reduce the weights of each vertex in R_i except the root 1 up to minimum weight	$1 + (7 - 6) + (3 - 2) + (2 - 1)$ $+(6 - 5) + (5 - 4) = 6$
Increase the weight 2 to the vertex 3 in L_i	$4 + 2 = 6$
$w^*(L_i)$	6
$w^*(R_i)$	6
The weights of $w^*(L_i)$ and $w^*(R_i)$ are equal	$w^*(L_i) = w^*(R_i)$
The vertex which is inverse 1-center	$i = 1$

Now we have the modified tree T'_{PER} (Figure 6) with modified vertex weight.

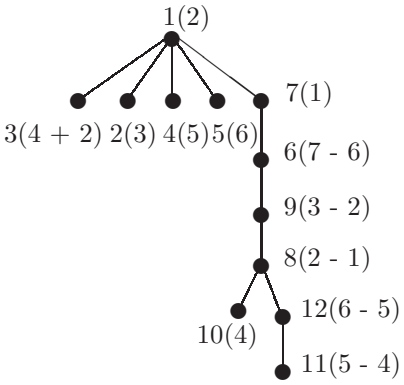

Figure 6: Modified tree T'_{PER} of the tree T_{PER} .

Next we shall prove the following important result.

Lemma 11 *The **Algorithm 1-INV-PER-TREE** correctly computes the inverse 1-center of the weighted permutation graph.*

Proof. Let i be the pre-specified vertex in T_{PER}. We have to prove that i is the inverse 1-center. At first, by Step 1, we have constructed the tree T_{PER} (as per section 2) 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 inverse 1-center 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 circular-arc 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 the inverse 1-center. Hence **Algorithm 1-INV-PER-TREE** correctly computes the inverse 1-center for any vertex weighted tree. □

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

Lemma 12 *The specified vertex i in the modified tree T'_{PER} is the inverse 1-center.*

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 inverse 1-center. □

The following describe the total time complexity of the algorithm to compute inverse 1-center problem on the weighted tree corresponding to the weighted permutation graph G.

Theorem 2 *The time complexity to find inverse 1-center problem on a given vertex weighted tree T'_{PER} corresponding to the weighted permutation graph G is $O(n)$, where n is the number of vertices of the graph.*

Proof. Step 1 takes $O(n)$ time, since the adjacency relation of permutation graph 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 sum of the weights of the paths. Also, Step 4.1 takes $O(1)$ time. Computation of k_1 and k_2, i.e., number of vertices in R_i and L_i 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 vertices and $(n-1)$ edges, so Step 4.3 can be executed in $O(n)$ time. Hence overall time complexity of our proposed **Algorithm 1-INV-PER-TREE** is $O(n)$ time, where n is the number of vertices of the permutation graph. □

6 Concluding remarks

In this paper, we investigated the inverse 1-center location problem with vertex weights on the tree corresponding to the weighted permutation graph G. Firstly, we develop minimum heighted tree with two branches of level difference either zero or one of the permutation graph. Secondly, we modified the tree maintaining the bounding conditions to get inverse 1-center. The time complexity of our proposed algorithm is $O(n)$, where n is the number of vertices of the permutation graph G. This idea can be applied to solve the 1-center location problem to other graphs.

7 Acknowledgement

We thank the anonymous referees for their valuable suggestions which improved the presentation and readability of the article. This work is partially supported by the Department of Higher Education, Science and Technology and Biotechnology , Govt. of West Bengal, India $(245(Sanc)/ST/P/SandT/16G-20/2017dt.25/3/2018)$.

References

[1] Alizadeh, B. and Burkard, R. E., Inverse 1-center location problems with edge length augmentation on tree, *Computing*, 86 (2009) 331-343.

[2] Alizadeh, B. and Burkard, R. E., Combinatorial algorithms for inverse absolute and vertex 1-center location problems on trees, *Networks*, 58 (2011) 190-200.

[3] Barman, S., Mondal, S. and Pal, M., An efficient algorithm to find next-to-shortest path on permutation graphs, *Communicated*.

