Filomat 35:4 (2021), 1333–1342
https://doi.org/10.2298/FIL2104333N

Published by Faculty of Sciences and Mathematics,
University of Niš, Serbia
Available at: http://www.pmf.ni.ac.rs/filomat

The Reverse Total Weighted Distance Problem on Networks with Variable Edge Lengths

Kien Trung Nguyen[a], **Nguyen Thanh Hung**[a]

[a]*Department of Mathematics, Teacher College, Can Tho University, Can Tho, Vietnam.*

Abstract. We address the problem of reducing the edge lengths of a network within a given budget so that the sum of weighted distances from each vertex to others is minimized. We call this problem the reverse total weighted distance problem on networks. We first show that the problem is NP-hard by reducing the set cover problem to it in polynomial time. Particularly, we develop a linear time algorithm to solve the problem on a tree. For the problem on cycles, we devise an iterative approach without mentioning the exact complexity. Additionally, if the cycle has uniform edge lengths, we can prove that the specified approach runs in $O(n^3)$ time as each edge of the cycle can be reduced at most once, where n is the number of vertices in the underlying cycle.

1. Introduction

While classical combinatorial optimization problems look for an optimal solution based on the input data, the inverse/reverse optimization setting aims to change parameters of the problem at minimum cost so that the decision maker obtains his goal. Recently, the inverse/reverse optimization problem has become an interesting topic in operational research with intensive publications and promising applications. To ease the readers, we review some existing literatures concerning the inverse and reverse optimization problems as in the following.

The inverse optimization problem is to modify parameters at minimum cost so that the prespecified solution becomes optimal with respect to new parameters. This problem was first investigated by Burton [8] in his work concerning the inverse shortest path problem. As cracks on earth move in the shortest path, this problem was applied to predict the movement of earthquake. Then, Ahuja and Orlin [3] considered the inverse linear programming problem. They proved that the inverse linear program can be reduced to the same type of problem based on the complementary slackness condition. Ahuja and Orlin [4] also developed a combinatorial algorithm to solve the inverse network flow problem. The inverse minimum spanning tree problem is solvable efficiently by Hochbaum [15]. Nguyen and Chassein [19] considered the inverse eccentric vertex problem. They showed the *NP*-completeness of the problem on cactus graphs and then developed linear time algorithms to solve the problem on cycles and on trees. Additionally, the inverse location problem was investigated with complexity result and efficient solution approach; for

2010 *Mathematics Subject Classification*. 90B10; 90B80; 90C27.
Keywords. Combinatorial optimization - Reverse optimization - Complexity - Tree - Cycle.
Received: 03 April 2020; Accepted: 03 December 2020
Communicated by Predrag Stanimirović
This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 101.01-2019.325.
Email addresses: trungkien@ctu.edu.vn (Kien Trung Nguyen), nthung@ctu.edu.vn (Nguyen Thanh Hung)

example, [1, 9, 10, 19–21], to mention a few. On a counter part, the reverse optimization problem on a network aims to modify parameters within a given budget so that the behavior network is improved as much as possible. The improvement of a network is appropriately defined by the decision maker. For the sake of reverse location problems on networks, we next review some of existing literatures.

Berman et al. [6, 7] were the first who investigated the reverse 1-median and 1-center on general networks. However, they proposed just the heuristic approach without paying attention to the exact procedure and the complexity. Then the reverse 1-center problem on uniform weighted trees was reduced to minimum cut problems on series-parallel graphs and solved in quadratic time; see Zhang et al. [24, 25]. The authors also claimed that the reverse center location on networks can be approximated by investigating the problem on a suitable spanning tree on the underlying network. Then, Nguyen [18] developed a quadratic algorithm for the reverse 1-center problem on weighted trees. The reverse 1-median problem was proved to be NP-hard on general graphs and then solved in linear time if the network is a cycle; see Burkard et al. [11]. Moreover, Burkard et al. [12] reduced both of the reverse 2-median problem on trees and the reverse 1-median problem on graphs with exactly one cycle to the reverse 2-median on a path. Then, they developed algorithm that solved the reduced problem in $O(n \log n)$ time, where n is the input size of the problem. Wang and Bai [23] investigated the reverse 2-median problem on a cycle and showed that the problem can be decomposed to the equivalent reverse 3-median problem on a path. Then, they devised a polynomial time algorithm for the corresponding induced problem. For the reverse undesirable location problem, Alizadeh and Etemad [2] developed linear time algorithms for the reverse obnoxious center problem on networks, where the modifying cost is measured under rectilinear norm or Hamming distance. Also, Etemad and Alizadeh [13] further studied the reverse obnoxious center problem on a tree $T = (V, E)$ and proposed an $O(n^2)$ time algorithm for the problem. Sepasian further considered the reverse 1-maxian problem with keeping the 1-median and developed an $O(n \log n)$ algorithm. The complexity of the corresponding problem can be improved to linear if vertex weights are allowed to be increased. Interesting readers can refer to a survey of inverse and reverse combinatorial optimization problem with solution methods of Heuberger [14].

