

# On the computational complexity of the Steiner $k$ -eccentricity

Xingfu Li<sup>a</sup>, Guihai Yu<sup>a</sup>, Aleksandar Ilić<sup>b</sup>, Sandi Klavžar<sup>c,d,e,†</sup>

<sup>a</sup>College of Big Data Statistics, Guizhou University of Finance and Economics  
Guiyang, Guizhou, 550025, China.

E-mail: xingfuli@mail.gufe.edu.cn; yuguihai@126.com

<sup>b</sup>Facebook Inc, Menlo Park 94025, California, USA

E-mail: aleksandari@gmail.com

<sup>c</sup> Faculty of Mathematics and Physics, University of Ljubljana, Slovenia

<sup>d</sup> Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

<sup>e</sup> Faculty of Natural Sciences and Mathematics, University of Maribor, Slovenia

sandi.klavzar@fmf.uni-lj.si

December 3, 2021

## Abstract

The Steiner  $k$ -eccentricity of a vertex  $v$  of a graph  $G$  is the maximum Steiner distance over all  $k$ -subsets of  $V(G)$  which contain  $v$ . A linear time algorithm for calculating the Steiner  $k$ -eccentricity of a vertex on block graphs is presented. For general graphs, an  $O(n^{\nu(G)+1}(n(G) + m(G) + k))$  algorithm is designed, where  $\nu(G)$  is the cyclomatic number of  $G$ . A linear algorithm for computing the Steiner 3-eccentricities of all vertices of a tree is also presented which improves the quadratic algorithm from [Discrete Appl. Math. 304 (2021) 181–195].

**Key words:** Steiner tree; Steiner  $k$ -eccentricity; block graph; tree; algorithm; computational complexity

## 1 Introduction

Every graph  $G = (V(G), E(G))$  in this paper is simple and undirected. The order of  $G$  will be denoted by  $n(G)$  and the size of  $G$  by  $m(G)$ . The *cyclomatic number*,  $\nu(G)$ , of  $G$  is the minimum number of edges of  $G$  whose removal makes  $G$  acyclic. If  $G$  is connected, then  $\nu(G) = m(G) - n(G) + 1$ . (The cyclomatic number of  $G$  can alternatively be defined as the dimension of its cycle space.) If every block of  $G$  is a clique, then  $G$  is a *block graph*. The *distance*  $d_G(u, v)$  between vertices  $u$  and  $v$  in  $G$  is the length of a shortest  $u, v$ -path.

The *eccentricity*  $\text{ecc}_G(v)$  of a vertex  $v$  in  $G$  is the maximum distance between  $v$  and all the other vertices of  $G$ . We refer to [3–8, 14, 19, 20] for different investigations of the

eccentricity. In this work we study a generalization of the eccentricity, the Steiner  $k$ -eccentricity. Its definition is based on Steiner trees which are in turn defined as follows. If  $S \subseteq V(G)$ , then a subgraph  $T$  of  $G$  is a *Steiner  $S$ -tree*, if  $T$  is a minimum connected subgraph of  $G$  which spans all vertices from  $S$ . Every vertex from  $S$  is called a *terminal* of  $T$ , and the set  $S$  is the *terminal set* of  $T$ . The *Steiner  $k$ -eccentricity*,  $\text{ecc}_k(v, G)$ , of a vertex  $v$  is the maximum size over all Steiner  $S$ -trees, where  $|S| = k$  and  $v \in S$ , that is,

$$\text{ecc}_k(v, G) = \max_{\substack{S \subseteq V(G) \\ |S|=k, v \in S}} \{m(T) : T \text{ is a Steiner } S\text{-tree}\}.$$

(For additional aspects of the Steiner distance see [12, 13, 15, 18, 22].) Note that  $\text{ecc}_G(v) = \text{ecc}_2(v, G)$ . A Steiner  $S$ -tree  $T$  that realizes  $\text{ecc}_k(v, G)$  is a *Steiner  $k$ -eccentricity tree* of  $v$ , we will shortly say that  $T$  is a *Steiner  $k$ -ecc  $v$ -tree*. The  $k$ -set  $S$  corresponding the the Steiner  $k$ -ecc  $v$ -tree is a *Steiner  $k$ -ecc  $v$ -set* in  $G$ . The problem to find a Steiner  $k$ -eccentricity tree of a given vertex is referred to as the *Steiner  $k$ -eccentricity tree* ( $k$ -ST) problem. The decision version of the *Steiner  $k$ -eccentricity tree problem* ( $k$ -ST) is presented in Table 1.