[4] Burton, D. and Toint, Ph. L., On an instance of the inverse shortest paths problem, *Math. Programming*, 53 (1992) 45-61.

[5] Burkard, R. E., Pleschiutschnig, C. and Zhang, J., Inverse median problems, *Discrete Optimization*, 1 (2004) 23-39.

[6] Burkard, R. E., Pleschiutschnig, C. and Zhang, J., The inverse 1-median problem on a cycle, *Discrete Optimization*, 16 (2007) 50-67.

[7] Burkard, R. E., Galavii, M. and Gassner, E., The inverse Fermal-Weber problem, *Technical Report 2008-14*, Graz University of Technology, Graz, 2008.

[8] Burkard, R. E., Alizadeh, B., Inverse center location problems, *International Symposium on Combinatorial Optimization*, 36 (2010) 105-110.

[9] Cai, M. C. and Li, Y. J., Inverse matroid intersection problem, *ZOR Math. Meth. Oper. Res.* 45 (1997) 235-243.

[10] Cai, M. C., Yang, X. G. and Zhang, J. Z., The complex analysis of the inverse center location problem, *Journal of Global Optimization*, 15 (1999) 213-218.

[11] Chen, C. C. Y. and Das, S. K., Breath-first traversal of trees and integer sorting in parallel, *Information Processing Letters*, 41 (1992) 39 - 49.

[12] Daskin, M. S., Network and discrete location: modeles, algorithms and applications, *Wiley, New York*, 1995.

[13] Deng, X., Hell, P. and Huang, J., Linear time representation algorithms for proper circular-arc graphs and proper interval graphs, *SIAM J. Comput.*, 25 (1996) 390-403.

[14] Deo, N., Graph theory with applications to engineering and computer science (*Prentice Hall of India Private Limited, New Delhi*, 1990).

[15] Dial, R. B., Minimal-revenue congestion pricing part-I: A fast algorithm for the single-origin case, *Transportation Res. Part B*, 33 (1999) 189-202.

[16] Drezner, Z., Hamacher, H. W., Facility location, applications and theory, *Springer, Berlin*, 2004.

[17] Duin, C. W. and Volgenant, A., Some inverse optimization problems under the Hamming distance, *European Journal of Operational Research*, 170 (2006) 887-899.

[18] Engl, H. W., Hanke, M. and Neubauer, A., Regularization of inverse problems, *Kluwer Academic Publishers Group: Dordrecht*, 1996.

[19] Francis, R. L., McGinnis, L. F. and White, J. A., Facility layout and location, an analytical approach. Prentice Hall, *Englewood Cliffs*, 1992.

[20] Gassner, E., The inverse 1-maxian problem with edge length modification, *J. Combinatorial Optimization*, 16 (2007) 50-67.

[21] Galavi, M., Inverse 1-median problems, Ph. D. thesis, Institue of Optimization and Discrete Mathematics, *Graz University of Technology, Graz*, 2008.

[22] Golumbic, M. C., *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 2004.

[23] Guan, X. and Zhang, J., Inverse bottleneck optimization problems under weighted Hamming Distance, *Lecture Notes in Computer Science*, 4041 (2006) 220-230.

[24] Heuberger, C. Inverse combinatorial optimization: A survey on problems, methods, and results, *Journal of Combinatorial Optimization*, 8 (2004) 329-361.

[25] Jana, B., Mondal, S. and Pal, M., Computation of inverse 1-center location problem on the weighted trees, *CiiT International Journal of Networking and Communication Engineering*, 4 (2012) 70 - 75.

[26] Jana, B. and Mondal, S. and Pal, M., Computation of a minimum average distance tree on circular-arc graphs, *International Journal of Electronics Communication and Computer Engineering*, 6 (2015) 384 - 390.

[27] Jana, B., Mondal, S. and Pal, M., Computation of inverse 1-center location problem on the weighted interval graphs, *Communicated*.

[28] Jana, B., Mondal, S. and Pal, M., Computation of inverse 1-center location problem on the weighted circular-arc graphs, *Communicated*.