This paper considers the reverse total weighted distance on networks. Here, we reduce the edge lenghs of the network within a certain budget such that the sum of weighted distances between any two vertices is minimized. The problem is possibly applied in network design. For example, we are given a network system, all nodes of the network play both roles, say the server and the client, and edge lengths in the network measure the time for communicating between two incident vertices. Thus, we require to reduce the lengths of edges in an efficient way such that the total communication time can be improved as much as possible. According to the best of our knowledge, this problem has not been studied so far in spite of its importance. The paper is organized as follows. We introduce the problem in Sect. 2 and prove the *NP*-hardness of the problem on general networks. We develop in Sect. 3 an efficient algorithm to solve the reverse total weighted distance on trees in linear time. Sect. 4 contributes to an algorithmic approach for the problem on cycles. Although the complexity of the proposed algorithm is not exactly known, we claim that the algorithm runs in $O(n^3)$ time on a cycle with uniform edge lengths, where n is the number of vertices.

2. Problem Statement and Complexity Result

Given a connected graph $G = (V(G), E(G))$, each vertex $v \in V(G)$ is associated with a nonnegative weight, say $w(v)$. Moreover, the length of edges in the graph is identified by a positive function $\ell : E \longrightarrow \mathbb{R}^+$, i.e., the length of an edge e is written as $\ell(e)$. The distance between two vertices u and v, denoted by $d(u, v)$, in $V(G)$ is the length of the shortest path connecting them. We have the following concept.

Definition 2.1. *The total weighted distance of the graph G is*

$$\textstyle\sum_{u \in V(G)} \sum_{v \in V(G), v \neq u} w(v) d(u, v).$$

To compute the total weighted distance of G, we first find the distances between each pair of vertices in G by the algorithm of Floyd-Warshall in $O(|V|^3)$ time and then sum up the relevant weighted distances.

In the reverse total weighted distance problem on G, the length of each edge e can be reduced by an amount $x(e)$. The smallest possible modified length of an edge e is $\underline{\ell}(e)$. We assume that the modified edge lengths are always nonnegative. Therefore, the modification of an edge e is limited within the upper bound $\bar{x}(e) := \ell(e) - \underline{\ell}(e)$. Denote by $\tilde{\ell}$ the modified edge lengths, i.e., $\tilde{\ell}(e) := \ell(e) - x(e)$. Let $\tilde{d}$ be the distance measure with respect to the modified lengths $\tilde{\ell}$. Assume that reducing one unit length of e costs $c(e)$. For a given budget B, we can formally state the reverse total weighted distance problem on G as follows.

- The modified total weighted distance $\sum_{u \in V(G)} \sum_{v \in V(G), v \neq u} w(v)\tilde{d}(u,v)$ is minimized.
- The total cost is limited within certain budget B, i.e., $\sum_{e \in E(E)} c(e)x(e) \leq B$.
- Modifications are feasible, i.e., $0 \leq x(e) \leq \bar{x}(e)$ for $e \in E(G)$.

Let us revisit the problem of improving minisum facilities on networks through edge length modifications (see Berman et al. [6]). Given a network G and a prespecified vertex v^*, we aim to reduce the edge lengths of the network within a given budget so that the total modified weighted distances to v^* is minimized. In this model, we can consider v^* as a server and all vertices as clients. On the other hand, if we consider all vertices of the network as servers which serve itself and other vertices, we get another version, say the reverse total weighted distance on G.

We first get the following complexity result of the problem.

Theorem 2.2. *The reverse total weighted distance problem on networks is NP-hard.*

Proof. Let us consider an instance of the Set Cover Problem (SC). Given a set $\mathcal{S} = \{1, \ldots, n\}$ and m nonempty subsets $\{\mathcal{P}_j\}_{j=1,\ldots,m}$ whose union equals to the set $\mathcal{S}$. (SC) is to determine whether there exists k subsets in $\{\mathcal{P}_j\}_{j=1,\ldots,m}$ such that their union is exactly $\mathcal{S}$? This problem is NP-complete; see Garey and Johnson [16].

Decision version of the reverse total weighted distance (RTWD) on networks is stated as follows. Let a graph G, a budget B, and modification bounds be given. Determine if there exists a modification of edge lengths within the budget B so that the modified total weighted distance is at most M?

Given an instance (SC), we can construct the instance of (RTWD) in polynomial time as follows.

