Shortest Longest-Path Graph Orientations for Trees

Yuichi Asahiro[1(✉)], Jesper Jansson[2], Avraham A. Melkman[3], Eiji Miyano[4], Hirotaka Ono[5], Quan Xue[6], Yoshichika Yano[7], and Shay Zakov[8]

[1] Kyushu Sangyo University, Fukuoka, Japan
asahiro@is.kyusan-u.ac.jp
[2] Kyoto University, Kyoto, Japan
jj@i.kyoto-u.ac.jp
[3] Ben-Gurion University of the Negev, Be'er Sheva, Israel
melkmana@gmail.com
[4] Kyushu Institute of Technology, Iizuka, Japan
miyano@ai.kyutech.ac.jp
[5] Nagoya University, Nagoya, Japan
ono@nagoya-u.jp
[6] The University of Hong Kong, Hong Kong, China
quan.xue@connect.polyu.hk
[7] The University of Tokyo, Tokyo, Japan
yyano-ut@g.ecc.u-tokyo.ac.jp
[8] Ruppin Academic Center, Kfar Monash, Israel
Zakov.Shay@ruppin365.net

Abstract. Graph orientation transforms an undirected graph into a directed graph by assigning a direction to each edge. Among the many different optimization problems related to graph orientations, we focus here on the Shortest Longest-Path Orientation problem (SLPO) which is a generalization of the well-known Minimum Graph Coloring problem. The input to SLPO is an edge-bi-weighted undirected graph in which every edge has two (possibly different and not necessarily positive) lengths associated with its two directions. The goal is to find an orientation of the input graph that minimizes the length of the longest simple directed path. Recently, polynomial-time algorithms for simple graph structures such as paths, cycles, stars, and trees were proposed, and a new polynomial-time inapproximability result was also established. This paper presents (i) an $O(n^2 \log n)$-time algorithm for trees, which is a significant improvement over the previously fastest algorithm whose time complexity was $\Omega(n^{14})$ and (ii) polynomial-time algorithms for trees and spiders that run even faster than (i) as long as every edge weight is an integer and the total weight of the edges is sub-exponential.

Keywords: Graph orientation · Longest path · Tree · Spider

1 Introduction

An *orientation* of an undirected graph is an assignment of a direction to each of its edges. Many interesting optimization problems involving graph orientation

R. Královič and V. Kůrková (Eds.): SOFSEM 2025, LNCS 15538, pp. 51–64, 2025.
https://doi.org/10.1007/978-3-031-82670-2_5

have been extensively researched, e.g., to find an orientation where the total arc-connectivity is maximized [12], an orientation minimizing the maximum outdegree [2,4–6,16], and an orientation maximizing the number of source-target vertex pairs from a given set that become connected by directed source-to-target paths [9,14]. Certain graph orientation problems are equivalent to well-known classic problems. For example, Minimum Vertex Cover (or Maximum Independent Set) is equivalent to orienting an undirected graph such that the number of vertices with outdegree at least one is minimized (or with outdegree zero is maximized) [1]. A natural generalization of Minimum Vertex Cover (or Maximum Independent set) where the outdegree threshold is raised from one (or zero) to any positive integer W was studied in [1].

A graph orientation problem called Unweighted Shortest-Longest-Path Orientation (USLPO) can be viewed as a generalization of Minimum Graph Coloring. The objective of USLPO is to find an orientation of an undirected, unweighted graph that minimizes the length of a longest simple directed path. For any undirected graph G, let $H(G)$ and $\chi(G)$ denote the length of a longest simple directed path in an optimal solution to USLPO for G and the chromatic number of G, respectively. It is known that $H(G) + 1 = \chi(G)$ [10,11,15,17], even when the output orientation must be acyclic [8]. This equality immediately implies the intractability of USLPO: USLPO is NP-hard since Minimum Graph Coloring is NP-hard [13]. Moreover, USLPO cannot be approximated within a ratio of $(3/2 - \varepsilon)$ for any constant $\varepsilon > 0$ in polynomial time unless P=NP even if restricted to 4-regular planar graphs, since it is NP-hard to determine if a 4-regular planar graph G satisfies $\chi(G) \leq 3$ [7].