[29] Kellerer, H., Pferschy, U. and Pisinger, D., Knapsack problems, *Springer, Berlin*, 2004.

[30] Mandal, S. and Pal, M., A Sequential Algorithm to Solve Next-To-Shortest Path Problem on Circular-Arc Graphs, *Journal of Physical Sciences*, 10 (2006) 201 - 217.

[31] Mandal, S. and Pal, M., Maximum Weight Independent set of Circular-Arc Graph and Its Application, *Journal of Applied Mathematics and Computing*, 22 (2006) 161 - 174.

[32] Mandal, S. and Pal, M., An Optimal Sequential Algorithm to Compute All Hinge Vertices on Circular-Arc Graphs, *Arab Journal of Mathematics and Mathematical Sciences*, 1 (2007) 1 - 12.

[33] Mandal, S., Pal, A. and Pal, M., An Optimal Algorithm to Find Centres and Diameter of a Circular-Arc Graph, *Advanced Modeling and Optimization*, 9 (2007) 155 - 170.

[34] Marlow, B. and Megiddo, N., Inverse problems and linear-time algorithms for linear programming in R3 and related problems, *SIAM J. Comput.*, 12 (1983) 759-776.

[35] Mirchandani, B. P. and Francis, R. L., Discrete location theory, *Wiley, New York*, 1990.

[36] Mondal, S., Pal, M. and Pal, T. K., An optimal algorithm for solving all-pairs shortest paths on trapezoid graphs, *Intern. J. Comput. Engg. Sci.* 3 (2002) 103-116.

[37] Mondal, S., Pal, A. and Pal, M., An optimal algorithm to find centres and diameter of a circular-arc graph, *Advanced Modeling and Optimization*, 9 (2007) 155 - 170.

[38] Mondal, S. and Pal, M., An optimal sequential algorithm to compute all hinge vertices on circular-arc graphs, *Arab Journal of Mathematics and Mathematical Sciences*, 1 (2007) 1 - 12.

[39] Moser, T. J., Shortest paths calculation of seismic rays, *Geophysics*, 56 (1991) 59-67.

[40] Muller, J. H. and Spinrad, J., Incremental modular decomposition, *J. ACM*, 36 (1989) 1 - 19.

[41] Nickel, S., Puerto, J., Location theory, a unified approach, *Springer, Berlin*, 2005.

[42] Paul, S., Pal, M. and Pal, A., L(2, 1) - labelling of Circular-Arc Graph, *Annals of Pure and Applied Mathematics*, 5 (2014) 208 - 219.

[43] Pnueli, A., Lempel. A. and Even, S., Transitive orientation of graphs and identification of permutation graphs, *Canad. J. Math.*, 23 (1971) 160 - 175.

[44] Saha, A., Pal, M. and Pal, T. K., An optimal parallel algorithm for solving all-pairs shortest paths problem on circular-arc graphs, *Journal of Applied Mathematics and Computing*, 17 (2005) 1 - 23.

[45] Saha, A., Computation of average distance, radius and center of a circular-arc graph in parallel, *Journal of Physical Sciences*, 10 (2006) 178 - 187.

[46] Saha, A., Pal, M. and Pal, T. K., Selection of program slots of television channels for giving advertisement : A graph theoretic approach, *Information Sciences*, 177(12) (2007) 2480 - 2492.

[47] Tarjan, R. E., Depth first search and linear graph algorithm, *SIAM J. Comput.*, 2 (1972) 146-160.

[48] Yang, C. and Zhang, J. Z., Two general methods for inverse optimization problems, *Appl. Math. Lett.* 12 (1999) 69-72.

[49] Yang, X. and Zhang, J., Inverse center location problem on a tree, *Journal of Systems Science and Complexity*, 21 (2008) 651-664.

[50] Zhang, J. Z. and Liu, Z. H., Calculating some inverse linear programming problems, *J. Computational and Applied Mathematics*, 72 (1996) 261-273.

[51] Zhang, J. Z. and Ma, Z. F., A network flow method for solving some inverse combinatorial optimization problems, *Optimization*, 37 (1996) 59-72.