- The graph $G = (V(G), E(G))$, with $V(G) = V_1 \cup V_2 \cup \{s\}$ and $E(G) = E_1 \cup E_2$. Here, we set

$$V_1 := \{u_i\}_{i=1,\ldots,m} \text{ and } V_2 := \{v_j\}_{j=1,\ldots,n}.$$
$$E_1 := \{(s, u_i)\}_{i=1,\ldots,m} \text{ and } E_2 := \{(u_i, v_j) \text{ if } j \in \mathcal{P}_i\}.$$

- The weight of s is 1 and other vertex weights are 0.
- The edge lengths are $\ell(e) := 2$ for $e \in E_1$ and $\ell(e) := 1$ for $e \in E_2$.
- Modifcation bounds are $\bar{x}(e) := 1$ for $e \in E_1$ and $\bar{x}(e) := 0$ for $e \in E_2$. Therefore, we know that $\underline{\ell}(e) = 1$ for all $e \in E$.
- We further choose $B := k$ and $M := 2n + 2m - k$.

In what follows, we prove that the answer to (SC) is 'yes' if and only if the answer to (RTWD) is 'yes'.

Assuming that there exists k subsets, say $\mathcal{P}_{i_1}, \mathcal{P}_{i_2}, \ldots, \mathcal{P}_{i_k}$, in $\{\mathcal{P}_j\}_{j=1,\ldots,m}$ so that their union equals to $\mathcal{S}$. We reduce the length of edges (s, u_{i_j}) for $j = 1, \ldots, k$ by one unit. Then the modifying cost is k. Moreover, we get the following modified weighted distances.

$$w(s)\tilde{d}(v_i, s) = 2 \text{ for } i = 1, \ldots, n.$$
$$w(s)\tilde{d}(u_j, s) = \begin{cases} 2 & \text{if } j \notin \{i_1, \ldots, i_k\}, \\ 1 & \text{otherwise.} \end{cases}.$$
$$w(u_j)\tilde{d}(v_i, u_j) = 0 \text{ and } w(v_i)\tilde{d}(u_j, v_i) = 0 \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, m.$$

Therefore, the total weighted distance of G is $2n + 2(m - k) + k = 2n + 2m - k = M$. This implies the answer to (RTWD) is 'yes'.

Conversely, assume that there exists a reduction of edge lengths with cost less than or equal to k and the total weighted distance of G is $2n + 2m - k$. As the reduction of edge length can not increase the total weighted distance, we can assume that the cost is exactly k. Hence, we get

$$\sum_{j=1}^{m} \tilde{d}(u_j, s) = \sum_{j=1}^{m} \tilde{d}(u_j, s) - k = 2m - k,$$

as we reduce the distances in $\{d(s, P_j)\}_{j=1,\ldots,m}$ by k. Since the total distance of G is not greater than $2n + 2m - k$, we obtain

$$\sum_{i=1}^{n} \tilde{d}(v_j, s) \leq 2n. \tag{1}$$

As $\tilde{d}(v_j, s) \geq 2$ for $j = 1, \ldots, n$, we obtain $\sum_{i=1}^{n} \tilde{d}(v_j, s) \geq 2n$. Thus, (1) holds with equality, i.e., $\tilde{d}(v_j, s) = 2$ for $j = 1, \ldots, n$. Let us denote by

$$J := \{j \in \{1, \ldots, m\} : \tilde{d}(s, u_j) = 1\}.$$

Note that $|J| \leq k$ as the cost is limited within k. For $i = 1, \ldots, n$, the shortest path from v_i to s is $P(v_i, s) = (v_i, u_j, s)$ for some $j \in J$. Indeed, if there does not exist $j \in J$ such that v_i is adjacent to u_j, then the modified weighted distance from v_i to s is strictly larger than 2. This contradicts the hypothesis. Therefore, we get the fact $\mathcal{S} = \cup_{j \in J} P_j$ or the answer to (SC) is 'yes'. □

The reverse total weighted distance on general graphs is NP-hard. Therefore, it is interesting to investigate some special cases which can be solved in polynomial time in the next coming sections.

3. The Problem on Trees

This section considers the reverse total weighted distance on a tree $T = (V(T), E(T))$. The total weighted distance of the tree T is reduced by Λ if

$$\textstyle\sum_{u \in V} \sum_{v \in V, v \neq u} w(v)\tilde{d}(u, v) = \sum_{u \in V} \sum_{v \in V, v \neq u} w(v)d(u, v) - \Lambda,$$

for some feasible reduction $\{0 \leq x(e) \leq \bar{x}(e)$ for e in E and $\sum_{e \in E} c(e)x(e) \leq B\}$. To minimize the total weighted distance of T, we have to maximize the reduction Λ.

After deleting an edge e in T, we get two connected components T_1^e and T_2^e. Denote by $V(\star)$ the set of vertices in a graph $\star$. Let us define an associated efficiency of e as

$$a(e) := |V(T_1^e)| \left(\sum_{v \in V(T_2^e)} w(v) \right) + |V(T_2^e)| \left(\sum_{v \in V(T_1^e)} w(v) \right).$$

We take into account the meaning of the associated efficiency of an edge as in the result below.