This paper considers a generalization of USLPO named Shortest Longest-Path Orientation (SLPO) that takes an *edge-bi-weighted* graph as input, in which every edge $\{u, v\}$ has two (potentially different and not necessarily positive) weights $w(u, v)$ and $w(v, u)$ representing the lengths of its two possible directions (u, v) and (v, u). The goal of SLPO is to find an orientation minimizing the length of a longest directed path in the resulting directed graph. If some edge lengths (weights) are negative, then a longest directed path is not necessarily a maximal directed path. Hence, we consider two variants SLPO_m and SLPO_s, in which the longest directed path is taken, respectively, among maximal simple directed paths *only* and among *all* simple directed paths. As stated above, USLPO is NP-hard which immediately implies that SLPO_m and SLPO_s are also NP-hard for general graphs. In fact, SLPO_m and SLPO_s are NP-hard even for subcubic planar graphs [3], where a graph is subcubic if the degree of every vertex is at most three. On the positive side, polynomial-time algorithms for simple graph structures (more precisely, trees, paths, cycles, and stars), are known [3].

This paper further investigates the computational complexity of SLPO_m and SLPO_s. The contributions are as follows.

1. An $O(n \log \Delta)$-time algorithm which determines if there is an orientation with cost at most a fixed value B for SLPO_m on trees, where n is the number of vertices and Δ is the maximum degree of a vertex. This leads to an $O(n^2 \log n)$-time algorithm for SLPO_m on trees, which is a significant improvement over

Table 1. Summary of the results from [3] (top) and this paper (bottom), where n is the number of vertices, Δ is the maximum degree of a vertex, and $Z = \sum_{\{u,v\}} \max\{|w(u,v)|, |w(v,u)|\}$. The time complexities indicated by "*" need the assumption that every edge weight is an integer.

Graph class	SLPO_m	SLPO_s	Reference/Theorems
Path	$O(n \log n)$	$O(n)$	[3]
Cycle	$O(n^2 \log n)$	$O(n)$	[3]
Star	$O(n \log n)$	$O(n \log n)$	[3]
Tree	$\Omega(n^{14})$	$\Omega(n^{14})$	[3]
Subcubic Planar	NP-hard	NP-hard	[3]
Tree	$O(n^2 \log n)$	$O(n^2 \log n)$	Theorems 1 and 2
	$O(n \log \Delta \log Z)^*$	$O(n \log \Delta \log Z)^*$	
Spider	—	$O((n + \Delta \log \Delta) \log Z)^*$	Theorem 3

the previously fastest algorithm that runs in polynomial time but has a time complexity of $\Omega(n^{14})$ [3]; and also an $O(n \log \Delta \log Z)$-time algorithm for SLPO_m on trees under the assumption that all edge weights are integers, where $Z = \sum_{\{u,v\}} \max\{|w(u,v)|, |w(v,u)|\}$. The latter runs faster than the former when Z is sub-exponential. Utilizing these algorithms for SLPO_m on trees, we can solve SLPO_s on trees in the same time complexity.

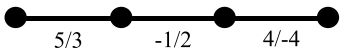
2. Even faster algorithm for SLPO_s on spiders (a subclass of trees), also under the assumption that all edge weights are integers and Z is sub-exponential.

Table 1 summarizes the previously known and the new results in this paper.

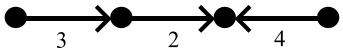
The organization of the paper is as follows. The problems SLPO_s and SLPO_m are defined formally in Sect. 2. Sections 3 and 4 respectively describe our new algorithms for trees and spiders. Concluding remarks are given in Sect. 5. Due to space limitations, many details and all proofs are omitted.

2 Preliminaries

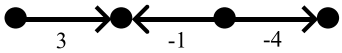
We shall use the following definitions and terminology from Sect. 2 in [3]. Let $G = (V, E)$ be an undirected graph. The vertex set and the edge set of G are denoted by $V(G)$ and $E(G)$, respectively. For a vertex v, its unweighted degree is denoted by $deg(v)$, and Δ denotes the maximum unweighted degree of all vertices. Replacing each undirected edge $\{u, v\} \in E$ by either the directed edge (u, v) (*di-edge* for short) or the di-edge (v, u) gives a directed graph (*di-graph* for short). The resulting di-graph $\widetilde{G}$ is called *an orientation* of G. The vertex set and the di-edge set of $\widetilde{G}$ are respectively denoted by $V(\widetilde{G})$ $(= V(G))$ and $E(\widetilde{G})$. Let $\mathcal{O}(G)$ denote the set of all orientations of G. We sometimes regard an orientation as a set of di-edges: If a di-edge (u, v) is included in $E(\widetilde{G})$, we write $(u, v) \in \widetilde{G}$. The unweighted indegree and the unweighted outdegree of a vertex v in $\widetilde{G}$ are denoted by $deg^-_{\widetilde{G}}(v)$ and $deg^+_{\widetilde{G}}(v)$, respectively.