Table 1: The Steiner  $k$ -eccentricity tree problem ( $k$ -ST)

Instance: Graph $G$ , $v \in V(G)$ , $k \in [n(G)]$ , constant $c$ .
Question: Is there a Steiner $S$ -tree $T$ , where $ S  = k$ and $v \in S$ , such that $m(T) \geq c$ ?

The minimum Steiner tree problem is a well-known NP-hard problem [11], but the hardness of the  $k$ -ST problem is still unknown. In [16], a linear time algorithm was designed to find the optimal value of the 3-ST problem on trees, while in [17] the result was extended to the  $k$ -ST problem. In the following section we design a linear time algorithm for the  $k$ -ST problem on block graphs. In the subsequent section we present an algorithm for the problem on general graphs with the time complexity  $O(n^{\nu(G)+1}(n(G) + m(G) + k))$ . In Section 4 we present a linear algorithm to calculate the Steiner 3-eccentricity for all vertices in a weighted tree. This improves the corresponding quadratic algorithm for (unweighted) trees from [16]. We conclude the paper by giving several directions for future work.

## 2 A linear algorithm for block graphs

In this section, we devise a linear-time algorithm to solve the  $k$ -ST problem on block graphs. The main idea is to reduce the problem from a block graph  $G$  to a special spanning tree  $T$  such that the equality  $\text{ecc}_k(v, G) = \text{ecc}_k(v, T)$  holds, and then to invoke the algorithm from [17].

Let  $G$  be a block graph. If  $v \in V(G)$  and  $B$  is a block of  $G$ , then let  $\text{Near}_G(v, B)$  be a nearest vertex to  $v$  in the block  $B$ , see Fig. 1 for an example. We first observe that  $\text{Near}_G(v, B)$  is unique.

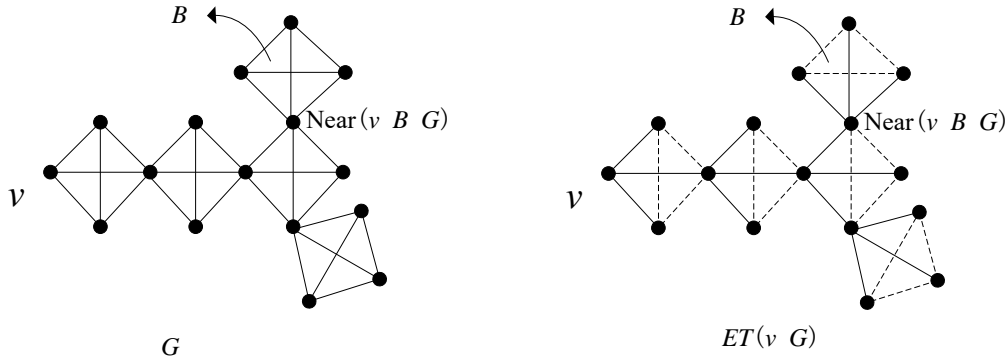


Figure 1: Block graph  $G$  (left) and  $T(v, G)$  (right). The dashed lines indicate the edges which must be removed from  $G$  to construct  $T(v, G)$ .

**Property 2.1** *If  $G$  is a block graph,  $v \in V(G)$ , and  $B$  a block of  $G$ , then  $\text{Near}_G(v, B)$  is unique.*

**Proof.** If  $v \in V(B)$ , then clearly  $\text{Near}_G(v, B) = v$  is unique. If  $v \notin V(B)$ , then let  $P$  be a shortest path between  $v$  and the block  $B$ . Then the end-vertex  $x$  of  $P$ ,  $x \neq v$ , is a cut vertex of  $B$  which in turn implies that  $\text{Near}_G(v, B) = x$  is again unique. ■