Proposition 3.1. *If the length of e is reduced by ε, then the total weighted distance of T is reduced by $a(e)\varepsilon$.*

Proof. For a vertex v in T_1^e, the modified weighted distance from v to u in T_2^e is

$$w(u)\tilde{d}(v, u) = w(u)d(v, u) - w(u)\varepsilon.$$

It means that the weighted distances from all vertices in T_1^e to u is reduced by $|V(T_1^e)|w(u)\varepsilon$. Therefore, the weighted distances from all vertices in T_1^e to all vertices in T_2^e is reduced by $|V(T_1^e)| \left(\sum_{v \in V(T_2^e)} w(v)\right) \varepsilon$. Similarly, we can compute the reduction of weighted distances from all vertices in T_2^e to all vertices in T_1^e and conclude that the total weighted distance of T is reduced by $a(e)\varepsilon$. □

Next we develop an algorithm to find $a(e)$ for all e in T. The key idea of the algorithm is to start with leaves of T. Then we contract the leaves in order to get an induced tree with new labels on the vertices.

Algorithm 1 Finds $a(e)$ for all $e \in E(T)$.

```
Input: A tree $T = (V(T), E(T))$ with $n$ vertices.
Compute $W := \sum_{v \in V(T)} w(v)$ and let $\mathcal{L}$ be the set of all leaves in the tree.
Label $La(v) := 1$ for each leaf $v \in \mathcal{L}$ and compute the degree $Deg(v)$ of each vertex $v \in V(T)$.
while $|\mathcal{L}| > 1$ do
  Set $Temp := \emptyset$
  for $v \in \mathcal{L}$ do
    Take the incident edge $e = (v', v)$ of $v$ and set $a(e) := La(v)(W - w(v)) + (n - La(v))w(v)$.
    Contract the edge $e$ to $v'$; set $Deg(v') := Deg(v') - 1$; and update $La(v') := La(v) + 1, w(v') := w(v) + w(v')$.

    if $Deg(v') = 1$ then
      Set $Temp := Temp \cup \{v'\}$.
    else
      if $Deg(v') = 0$ then
        Break the for loop.
      end if
    end if
  end for
  $\mathcal{L} := Temp$
end while
Output: Associated efficiencies $a(e)$ for all $e \in E$.
```

In each iteration, Algorithm 1 computes the associated efficiency $a(e)$ of the edge e that is incident to some leaf vertex. As the label $La(v)$ of a vertex v stands for the number of vertices of one subtree induced by deleting an adjacent edge of v, $n - La(v)$ is the number of vertices of another subtree. The total weight of one subtree induced by deleting an edge e is also updated in each iteration, the total weight of another subtree is also easy to obtain from W. Therefore, the associated number of each edge is computed correctly. As each iteration can be computed in constant time and there exists at most linear time many iterations, the algorithm runs in linear time.

Example 3.2. *Let an instance of a tree T be given in Figure 1 with edge lengths and vertex weights being labeled on the corresponding edges and vertices. We can apply Algorithm 1 to compute the associated number of each edge in Table 1.*

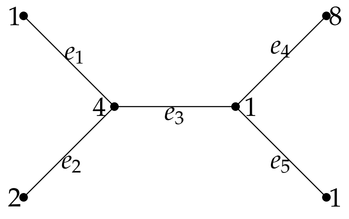

Figure 1: An instance of a tree T

After identifying all associated efficiency $a(e)$ for $e \in E$, we can formulate the reverse total weighted

i	1	2	3	4	5
$a(e_i)$	21	25	51	49	21

Table 1: Associated number of each edge in the tree T

distance as follows.

$$\begin{aligned} \max \quad & \sum_{e\in E} a(e)x(e) \\ \text{s.t.} \quad & \sum_{e\in E} c(e)x(e) \le B, \\ & 0 \le x(e) \le \bar{x}(e), \quad \forall e \in E. \end{aligned} \tag{2}$$

Problem (2) is a linear knapsack problem, it is therefore solvable in linear time; for example, see Balas and Zemel [5].

Theorem 3.3. *The reverse total weighted distance problem on a tree can be solved in linear time.*

4. The Problem on Cycles

We index the cycle $C = (V(C), E(C))$, with $|V(C)| = n$, in counter-clockwise direction to get $V(C) = \{v_1, v_2, \ldots, v_n\}$ and $e_i = (v_i, v_{i+1})$ for $i = 1, \ldots, n-1$ and $e_n = (v_n, v_1)$. From here on, if an index $k > n$, we take the corresponding modulo n. For example, the vertex v_{n+1} coincides v_1 and so on. To simplify the denotation of vertex weight, we set $w_i := w(v_i)$. Let us denote by $\ell(C) := \sum_{i=1}^{n} \ell(e_i)$ the length of the cycle C. Also, we orentiate C to get two types of distances, the distance in the counter-clockwise direction d^L and distance in clockwise direction d^R. Then, the distance between two vertices v_i and v_j can be written as

$$d(v_i, v_j) := \min\left\{d^L(v_i, v_j), d^R(v_i, v_j)\right\}.$$

We call an edge $e_k = (v_k, v_{k+1})$ a critical edge w.r.t v_i if $d(v_i, v_k) = d^R(v_i, v_k)$ and $d(v_i, v_{k+1}) = d^L(v_i, v_{k+1})$. A point m_i, lying either on an edge or a vetex of C, is called the midpoint corresponding to v_i iff $d(v_i, m_i) = \frac{\ell(C)}{2}$. We further label $e = Emid(m_i)$ an edge that contains the midpoint m_i or incident to m_i in the case m_i is a vertex. Finding all midpoints m_i for $i = 1, \ldots, n$ and labeling them cost linear time based on the searching for corresponding critical edges.