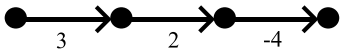
(a) The w_l/w_r below each edge specifies the weights w_l and w_r of that edge's left and right directions.



(b) An orientation $\widetilde{G}$ with $h_s(\widetilde{G}) = 5$ and $h_m(\widetilde{G}) = 5$.



(c) An orientation $\widetilde{G}$ with $h_s(\widetilde{G}) = 3$ and $h_m(\widetilde{G}) = 3$.



(d) An orientation $\widetilde{G}$ with $h_s(\widetilde{G}) = 5$ and $h_m(\widetilde{G}) = 1$.

Fig. 1. (a) An edge-bi-weighted graph G, (b) an example orientation of G, (c) an optimal orientation of G under cost function H_s (here, $H_s(G) = 3$), and (d) an optimal orientation of G under cost function H_m (here, $H_m(G) = 1$).

An *edge-bi-weighted graph* $G = (V, E, w)$ is an undirected graph $G = (V, E)$ in which every edge $\{u, v\} \in E$ has a pair of weights $w(u, v)$ and $w(v, u)$ associated with the two directions (u, v) and (v, u), respectively. The weights $w(u, v)$ and $w(v, u)$ are possibly nonpositive. We define $Z = \sum_{\{u,v\}\in E} \max\{|w(u, v)|, |w(v, u)|\}$. A *directed path* (*di-path* for short) in a di-graph $\widetilde{G}$ is a sequence $\langle v_1, v_2, \ldots, v_q\rangle$ of vertices such that for $k \in \{1, 2, \ldots, q-1\}$, the graph $\widetilde{G}$ contains the di-edge (v_k, v_{k+1}). We say that this di-path *starts from* v_1, *ends at* v_q, and *passes* v_i for $2 \leq i \leq q - 1$. The *length* of a di-path $\overrightarrow{P} = \langle v_1, v_2, \ldots, v_q\rangle$ in an orientation of an edge-bi-weighted graph is: $W(\overrightarrow{P}) = \sum_{k=1}^{q-1} w(v_k, v_{k+1})$. The di-path $\overrightarrow{P}$ is *simple* if all vertices in $\overrightarrow{P}$ are distinct. Also, $\overrightarrow{P}$ is *maximal* if it is not contained in any di-path with more vertices. If some edge weights are negative, then a longest di-path is not necessarily maximal. Hence, two alternative cost functions for an orientation are defined; see Fig. 1 for an example that shows the difference between them.

Definition 1 [3]**.** *Define the following two cost measures for an orientation* $\widetilde{G}$ *of an edge-bi-weighted graph* G*:*

$$h_s(\widetilde{G}) = \max\{W(\overrightarrow{P}) \mid \overrightarrow{P} \text{ is a simple di-path in } \widetilde{G}\} \text{ and}$$
$$h_m(\widetilde{G}) = \max\{W(\overrightarrow{P}) \mid \overrightarrow{P} \text{ is a maximal simple di-path in } \widetilde{G}\}.$$

The corresponding two cost functions for orienting G *are:*

$$H_s(G) = \min\{h_s(\widetilde{G}) \mid \widetilde{G} \in \mathcal{O}(G)\} \text{ and } H_m(G) = \min\{h_m(\widetilde{G}) \mid \widetilde{G} \in \mathcal{O}(G)\}.$$

Note that a di-path including only one vertex is a simple di-path with zero length. Hence $h_s(\widetilde{G}) \geq 0$ holds for any $\widetilde{G}$, and thus $H_s(G) \geq 0$ holds for any G.