Let  $v$  be a vertex of a block graph  $G$ . For every block  $B$  of  $G$  remove every edge which is not incident with  $\text{Near}_G(v, B)$  and denote the resulting graph by  $T(v, G)$ , for an example see Fig. 1 again. Considering shortest paths between  $v$  and the cut vertices of  $G$  we infer that  $T(v, G)$  is connected. Moreover, it is also clear (having in mind that  $G$  is a block graph) that  $T(v, G)$  has no cycle. Hence  $T(v, G)$  is a spanning tree of  $G$ . In addition, with Property 2.1 in hands it immediately follows from the construction that  $T(v, G)$  is unique for each vertex  $v$  of a block graph  $G$ .

We are now ready for the key result needed for our algorithm for block graphs.

**Theorem 2.2** *Let  $v$  be a vertex of a block graph  $G$ . Then a Steiner  $k$ -ecc  $v$ -tree in  $T(v, G)$  is also a Steiner  $k$ -ecc  $v$ -tree in  $G$ .*

**Proof.** Let  $T_1$  be a Steiner  $k$ -ecc  $v$ -tree in  $T(v, G)$  and suppose on the contrary that  $T_1$  is not a Steiner  $k$ -ecc  $v$ -tree in  $G$ . Let  $T_2$  be a Steiner  $k$ -ecc  $v$ -tree in  $G$ . Then we have  $m(T_1) \neq m(T_2)$ . Moreover, since  $T_1$  is also a subtree of  $G$ , we must have  $m(T_1) < m(T_2)$ . We are now going to construct a tree  $T'_2$  of  $T(v, G)$  with  $m(T'_2) > m(T_1)$ , which will contradict the fact that  $T_1$  is a Steiner  $k$ -ecc  $v$ -tree in  $T(v, G)$ .

Construct the tree  $T'_2$  from  $T_2$  through the following procedure. For every block  $B$  of  $G$ , if there is an edge  $e \in E(B) \cap E(T_2)$  such that neither endpoint of  $e$  is the vertex  $\text{Near}_G(v, B)$ , then delete the edge  $e$  from  $T_2$ , and add an edge between one endpoint of  $e$  and  $\text{Near}_G(v, B)$ . After finishing the whole procedure for all blocks of  $G$ , the tree  $T'_2$  is

constructed. Since the edge deletion and addition occur pairwise,  $m(T_2') = m(T_2)$ . Since  $m(T_1) < m(T_2)$ , we have the announced contradiction  $m(T_1) < m(T_2')$ . ■

Theorem 2.2 directly leads to Algorithm 1.

---

**Algorithm 1:** k-ECC-Block( $v, G, k$ )

---

**Input:** Block graph  $G$ , vertex  $v \in V(G)$ , an integer  $k \geq 3$ .

**Output:**  $\text{ecc}_k(v, G)$ .

- 1 Determine  $T(v, G)$ ;
  - 2 Return  $\text{ecc}_k(v, T(v, G))$ ;
- 

**Theorem 2.3** *If  $G$  is a block graph and  $v \in V(G)$ , then Algorithm 1 computes  $\text{ecc}_k(v, G)$  and can be implemented to run in  $O(k(n(G) + m(G)))$  time.*

**Proof.** The correctness of the algorithm follows from Theorem 2.2.

Since  $T = T(v, G)$  is a tree, Step 2 can be implemented in  $O(k(n(T) + m(T)))$  time by invoking the corresponding algorithm from [17]. As for Step 1, to determine  $T(v, G)$  efficiently, Algorithm 2 modifies the depth-first search (DFS) algorithm, and runs in linear time. ■

---

**Algorithm 2:** Get-Tree( $v, G$ )

---

**Input:** Block graph  $G$ , vertex  $v \in V(G)$ .

**Output:**  $T(v, G)$ .

- 1 Mark all vertices as 'unvisited', and mark the vertex  $v$  as 'visited';
  - 2 **for** each unvisited vertex  $u \in N_G(v)$  **do**
  - 3     **for** each vertex  $w \in N_G(u)$  **do**
  - 4         **if**  $wv \in E(G)$  **then**
  - 5             Delete  $uw$  from  $G$ ;
  - 6         **end**
  - 7     **end**
  - 8     Get-Tree( $u, G$ );
  - 9 **end**
  - 10 **return**  $G$ ;
- 

### 3 On general graphs

The basic property that allows a fast algorithm for calculating the Steiner  $k$ -eccentricity of a vertex in a tree is that every Steiner  $k$ -ecc  $v$ -tree contains a Steiner  $(k - 1)$ -ecc  $v$ -tree [17]. This property does not hold in general graphs. In fact, as the example from Fig. 2 demonstrates, the property does not hold even on unicycle graphs.