To ease the readers, we can write the objective function as in the following

$$\textstyle\sum_{u\in V(C)} \sum_{v\in V(C), v\neq u} w(v)\tilde{d}(u, v) = \sum_{u\in V(C)} \widetilde{f}(u),$$

where $\widetilde{f}(u) := \sum_{v\in V(C)} w(v)\widetilde{d}(u, v)$. We next consider the reduction of an edge e contribute to the reduction of each $\widetilde{f}(u)$ for $u \in V(C)$.

Let us take into account an edge $e_k = (v_k, v_{k+1})$ with the new length $\widetilde{\ell}(e_k) = \ell(e_k) - \varepsilon$ for a small amount ε. For a vertex v_i, we consider the contribution of this vertex, say $\delta^{e_k}(v_i)$, to the objective function, i.e.,

$$\widetilde{f}(v_i) := f(v_i) - \delta^{e_k}(v_i)\varepsilon.$$

Assuming that $Emid(m_i) = e_l = (v_l, v_{l+1})$, we analyze the following cases.

1. If $e_l \equiv e_k$ or $m_i = v_k$ or $m_i = v_{k+1}$, we can compute the contribution of v_i to the objective according to the position of m_i. If $m_i \neq v_k$ and $m_i \neq v_{k+1}$, then the reduction of e_k does not effect the vertex v_i. We can set the contribution of v_i to the objective function $\delta^{e_k}(v_i) := 0$. If $m_i = v_k$, we obtain $\widetilde{d}(v_j, v_i) = d(v_j, v_i)$ for $j \neq k$ and $\widetilde{d}(v_k, v_i) = d(v_k, v_i) - w_k\varepsilon$. Thus, we can set $\delta^{e_k}(v_i) := w_k$. Similarly, if $m_i = v_{k+1}$, then $\delta^{e_k}(v_i) := w_{k+1}$.
2. If $\ell \geq k+1$ and $m_i \neq v_{k+1}$, then we know that $\widetilde{d}(v_j, v_i) = d(v_j, v_i)$ for $j \leq k$ and $\widetilde{d}(v_j, v_i) = d(v_j, v_i) - w_j\varepsilon$ for $j \geq l+1$. Therefore, $\delta^{e_k}(v_i) := \sum\limits_{j=k+1}^{\ell} w_j$.
3. If $l+1 \leq k$ and $m_i \neq v_k$, we can set $\delta^{e_k}(v_i) := \sum\limits_{j=l+1}^{k} w_j$.

Therefore, reducing the length of e_k by ε yields $\Lambda := \left(\sum_{i=1}^{n} \delta^{e_k}(v_i)\right)\varepsilon$, where Λ is the improvement of objective function (see Section 3). We call $\Delta(e_k) := \sum_{i=1}^{n} \delta^{e_k}(v_i)$ the efficiency of reducing the edge e_k.

Example 4.1. *Considering a cycle $C = (V(C), E(C))$ in Figure 2, we aim to compute the efficiency of reducing the edge $e_3 = (v_3, v_4)$.*

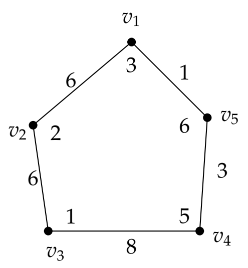

Figure 2: An instance of a cycle C

By elementary computations, one attains

$$\ell(C) = 24, m_1 \equiv v_3, m_2 \in (v_3, v_4), m_3 \equiv v_1, m_4 \in (v_2, v_3), m_5 \in (v_2, v_3).$$

The contribution of each vertex w.r.t the reduction of e_3 is calculated in Table 2. We finally obtain the efficiency of

i	1	2	3	4	5
$\delta^{e_3}(v_i)$	1	0	14	1	1

Table 2: The contribution of vertices w.r.t the reduction of e_3

reducing e_3 as $\Delta(e_3) := \sum_{i=1}^{5} \delta^{e_3}(v_i) = 17$.

We now consider the smallest reduction of an edge e_k such that the corresponding efficiency of reducing e_k changes. Observe that the efficiency $\delta(e_k)$ changes if one of midpoints of vertices in V changes its position in the previous case analysis. Let us focus on vertex v_i in the following cases.