The two problem variants that we consider in this article are the following:

The Shortest Longest-Path Orientation Problem, variants SLPO$_s$ & SLPO$_m$:
Input: An undirected, edge-bi-weighted graph G.
Output: An orientation $\widetilde{G}$ of G such that $h_s(\widetilde{G}) = H_s(G)$ (for SLPO$_s$) or $h_m(\widetilde{G}) = H_m(G)$ (for SLPO$_m$).

For an edge-bi-weighted graph G and a fixed value B, an orientation $\widetilde{G}$ of G is *B-feasible* if $h_x(\widetilde{G}) \leq B$ for SLPO$_x$ ($x \in \{s, m\}$). Our basic strategy for solving SLPO$_s$/SLPO$_m$ is to design an algorithm that answers the following question:

Question 1. Does G have a B-feasible orientation?

3 Trees

In this section, we first design a dynamic programming algorithm to answer Question 1 for cost function H_m, and then utilize it to solve both of SLPO$_m$ and SLPO$_s$. First, we introduce several definitions only used for trees in Sect. 3.1 Then, Sects. 3.2 and 3.3 describe three procedures that are to be applied locally to subtrees of the input tree in order to transform it into a star. Section 3.4 presents an algorithm that uses these procedures in order to answer Question 1 for H_m. Finally, Sect. 3.5 describes the main algorithm for SLPO$_m$ and how to apply it to also solve SLPO$_s$ restricted to trees.

3.1 Preliminaries for Trees

Let $T = (V, E)$ be an undirected edge-bi-weighted tree with root r. Assume that $V = \{v_1, v_2, \ldots, v_{n-1}, v_n\}$ and $v_n = r$ without loss of generality. The parent of a vertex v_i in T is denoted by $p(v_i)$ if it exists. For a vertex v_i, I_i denotes the set of indices of children of v_i in T. The subtree rooted at v_i in T is denoted by T_i.

Consider an orientation $\widetilde{T}$ of T. Let $\widetilde{T}_i$ be the directed subtree rooted at v_i in $\widetilde{T}$. For a di-path $\overrightarrow{P}$ from v_i to v_j (note that it is unique) in a directed tree $\widetilde{T}$ of T, the length of $\overrightarrow{P}$ is defined as the total weight of di-edges in $\overrightarrow{P}$, and is denoted by $W(i, j)$. On the other hand, if there is no path from v_i to v_j, $W(i, j) = \infty$. Let $V^i_{\uparrow}$ be the set $\{v_j \mid \widetilde{T}_i$ contains an upward di-path from v_j to $v_i\}$, i.e., the set of all vertices that can reach v_i by following an upward path in $\widetilde{T}_i$. Similarly, $V^i_{\downarrow}$ is defined as the set $\{v_j \mid \widetilde{T}_i$ contains an downward di-path from v_i to $v_j\}$. We use the convention that a path may have zero edges, so $v_i \in V^i_{\uparrow}$ and $v_i \in V^i_{\downarrow}$, by which $V^i_{\uparrow} \neq \emptyset$, $V^i_{\downarrow} \neq \emptyset$, and $W(i, i) = 0$. Note that $V^i_{\uparrow} \cup V^i_{\downarrow} = V(T_i)$ does not necessarily hold. Next, let $W^i_{\uparrow}$ for SLPO$_m$ be the length of a longest di-path from vertices in $V^i_{\uparrow}$ to v_i, and analogously for $W^i_{\downarrow}$. Formally,

$$W^i_{\uparrow} = \max\left\{W(j, i) \mid v_j \in V^i_{\uparrow},\ deg^-_{\widetilde{T}_i}(v_j) = 0\right\} \text{ and}$$
$$W^i_{\downarrow} = \max\left\{W(i, j) \mid v_j \in V^i_{\downarrow},\ deg^+_{\widetilde{T}_i}(v_j) = 0\right\}.$$

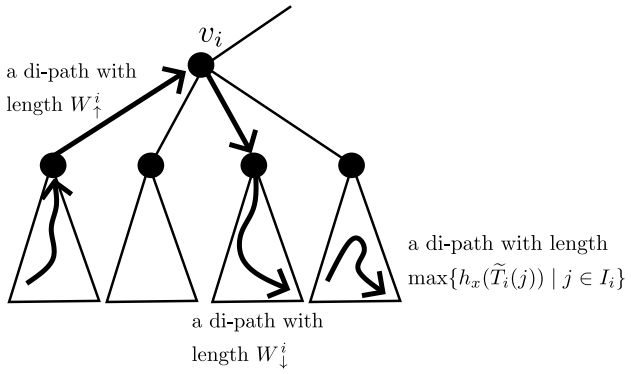


Fig. 2. Illustration of Observation 1. We consider three di-paths which may be (a part of) a (maximal) longest di-path.