This example illustrates that it could be difficult to find a property that would lead to a fast algorithm for calculating the Steiner  $k$ -eccentricity of a vertex on general graphs. In

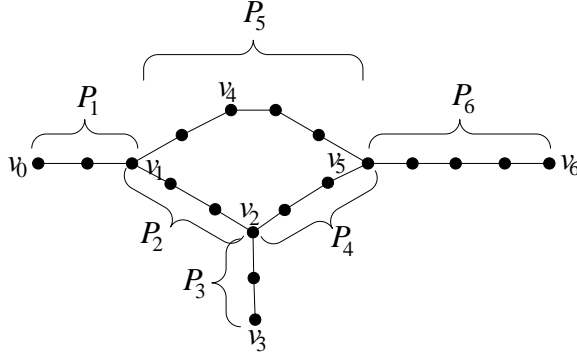


Figure 2: The Steiner 3-ecc tree of the vertex  $v_0$  is formed by the paths  $P_1$ ,  $P_2$ ,  $P_3$ ,  $P_4$ , and  $P_6$ . The Steiner 2-ecc tree of  $v_0$  is formed by the paths  $P_1$ ,  $P_5$  and  $P_6$ . Hence the Steiner 3-ecc  $v_0$ -tree does not contain a Steiner 2-ecc  $v_0$ -tree.

this section, we present two algorithms to solve the  $k$ -ST problem in general graphs. The first one is a brute-forced method, the other one reduces the problem from general graphs to trees. The running time of the brute-forced method grows exponentially with  $k$ , while the running time of the latter algorithm grows exponentially in  $\nu(G)$ .

### 3.1 Brute-force algorithm

The definition of the Steiner  $k$ -eccentricity of a vertex leads to a direct, brute-force algorithm as follows. Let  $v$  be a vertex for which we are going to calculate the Steiner  $k$ -eccentricity in a graph  $G$ . Initially, enumerate all  $(k - 1)$ -subsets  $S$  in  $V(G) \setminus \{v\}$ . For each of these sets  $S$  then invoke an algorithm to find a minimum Steiner tree for the set  $S \cup \{v\}$ . Finally, choose the maximum size among all these minimum Steiner trees.

The Steiner tree problem is a well-known NP-hard problem [11]. Erickson, Monma, and Veinott [9] designed an exact algorithm for the Steiner tree problem with running time  $O(3^k n + 2^k(m + n \log n))$ , where  $n = n(G)$ ,  $m = m(G)$ , and  $k$  is the number of terminals. Fuchs et al. [10] and Vygen [21] followed by exact algorithms with running times  $O(2^{k+(k/2)^{1/3}(\ln n)^{2/3}})$  and  $O(nk2^{k+\log_2 k \log_2 n})$ , respectively. Note that the running time of all these algorithms grows exponentially with the number of terminals. So the running time of the brute-forced method also grows exponentially with respect to the number of terminals.

### 3.2 Reducing to trees

We now devise a novel algorithm whose running time does not grow exponentially on the number of terminal, that is, on  $k$ . The key for the algorithm is the following result, where  $\mathcal{T}(G)$  denotes the set of all spanning trees of a connected graph  $G$ .

**Theorem 3.1** *If  $G$  is a connected graph, then*

$$\text{ecc}_k(v, G) = \min\{\text{ecc}_k(v, T) : T \in \mathcal{T}(G)\}. \quad (1)$$

**Proof.** Let  $G$  be a graph,  $v \in V(G)$ , and  $T$  s spanning tree of  $G$ . Then we claim that the size of a Steiner  $k$ -ecc  $v$ -tree in  $G$  is not larger than the size of a Steiner  $k$ -ecc  $v$ -tree in  $T$ .