1. If $m_i \equiv v_k$, we know that $\delta^{e_k}(v_i) = w_k$. The contribution of v_i w.r.t e_k changes if $m_i \equiv v_{k-1}$, i.e., $\tilde{d}^L(v_i, v_{k-1}) = \tilde{d}^R(v_i, v_{k-1})$. As $\tilde{d}^L(v_i, v_{k-1}) = d^L(v_i, v_{k-1})$ and $\tilde{d}^R(v_i, v_{k-1}) = d^R(v_i, v_{k-1}) - \varepsilon$, we obtain $\varepsilon = d^R(v_i, v_{k-1}) - d^L(v_i, v_{k-1}) = \ell(C) - 2d(v_i, v_{k-1})$. Then, we can recompute $\delta^{e_k}(v_i) = w_{k-1}$.
2. If $m_i \equiv v_{k+1}$, by the same argument in the previous case, if we reduce e_k by $\varepsilon = d^R(v_i, v_{k-1}) - d^L(v_i, v_{k-1}) = \ell(C) - 2d(v_i, v_{k+2})$ the contribution of v_i w.r.t e_k is $\delta^{e_k}(v_i) = w_{k+2}$.

3. If $l \geq k+1$, we know that $\delta^{e_k}(v_i) = \sum_{j=k+1}^{l} w_j$. The contribution of v_i w.r.t e_k changes if $m_i \equiv v_{l+1}$, i.e., $\tilde{d}^L(v_i, v_{l+1}) = \tilde{d}^R(v_i, v_{l+1})$. As $\tilde{d}^L(v_i, v_{l+1}) = d^L(v_i, v_{l+1}) - \varepsilon$ and $\tilde{d}^R(v_i, v_{l+1}) = d^R(v_i, v_{l+1})$, we obtain $\varepsilon = d^L(v_i, v_{l+1}) - d^R(v_i, v_{l+1}) = \ell(C) - 2d(v_i, v_{l+1})$. Then, $\delta^{e_k}(v_i) = \sum_{j=k+1}^{l+1} w_j$.
4. If $l+1 \leq k$, by the same argument in the previous case, if we reduce e_k by $\varepsilon = \ell(C) - 2d(v_i, v_l)$ the contribution of v_i w.r.t e_k is $\delta^{e_k}(v_i) := \sum_{j=l}^{k} w_j$.

For a vertex v_i with $m_i \in (v_l, v_{l+1})$, we set

$$\Gamma^{e_k}(v_i) := \begin{cases} \ell(C) - 2d(v_i, v_{k-1}) & \text{if } m_i \equiv v_k, \\ \ell(C) - 2d(v_i, v_{k+2}) & \text{if } m_i \equiv v_{k+1}, \\ \ell(C) - 2d(v_i, v_{l+1}) & \text{if } l \geq k+1, \\ \ell(C) - 2d(v_i, v_l) & \text{if } l+1 \leq k. \end{cases}$$

Lemma 4.2. *One of the contributions $\delta^{e_k}(v_i)$ for $i = 1, \ldots, n$ changes if we reduce the length of e_k at least*

$$\Gamma(e_k) := \min_{i=1}^{n} \{\Gamma^{e_k}(v_i)\}.$$

As $\Delta(e_k) = \sum_{i=1}^{n} \delta^{e_k}(v_i)$, then $\Delta(e_k)$ also changes if one of the contributions $\delta^{e_k}(v_i)$ for $i = 1, \ldots, n$ changes.

We now develop a combinatorial algorithm for the reverse total weighted distance problem on a cycle based on a greedy approach. The idea is to compute the efficiency for reducing each edge of the cycle and reduce the edge corresponding to the largest ratio of efficiency and cost. If the efficiency of the current edge changes by Lemma 4.2, we update the new efficiencies. Then, we continue with the edge corresponding to the largest efficiency. The algorithm stop if the budget is fully used. For detail, one observes Algorithm 2.

Algorithm 2 Solves the reverse total weighted distance problem on a cycle

```
Input: An instance of the problem on a cycle $C$.
Set $\mathcal{A} := \{e \in E : \bar{x}(e) > 0\}$.
Initialize the reduction of each edge $x(e) := 0$ for $e \in E$ and the reduction of the total weighted distance $\Lambda := 0$.
while $B > 0$ do
    Compute $\Delta(e)$ for all $e \in E$.
    Take $e^* := arg\max_{e \in \mathcal{A}} \{\Delta(e)/c(e) : e \in \mathcal{A}\}$.
    Reduce $e^*$ by $tempt(e^*) := \min_{i=1}^{n} \{B/c(e), \Gamma^{e^*}(v_i), \bar{x}(e^*)\}$.
    Set $x(e^*) := x(e^*) + tempt(e^*)$ and set $\bar{x}(e^*) := \bar{x}(e^*) - tempt(e^*)$.
    if $\bar{x}(e^*) = 0$ then
        Set $\mathcal{A} := \mathcal{A} \backslash \{e^*\}$.
    end if
    Set $Val := Val + \Delta(e^*) tempt(e^*)$ and $B := B - c(e) tempt(e^*)$.
end while
Output: Optimal solution $\{x(e)\}_{e \in E}$ and maximum reduction $\Lambda$.
```

We illustrate Algorithm 2 by the following example.