For SLPO$_s$, $W^i_\uparrow$ and $W^i_\downarrow$ are defined to take all simple di-paths into account:

$$W^i_\uparrow = \max\left\{W(j,i) \mid v_j \in V^i_\uparrow\right\} \text{ and } W^i_\downarrow = \max\left\{W(i,j) \mid v_j \in V^i_\downarrow\right\}.$$

For $\widetilde{T}_i$ and a child v_j of v_i, i.e., $j \in I_i$, $\widetilde{T}_i(j)$ represents the directed subtree rooted at v_j in $\widetilde{T}_i$. If we are given an orientation $\widetilde{T}$ of T, then $\widetilde{T}_j$ equals $\widetilde{T}_i(j)$. However, we sometimes need to define or construct a part of an orientation of the whole graph before the other parts. For this purpose, we define $\widetilde{T}_i(j)$. The next observation gives the maximum length of a (maximal) simple di-path in $\widetilde{T}_i$, where $\max\{h_x(\widetilde{T}_i(j)) \mid j \in I_i\}$ for $x \in \{m,s\}$ represents the maximum length of a (maximal) simple di-path in $\widetilde{T}_i$, which does not pass v_i. Figure 2 illustrates this observation.

Observation 1. *For a directed subtree $\widetilde{T}_i$, it holds that*

$$h_m(\widetilde{T}_i) = \max\left\{W^i_\uparrow + W^i_\downarrow,\ \max\left\{h_m(\widetilde{T}_i(j)) \mid j \in I_i\right\}\right\} \text{ and}$$
$$h_s(\widetilde{T}_i) = \max\left\{W^i_\uparrow,\ W^i_\downarrow,\ W^i_\uparrow + W^i_\downarrow,\ \max\left\{h_s(\widetilde{T}_i(j)) \mid j \in I_i\right\}\right\}.$$

In the above, we assumed that the root vertex $r = v_n$. However the choice of the root vertex r is not important for the following reason. Based on Observation 1, for a directed tree $\widetilde{T}$ with root $r = v_n$, it holds that $h_m(\widetilde{T}) = h_m(\widetilde{T_n})$ and $h_s(\widetilde{T}) = h_s(\widetilde{T_n})$. Pick a child v_i of $r(= v_n)$, where $i \neq n$. When computing $h_m(\widetilde{T_n})$ or $h_s(\widetilde{T_n})$, a di-edge between r and v_i are in consideration as a part of computing $W^n_\uparrow$ and $W^n_\downarrow$. Before this, di-edges between v_i and children of v_i were taken into account during computation of $W^i_\uparrow$ and $W^i_\downarrow$. Thus, letting v_i as a root of $\widetilde{T}$ instead of r, just changes the order of the computation of $W_\uparrow$'s and $W_\downarrow$'s. Thus, we can choose an arbitrary vertex as a root of a tree.

The $\Omega(n^{14})$-time algorithm in [3] was a dynamic programming algorithm that implemented Observation 1 directly. To obtain a fast algorithm, we need a more

Algorithm 1: OrientStarDown$_B(T_i)$

Input: an edge-bi-weighted star (subgraph) T_i of T, where T_i is rooted at v_i, the vertex set of T_i is $\{v_i, u_1, u_2, \ldots, u_d\}$, edge set or T_i is $\{\{v_i, u_1\}, \{v_i, u_2\}, \ldots, \{v_i, u_d\}\}$, and $u_1, u_2, \ldots, u_d$ are all leaves of T

Output: an orientation of T_i and its cost

1 Sort $w(v_i, u_j)$'s in the non-decreasing order. Without loss of generality, we assume $w(v_i, u_1) \leq w(v_i, u_2) \leq \cdots \leq w(v_i, u_d)$;