Suppose on the contrary that there is a Steiner  $k$ -ecc  $v$ -tree  $T_1$  in  $T$  such that  $m(T_1)$  is less than the size of a Steiner  $k$ -ecc  $v$ -tree in  $G$ . Let  $T_v$  be a Steiner  $k$ -ecc  $v$ -tree in  $G$  and  $S_v$  be the corresponding Steiner  $k$ -ecc  $v$ -set. Then we have  $m(T_1) < m(T_v)$ . Since  $T$  is a spanning tree of  $G$ , we clearly have  $S_v \subseteq V(T)$ . Moreover, the size of the minimum Steiner tree on the set  $S_v$  in  $T$  is not less than the size of  $T_v$ . Let  $T_2$  be a minimum Steiner tree on the set  $S_v$  in  $T$ . Then we have  $m(T_v) \leq m(T_2)$  and therefore,  $m(T_1) < m(T_2)$ . This contradicts to the assumption that  $T_1$  is a Steiner  $k$ -ecc  $v$ -tree in  $T$ , hence the claim is proved.

From the claim it now follows that the value of  $\text{ecc}_k(v, G)$  is equal to  $\min\{\text{ecc}_k(v, T) : T \in \mathcal{T}(G)\}$ . ■

In order to determine the Steiner  $k$ -eccentricity of a vertex in a graph  $G$ , Theorem 3.1 says that it suffices to calculate the Steiner  $k$ -eccentricity of the vertex in every spanning tree. In our algorithm we enumerate all possible edge sets of size  $\nu(G)$  rather than enumerating all spanning trees. Moreover, we calculate the Steiner  $k$ -eccentricity of a vertex  $v$  in a spanning tree as soon as the spanning tree is enumerated, and maintain the maximum Steiner  $k$ -eccentricity of  $v$  over all currently enumerated spanning trees. The enumerating method is based on the following recursive equation. (Recall that if a graph  $G$  is a tree itself, then one can invoke the linear time algorithm from [17] to calculate the Steiner  $k$ -eccentricity of  $v$ .)

$$\text{ecc}_k(v, G) = \begin{cases} \text{ecc}_k(v, G); & G \text{ is a tree,} \\ \min_{e \in E(C)} \{\text{ecc}_k(v, G - e)\}; & \text{otherwise,} \end{cases} \quad (2)$$

where  $C$  is a cycle of  $G$ . The whole procedure is summarized in Algorithm 3, where the parameter *current-opt* is initialized to be zero and used to store the currently maximum Steiner  $k$ -eccentricity.

---

**Algorithm 3:** k-ECC( $v, G, k, \text{current-opt}$ )

---

**Input:** Graph  $G$ ,  $v \in V(G)$ , integer  $k \geq 3$ .

**Output:** The Steiner  $k$ -eccentricity of  $v$  in  $G$ .

```
1 if  $G$  is a tree then
2   |  $\text{temp} \leftarrow$  Steiner  $k$ -eccentricity of  $G$ ;
3   | if  $\text{temp} < \text{current-opt}$  then
4   |   |  $\text{current-opt} \leftarrow \text{temp}$ 
5   |   | end
6   |   | return  $\text{current-opt}$ ;
7 end
8 else
9   |  $C \leftarrow$  simple cycle of  $G$ ;
10  | for each edge  $e$  of  $C$  do
11  |   |  $H \leftarrow G - e$ ;
12  |   | k-ECC( $v, H, k, \text{current-opt}$ );
13  |   | end
14 end
```

---

**Theorem 3.2** *Let  $G$  be a connected graph and  $v \in V(G)$ . Then Algorithm 3 calculates the Steiner  $k$ -eccentricity of  $v$  and can be implemented to run in  $O(n^{\nu(G)+1}(n(G) + m(G) + k))$  time.*

**Proof.** The correctness of the algorithm follows from Theorem 3.1.

For the time complexity, we first note that in Step 2 we can apply the linear algorithm from [17] and that for Step 9 we can use the BFS algorithm starting from  $v$ .