Example 4.3. *(Example 4.1 cont'd) We consider a cycle C in Example 4.1. Furthermore, the upper bound of edge length modifications are in Table 3. The budget is $B := 7$ and the cost coefficients are uniform, i.e., $c(e) = 1$ for $e \in E(C)$. After setting $\mathcal{A} = \{e_1, e_2, e_3, e_4\}$, $x(e) := 0$ for all $e \in E(C)$ and $\Lambda := 0$, we solve the problem in the following iterations.*

Iter. 1. *We compute efficiency for reducing each edge in $\mathcal{A}$; see Table 4. After computing all $\Delta(e_i)$ for $i = 1, \ldots, n$, we get $e_4 := arg\max_{e \in \mathcal{A}} \{\Delta(e)/c(e) : e \in \mathcal{A}\}$ and $tempt(e_4) := 1$. We also update $\mathcal{A} := \{e_1, e_2, e_3\}$, $\Lambda := 30$, and $B = 6$.*

i	1	2	3	4	5
$\bar{x}(e_i)$	3	4	6	1	0

Table 3: Modifying bounds of edges

i	1	2	3	4
$\Delta(e_i)$	24	7	17	30

Table 4: Efficiency for reducing each edge in $\mathcal{A}$

Iter. 2. *We recompute efficiency for reducing each edge in* $\mathcal{A}$ *as* $\Delta(e_1) = 20$, $\Delta(e_2) = 3$, $\Delta(e_3) = 17$. *We get* $e_1 := arg\max_{e\in\mathcal{A}}\{\Delta(e)/c(e) : e \in \mathcal{A}\}$ *and* $tempt(e_1) := 1$. *We also update* $\bar{x}(e_1) = 2$, $\mathcal{A} := \{e_1, e_2, e_3\}$, $\Lambda := 50$, *and* $B = 5$.

Iter. 3. *We recompute efficiency for reducing each edge in* $\mathcal{A}$ *as* $\Delta(e_1) = 24$, $\Delta(e_2) = 7$, $\Delta(e_3) = 17$. *We get* $e_1 := arg\max_{e\in\mathcal{A}}\{\Delta(e)/c(e) : e \in \mathcal{A}\}$ *and* $tempt(e_1) := 2$. *We also update* $\bar{x}(e_1) := 0$, $\mathcal{A} := \{e_2, e_3\}$, $\Lambda := 98$, *and* $B := 3$.

Iter. 4. *We recompute efficiency for reducing each edge in* $\mathcal{A}$ *as* $\Delta(e_2) = 4$, $\Delta(e_3) = 14$. *We get* $e_3 := arg\max_{e\in\mathcal{A}}\{\Delta(e)/c(e) : e \in \mathcal{A}\}$ *and* $tempt(e_1) := 3$. *We also update* $\bar{x}(e_1) = 1$, $\mathcal{A} := \{e_2, e_3\}$, $\Lambda := 140$, *and* $B = 0$. *We stop the algorithm.*

The optimal solution is $x(e_1) := 3$, $x(e_2) := 3$, $x(e_3) := 0$, $x(e_4) := 1$, $x(e_5) := 0$ *and the optimal objective is* $\Lambda := 140$.

The exact complexity of the iterative as in Algorithm 2 is still unknown to us. However, for the cycle with uniform edge lengths, i.e., all lentghs in the cycle are equal, we can derive a nice property as follows.

Lemma 4.4. *In each iteration of Algorithm 2, we obtain* $\bar{x}(e^*) < \Gamma^{e^*}(v_i)$ *for all* $i = 1, \ldots, n$.

Proof. Without loss of generality, we can assume that the midpoint m_i corresponding to v_i is in the edge (v_{p-1}, v_p) and we reduce an edge e_k with $k \leq p-1$. According to the four cases, we know that the midpoint corresponding to v_i coincides with a vertex v_p if $\Gamma^{e^*}(v_i) = \ell(C) - 2d(v_i, v_p)$ for $p \neq i$. Moreover, we can write

$$\ell(C) - 2d(v_i, v_p) := \ell(C) - 2(d(v_i, m_i) - d(m_i, v_p)) = 2d(m_i, v_p).$$

As the cycle has equal edge lengths, the midpoint m_i is either v_{p-1} or the midpoint of the edge (m_{p-1}, m_p). We then get $2d(m_i, v_p) \geq \ell(e_k) > \bar{x}(e_k)$. Therefore, the inequality $\bar{x}(e^*) < \Gamma^{e^*}(v_i)$ holds for $i = 1, \ldots, n$. □

Let us now consider an instance of the reverse total weighted distance problem on cycles with uniform edge lengths. In each iteration, we recompute the efficiency of each edge of the tree in $O(n^2)$ time. Furthermore, by Lemma 4.4, an edge e can be reduced at most once in Algorithm 2. Therefore, the total complexity of the algorithm in this special case is $O(n^3)$ time.