2 For each $j \in \{1, 2, \ldots, d-1\}$, $c_j \leftarrow \max\{w(u_k, v_i) \mid j+1 \leq k \leq d\}$ and let $\overline{j}$ be an index such that $w(u_{\overline{j}}, v) = c_j$;

3 Find the smallest $j \in \{1, 2, \ldots, d-1\}$ such that $c_j + w(v_i, u_j) \leq B$ holds. If there exists such j, then let $\widetilde{T}_i$ be an orientation of T_i in which each edge $\{v_i, u_k\}$ is directed toward u_k for $1 \leq k \leq j$, and toward v_i for $j+1 \leq k \leq d$, and $C \leftarrow c_j + w(v_i, u_j)$. Otherwise, $C \leftarrow \infty$.

4 Let $\widetilde{T}_i^{up}$ (or $\widetilde{T}_i^{down}$) be an orientation of T_i in which every edge $\{v_i, u_k\}$ is directed toward v_i (or toward u_k). If $\max\{w(u_k, v_i) \mid 1 \leq k \leq d\} \leq B$, then $C^{up} \leftarrow 0$, otherwise $C^{up} \leftarrow \infty$. Also, $C^{down} \leftarrow w(v_i, u_d)$.

5 $C_{\min} \leftarrow \min\{C, C^{up}, C^{down}\}$. If $C_{\min} = C$, C^{up}, or C^{down}, then output $(\widetilde{T}_i, C)$, $(\widetilde{T}_i^{up}, C^{up})$, or $(\widetilde{T}_i^{down}, C^{down})$, respectively;

refined approach. To this end, we propose a new technique for answering Question 1 efficiently by using graph transformations and another type of dynamic programming. Our strategy is to minimize $W_\uparrow^i$ and $W_\downarrow^i$ for every vertex v_i under the condition that $\max\{h_x(\widetilde{T}_i(j)) \mid j \in I_i)\} \leq B$ for $x \in \{m, s\}$ and a fixed value B in a bottom-up manner by maintaining two values $L_\uparrow^i$ and $L_\downarrow^i$ for each vertex v_i, defined as follows:

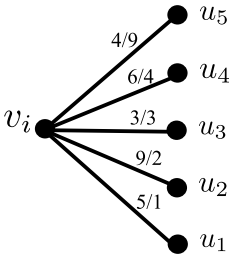
$$L_\uparrow^i = \min\{W_\uparrow^i \mid \widetilde{T}_i \text{ is an orientation of } T_i,\ \max\{h_x(\widetilde{T}_i(j)) \mid j \in I_i\} \leq B\} \text{ and}$$
$$L_\downarrow^i = \min\{W_\downarrow^i \mid \widetilde{T}_i \text{ is an orientation of } T_i,\ \max\{h_x(\widetilde{T}_i(j)) \mid j \in I_i\} \leq B\},$$

where $x = m$ for SLPO$_m$ and $x = s$ for SLPO$_s$. For a leaf v_i, $L_\uparrow^i = L_\downarrow^i = 0$.

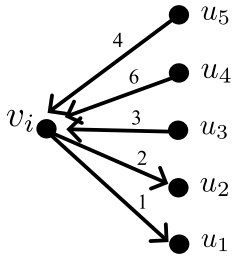
Algorithms to compute $L_\uparrow^i$ and $L_\downarrow^i$ of a subtree in the input tree for SLPO$_m$ are given in Sect. 3.2. In Sect. 3.3, we transform a tree into another one based on these algorithms. Based on Observation 1, an algorithm to answer Question 1 for SLPO$_m$ is given in Sect. 3.4. Finally, Sect. 3.5 describes the whole algorithm for SLPO$_m$ and how to apply it to SLPO$_s$.

3.2 Procedures for Stars

Let T be a tree with root r. Suppose that children of a vertex v_i are all leaves (note that there must exist such a vertex). Let $u_1, \ldots, u_d$ for $d \geq 1$ be the children of v_i. The subtree T_i rooted at v_i is a star, whose vertex set is $\{v_i, u_1, u_2, \ldots, u_d\}$. In this subsection, we propose an algorithm for a star, which will be used as a subroutine. For a fixed B, an algorithm OrientStarDown$_B$ (listed in Algorithm 1)



(a) A star T_i.