For the rest of the proof set  $s = \nu(G)$ ,  $n = n(G)$ , and  $m = m(G)$ . Let  $T(v, s)$  be the running time of Algorithm 3. By (2) we have

$$T(v, s) = \begin{cases} O(kn); & s = 0, \\ O(n + m) + \ell_1 * T(v, s - 1); & s > 0, \end{cases} \quad (3)$$

where  $\ell_1 = m(C)$ . Setting  $M = n + m$ , we can argue as follows:

$$\begin{aligned} T(v, s) &= O(M) + \ell_1 * T(v, s - 1) \\ &= O(M)(1 + \ell_1 + \ell_1 * \ell_2 + \dots + \prod_{i=1}^s \ell_i) + (\prod_{i=1}^s \ell_i) * T(v, 0) \\ &= O(M)(1 + O(n) + O(n) * O(n) + \dots + \prod_{i=1}^s O(n)) + (\prod_{i=1}^s O(n)) * O(kn) \\ &= O(n^{s+1}(n + m + k)), \end{aligned}$$

and we are done. ■

Note that the time complexity of Algorithm 3 grows exponentially with  $\nu(G)$  rather than with  $k$ .

## 4 Linear time to calculate Steiner 3-eccentricities for all vertices

As already mentioned, in [16] a linear time algorithm to calculate the Steiner 3-eccentricity of a vertex of a tree was designed. In case one wishes to determine the Steiner 3-eccentricity of all vertices, for instance in order to compute the average Steiner 3-eccentricity, then this approach yields a quadratic algorithm. In this section we demonstrate that also the Steiner 3-eccentricity of all vertices of a tree can be computed in linear time. Moreover, we also extend this result to weighted trees.

Let  $T$  and  $T_r$ , respectively, be a rooted weighted tree on  $n$  vertices and the subtree rooted at a vertex  $r \in V$ . In other words,  $T_r$  is a subgraph induced on vertex  $r$  and all of its descendants.

The root of  $T$  is assigned by an arbitrary vertex. The weights of edges are stored in an adjacency list named as  $adj$ . The linear algorithm is two-stage DFS procedures. Details for the DFS algorithm can be found in [1]. The first stage is to compute the longest paths from  $v$  in the subtree rooted at  $v$  for every vertex  $v$ , while the second one is to update the longest paths with the upwards path via parent node for the purpose of computing Steiner 3-eccentricities. The first stage and the second stage are respectively showed in Algorithms 4 and 5.

---

### Algorithm 4: DFS\_stage1 ( $v$ )

---

**Input:** The adjacency matrix of the tree  $T$  with the root vertex  $root$ .

**Output:** The downwards arrays  $path\_weight$ ,  $path\_index$ ,  $attached\_weight$ .

$path\_weight[v] = (0, 0, 0);$

$path\_index[v] = (-1, -1, -1);$

$attached\_weight[v] = (0, 0, 0);$

**for every neighbor  $u$  of  $v$  do**

**if**  $parent[u] = -1$  **and**  $u \neq root$  **then**

$parent[u] = v;$

$DFS\_stage1(u);$

$update(v, u, adj\_weight[v][u] +$

$path\_weight[0][u], \max(path\_weight[1][u], attached\_weight[0][u]));$

**end**

**end**

---



---

**Algorithm 5:** DFS\_stage2 ( $v$ )

---

**Input:** The outputs of  $DFS\_stage1$ .

**Output:** The arrays  $path\_weight, path\_index, attached\_weight$ .

```
mark[v] = 1;
u = parent[v];
if u  $\neq$  -1 then
    up_path_weight = 0;
    up_attached_weight = 0;
    if path_index[0][u]  $\neq$  v then
        up_path_weight = adj_weight[u][v] + path_weight[0][u];
        if path_index[1][u]  $\neq$  v then
            up_attached_weight = max(path_weight[1][u], attached_weight[0][u]);
        end
        else
            up_attached_weight = max(path_weight[2][u], attached_weight[0][u]);
        end
    end
    else
        up_path_weight = adj_weight[u][v] + path_weight[1][u];
        up_attached_weight = max(path_weight[2][u], attached_weight[1][u]);
    end
    update(v, u, up_path_weight, up_attached_weight);
end
for every neighbor u of v do
    if mark[u] = -1 then
        DFS_stage2(u);
    end
end
```