Theorem 4.5. *The reverse total weighted distance problem on a cycle with uniform edge lengths can be solved in* $O(n^3)$ *time, where* n *is the number of vertices.*

5. Conclusion

We consider the problem on reducing the edge lengths of a network in order to reduce its total weighted distance as much as possible within a given budget. It is shown that the problem can be applied to enhance the overall interaction between vertices of a network. For the problem on general graphs, we obtain the NP-hardness result. Furthermore, we develop efficient combinatorial algorithm that solves the problem on trees in linear time. For the problem on cycles, we devise a greedy type algorithm. Furthermore, the algorithm runs in $O(n^3)$ time if the underlying cycle has equal edge lengths, where n is the number of vertices.

Acknowledgements: *The authors would like to thank the anonymous referees for valuable comments which help to improve the presentation of this paper.*

References

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

[2] Alizadeh, B., Etemad, R.: The linear time optimal approaches for reverse obnoxious center location problems on networks. Optimization **65**: 2025-2036 (2016).

[3] Ahuja, R.K., Orlin, J.B.: Inverse optimization. Operations Research **49**, 771-783 (2001).

[4] Ahuja, R.K., Orlin, J.B.: Combinatorial algorithms for inverse network flow problems. Networks **40**, 181-187 (2002).

[5] Balas, E., Zemel, E.: An algorithm for large zero-one knapsack problems. Operations Research **28**, 1130-1154 (1980).

[6] Berman, O., lngco, D.I., Odoni, A.: Improving the location of minisum facilities through network modification. Ann. Operations Res. **40**, 1-16, (1992).

[7] Berman, O., lngco, D.I., Odoni, A.: Improving the location of minimax facilities through network modification. Networks **24**, 31-41 (1994).

[8] D. Burton, Inverse shortest path problem (PhD thesis). FUNDP, Namur (1993).

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

[10] Burkard, R.E., Pleschiutschnig, C., Zhang, J.Z.: The inverse 1-median problem on a cycle, Discrete Optimization **5**: 242-253 (2008).

[11] Burkard, R.E., Gassner, E., Hatzl, J.: A linear time algorithm for the reverse 1-median problem on a cycle. Networks **48**, 16-23 (2006).

[12] Burkard, R.E., Gassner, E., Hatzl, J.: Reverse 2-median problem on trees. Discrete Appl. Math. **156**, 1963-1976 (2008).

[13] Etemad, R., Alizadeh, B.: Reverse selective obnoxious center location problems on tree graphs, Mathematical Methods of Operations Research **87**, 431-450 (2018).

[14] Heuberger, C.: Inverse combinatorial optimization: a survey on problems, methods, and results. J Comb Optim **8**, 329-361 (2004).

[15] Hochbaum, D.S.: Efficient algorithms for the inverse spanning-tree problem. Oper Res **51**, 785-797 (2003).

[16] Garey, M.R, Johnson, D.S.: Computers and intractability: A guide to the theory of *NP*-completeness. W. H. Freeman and Co., Sanfrancisco, CA, 1979.

[17] Nguyen, K.T.: Some polynomially solvable cases of the inverse ordered 1-median problem on trees. Filomat **31**:3651-3664, (2017).

[18] Nguyen, K.T.: Reverse 1-center problem on weighted trees. Optimization **65**, 253-264 (2015).

[19] Nguyen K.T., Chassein, A.: Inverse eccentric vertex problem on networks. Cent. Eur. J. Oper. Res. **23**, 687-698 (2014).

[20] Nguyen, K.T., Chassein, A.: The inverse convex ordered 1-median problem on trees under Chebyshev norm and Hamming distance. European Journal of Operational Research **247**: 774-781 (2015).

[21] Nguyen, K.T., Hung, N.T, Nguyen-Thu, H., Le, T.T., Pham, V.H.: On some inverse 1-center location problems. Optimization **68**, 999-1015 (2019).

[22] Sepasian, A.R.: Reverse 1-maxian problem with keeping existing 1-median. OPSEARCH **56**: 1-13(2019).

[23] Wang, Q.,Yanqin Bai, Y.: An efficient algorithm for reverse 2-median problem on a cycle. Jornal of Networks **5**: 1169-1176 (2010).

[24] Zhang, J., Liu, Z., Ma, Z.: Some reverse location problems, Eur. J. Oper. Res. **124**, 77-88 (2000).

[25] Zhang, J., Yang, X., Cai, M.C.: Reverse center location problem. In International Symposium on Algorithms and Computation, pages 279-294 (1999).