(b) Computing $c_2 = w(u_4, v_i) = 6$ and hence $\overline{2} = 4$ for T_i.

Fig. 3. An example of OrientStarDown$_B$ for a star T_i in (a) and $B = 8$. (b) is the obtained orientation $\widetilde{T}_i$ of T_i.

outputs an orientation of T_i such that the length of the maximal di-path (di-edge) starting from v is minimum and every other maximal di-path ending at v or passing v has length at most B, if such an orientation exists.[1]

Figure 3(b) illustrates an example of the output of OrientStarDown$_B$ for the star T_i in Fig. 3(a) and $B = 8$. In Step 2, we compute c_2 by directing the edges $\{v_i, u_1\}$ and $\{v_i, u_2\}$ as (v_i, u_1) and (v_i, u_2), respectively, and directing other edges toward v_i. Then, we obtain $c_2 = \max\{w(u_k, v_i) \mid 3 \leq k \leq 5\} = w(u_4, v_i)$ and hence we set $\overline{2} = 4$, where $\overline{2}$ indicates a di-edge having maximum weight among di-edges (u_k, v_i)'s for $2 < k$. Then, we see that the longest maximal di-path in this orientation is $\langle u_{\overline{2}}, v_i, u_2\rangle = \langle u_4, v_i, u_2\rangle$ with length 8 and thus $C = 8$ in Fig. 3(b). As for Step 4, $C^{up} = \infty$ since $w(u_2, v_i) = 9 > B(= 8)$ and also $C^{down} = w(v_i, u_5) = 9$. Step 5 may output $\widetilde{T}_i^{down}$ (or $\widetilde{T}_i^{up}$) even if $C^{down} > B$ (or $C^{up} > B$), when $C > C^{down}$ (or $C > C^{up}$). The reason is that $\widetilde{T}_i^{down}$ (or $\widetilde{T}_i^{up}$) may minimize the length of a longest maximal di-path passing v_i in the whole orientation of T due to the possible existence of an edge with negative weight. (This is not the case in the example in Fig. 3.) Finally, the orientation in Fig. 3(b) is the output of OrientStarDown$_B(T_i)$ for $B = 8$, since $C < C^{down}$.

Let $\widetilde{T}_i^D$ be the orientation obtained by OrientStarDown$_B$ for T_i. Define an index i_D as follows: If $C_{\min} = C^{up}$, then let $i_D = 0$ and let $w(v, u_{i_D}) = w(v, u_0) = 0$ (though there is no vertex u_0). If $C_{\min} = C^{down}$, then let $i_D = d$ (and hence $w(v, u_{i_D}) = w(v, u_d)$). Otherwise if $C_{\min} = C$, then let i_D be the index j found in Step 3 of OrientStarDown$_B$. The following lemma guarantees that OrientStarDown$_B$ outputs an intended orientation which gives $L^i_{\downarrow}(= h_m(\widetilde{T}_i^D))$.

Lemma 1. *In $\widetilde{T}_i^D$, the length of the longest di-edge starting from v_i is the minimum among all orientations of $\widetilde{T}_i$ such that every maximal di-path ending at v_i or passing v_i has length at most B. Moreover, the orientation $\widetilde{T}_i^D$ of T_i can be obtained in $O(d \log d)$ time.*

[1] This algorithm differs from the known ones for stars in [3].

Algorithm 2: DeleteLeaf(T, v_i, $L^i_\downarrow$, $L^i_\uparrow$)

Input: an edge-bi-weighted rooted tree T, a vertex v_i whose children are all leaves in T, and two values $L^i_\downarrow$ and $L^i_\uparrow$

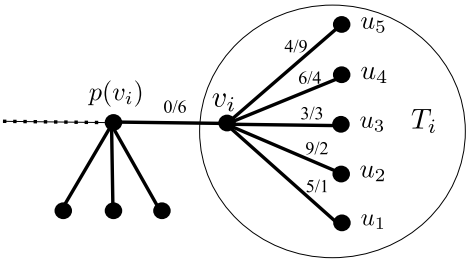
Output: an edge-bi-weighted tree T'