---

The Steiner 3-eccentricity of the vertex  $v$  can be computed as

$$\epsilon_3(v) = path\_weight[v][0] + \max\{path\_weight[v][1], attached\_weight[v][0]\},$$

where  $path\_weight[v]$  is initialized as  $(0, 0, 0)$  and represents the length of the longest path from the vertex  $v$  in the subtree  $T_v$ ;  $path\_index[v]$  is initialized as  $(-1, -1, -1)$  and represents the neighbor of  $v$  on the longest path from the vertex  $v$  in the subtree  $T_v$ ;  $attached\_weight[v]$  is initialized as  $(0, 0, 0)$  and represents the length of the longest subpath attached at the longest path in the subtree  $T_v$  strictly below  $v$ .

The main helper function  $update(v, u, new\_weight, new\_attached\_weight)$  is to keep the top three values for the longest paths coming from the vertex  $v$  and implemented by the C code in Appendix.

The first stage *DFS\_stage1* is to recursively traverse all neighbors of the vertex  $v$ . After the subtree is processed, we update the values for the node  $v$  based on the values for each of the subtrees. We traverse the nodes in the pre-order phase in the second stage *DFS\_stage2*. We update the values for the root and then run computation for the neighbors.

**Theorem 4.1** *There is a linear-time algorithm to calculate the Steiner 3-eccentricities for all vertices in a weighted tree  $T$ .*

**Proof.** Given that the algorithm consists of two traversals using DFS algorithms and with linear initialization, the time complexity of the algorithm equals  $O(n)$ . There are six additional vectors of size  $n$ , and therefore the memory complexity of the algorithm equals  $O(n)$  as well.

The correctness of the algorithm directly follows from the definition of Steiner 3-eccentricities for the root vertex  $v$ . For other vertices, we effectively compute the top three longest paths and attached paths - which is equivalent as considering those vertices as roots. ■

## 5 Conclusion

We presented a linear-time algorithm to solve the  $k$ -ST problem on block graphs, and an algorithm to solve the  $k$ -ST problem on general graphs, where the exponential growth of the running time depends only on the cyclomatic number of a graph.

It seems that if a graph is dense, then the Steiner  $k$ -eccentricity of a vertex may be easy to find. For instance, this is the case for complete graphs and for complete graphs with one or two edges removed. Inspired by this, an open question is how to modify Algorithm 3 so that it works well not only for small values of  $\nu$  but also when  $\nu$  is large.

For calculating the Steiner  $k$ -eccentricity of a vertex in a graph  $G$ , we showed that there is an algorithm whose running time grows exponentially on  $\mu$  but is independent of the input parameter  $k$ . On the other hand, is there is a fixed-parameter tractable algorithm [2] to solve this problem? Moreover, there is still no answer to the question of whether the problem is NP-hard.

Let's turn back to the  $k$ -ST problem. One can also ask whether there is a  $k$ -set  $S$  such that the size of the minimum Steiner tree on  $S$  is at least  $c$ . This yields the *Steiner  $k$ -eccentricity set problem* ( $k$ -SES). The decision version of the *Steiner  $k$ -eccentricity set problem* ( $k$ -SES) is presented in Table 2.

For every vertex  $v$  in a graph  $G$ , there is a  $k$ -set  $S_1$  such that a minimum Steiner tree on  $S_1$  has at least  $c$  edges if and only if there is a minimum Steiner tree on some  $k$ -set  $S_2$  such that the size of the Steiner tree is at least  $c$ , where  $v \in S_1$  and  $v \in S_2$ . In other words, there is a "YES" answer to the  $k$ -ST problem if and only if there is a "YES" answer to the  $k$ -SES problem. Therefore, the  $k$ -ST problem is as hard as the  $k$ -SES problem.

Table 2: The Steiner  $k$ -eccentricity set problem ( $k$ -SES)

Instance: Graph $G$ , $v \in V(G)$ , $k \in [n(G)]$ , constant $c$ .
Question: Is there a $k$ -set $S$ , where $v \in S$ , such that the size of a minimum Steiner tree on $S$ is at least $c$ ?

However, algorithms to solve these two problems could be different. It seems that there is no brute-force algorithm to solve the  $k$ -SES problem.

Finally, we designed an  $O(n)$ -time algorithm to calculate Steiner 3-eccentricities for all vertices of a tree. Does this result extend to  $k \geq 4$ ?

## Acknowledgements

This work was supported by Science Foundation of Guizhou University of Finance and Economics (2020XYB16), Science Foundation of Guizhou University of Finance and Economics (2019YJ058). Sandi Klavžar acknowledges the financial support from the Slovenian Research Agency (research core funding P1-0297, and projects N1-0095, J1-1693, J1-2452).

## References

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Third Edition, MIT Press, Cambridge, 2009.
- [2] M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer, Cham, 2015.
- [3] P. Dankelmann, W. Goddard, C. Swart, The average eccentricity of a graph and its subgraphs, Util. Math. 65 (2004) 41–51.
- [4] P. Dankelmann, S. Mukwembi, Upper bounds on the average eccentricity, Discrete Appl. Math. 167 (2014) 72–79.
- [5] P. Dankelmann, F. J. Osaye, Average eccentricity,  $k$ -packing and  $k$ -domination in graphs, Discrete Math. 342 (2019) 1261–1274.
- [6] P. Dankelmann, F. J. Osaye, S. Mukwembi, B. Rodrigues, Upper bounds on the average eccentricity of  $K_3$ -free and  $C_4$ -free graphs, Discrete Appl. Math. 270 (2019) 106–114.
- [7] Z. Du, A. Ilić, On AGX conjectures regarding average eccentricity, MATCH Commun. Math. Comput. Chem. 69 (2013) 597–609.
- [8] Z. Du, A. Ilić, A proof of the conjecture regarding the sum of the domination number and average eccentricity, Discrete Appl. Math. 201 (2016) 105–113.

- [9] R. E. Erickson, C. Monma, A. F. Veinott, Send-and-split method for minimum-concave-cost network flows, *Math. Oper. Res.* 12 (1987) 634–664.
- [10] B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, X. Wang, Dynamic programming for minimum Steiner trees, *Theory Comput. Syst.* 41 (2007) 493–500.
- [11] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979.
- [12] T. Gologranc, Steiner convex sets and Cartesian product, *Bull. Malays. Math. Sci. Soc.* 41 (2018) 627–636.
- [13] S. Klavžar, D. Kuziak, I. Peterin, I. G. Yero, A Steiner general position problem in graph theory, *Comput. Appl. Math.* 40 (2021) Paper 223.
- [14] A. Ilić, On the extremal properties of the average eccentricity, *Comput. Math. Appl.* 64 (2012) 2877–2885.
- [15] X. Li, Y. Mao, I. Gutman, The Steiner Wiener index of a graph, *Discuss. Math. Graph Theory* 36 (2016) 455–465.
- [16] X. Li, G. Yu and S. Klavžar, On the average Steiner 3-eccentricity of trees, *Discrete Appl. Math.* 304 (2021) 181–195.
- [17] X. Li, G. Yu, S. Klavžar, J. Hu, B. Li, The Steiner  $k$ -eccentricity on trees, *Theoret. Comput. Sci.* 889 (2021) 182–188.
- [18] Y. Mao, P. Dankelmann, Z. Wang, Steiner diameter, maximum degree and size of a graph, *Discrete Math.* 344 (2021) Paper 112468.
- [19] H. Smith, L. A. Székely, H. Wang, Eccentricity sum in trees, *Discrete Appl. Math.* 207 (2016) 120–131.
- [20] Y. Tang, B. Zhou, On average eccentricity, *MATCH Commun. Math. Comput. Chem.* 67 (2012) 405–423.
- [21] J. Vygen, Faster algorithm for optimum Steiner trees, *Inform. Process. Lett.* 111 (2011) 1075–1079.
- [22] D. Weißbauer, Isometric subgraphs for Steiner distance, *J. Graph Theory* 94 (2020) 597–613.