1 Let the children of v_i in T be $u_1, u_2, \ldots, u_d$ ($d \geq 1$);

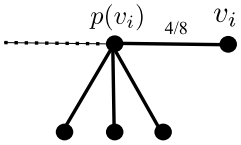
2 Remove the vertices $u_1, u_2, \ldots, u_d$ and the edges $\{v_i, u_1\}, \{v_i, u_2\}, \ldots, \{v_i, u_d\}$ from T;

3 Update the weights of the edge $\{p(v_i), v_i\}$ as $w(p(v_i), v_i) \leftarrow w(p(v_i), v_i) + L^i_\downarrow$ and $w(v_i, p(v_i)) \leftarrow w(v_i, p(v_i)) + L^i_\uparrow$;

4 Output the resulting T (as T');



(a) A tree T and T_i (in Fig. 3), where $i_D = 2$ and $i_U = 5$.



(b) Another tree T' constructed from T by DeleteLeaf.

Fig. 4. An example of DeleteLeaf for $B = 8$.

We define an algorithm OrientStarUp$_B$ in the same way as OrientStarDown$_B$, but where the order of v_i and u_k is swapped when weights of edges are handled and where the directions of edges determined in Steps 3 and 4 are the opposite. (We omit the description of OrientStarUp$_B$, since it is very similar to OrientStarDown$_B$.) Then, Let $\widetilde{T}^U_i$ be the orientation with cost $L^i_\uparrow (= h_m(\widetilde{T}^U_i))$, which is obtained by applying OrientStarUp$_B$ to T_i. The index i_U is defined similarly to the index i_D.

Lemma 2. *In $\widetilde{T}^U_i$, the length of the longest di-edge ending at v_i is the minimum among all orientations of $\widetilde{T}_i$ such that every maximal di-path starting from v_i or passing v_i has length at most B. Moreover, the orientation $\widetilde{T}^U_i$ of T_i can be obtained in $O(d \log d)$ time.*

3.3 Transformation

Let T be an edge-bi-weighted tree, which is not a star. Taking $L^i_\downarrow$ and $L^i_\uparrow$ computed by the procedures in the previous subsection as input, the algorithm DeleteLeaf (listed in Algorithm 2) constructs another tree T' that has fewer vertices than T. The resulting T' may be a star.

Figure 4 illustrates an example of DeleteLeaf for $B = 8$. In Fig. 4(a), the part T_i is the same as in Fig. 3, where $i_D = 2$ with $L^2_\downarrow = 2$ and $i_U = 5$ with $L^2_\uparrow =$

4. Thus, DeleteLeaf constructs another tree in Fig. 4(b), in which $w(p(v_i), v_i))$ is updated to $w(p(v_i), v_i) + (v_i, u_2) = 8$ and also $w(v_i, p(v_i))$ is updated to $w(v_i, p(v_i)) + w(u_5, v_i) = 4$.

Procedure DeleteLeaf only modifies T around v_i and its neighboring vertices, so its running time can be bounded as follows.

Lemma 3. *Given an edge-bi-weighted rooted tree T and a vertex v_i (with $L^i_\downarrow$ and $L^i_\uparrow$) as input, DeleteLeaf runs in $O(d)$ time, where $d = deg(v_i)$.*

The following lemma relates the optimal cost of T' to that of T. This leads to an algorithm OrientTree$_B$ in the next subsection.

Lemma 4. *$H_m(T) \leq B$ if and only if $H_m(T') \leq B$.*

3.4 An Algorithm to Answer Question 1

In this subsection, we describe the algorithm OrientTree$_B$ which answers Question 1 based on the results in the previous subsections. The basic strategy of OrientTree$_B$ is as follows. First we construct a sorted list S of vertices in the input tree T. This list indicates the order of vertices to apply OrientStarDown$_B$, OrientStarUp$_B$, and DeleteLeaf. By DeleteLeaf, T is transformed to another tree T'. Then we remove the first element of S for T', where the first element of the resulting new list indicates the next target to apply OrientStarDown$_B$, OrientStarUp$_B$, and DeleteLeaf. In this way, we iteratively apply these three procedures and then obtain a star. Finally, we apply BestOrientStar$_m$ in [3] to compute an optimal orientation of the obtained star.

We start with the construction of the list S of vertices in T. First choose a vertex arbitrarily as the root r. By using the breadth first search starting from r, we construct a sorted list S which contains all non-leaf vertices in T, and if $i \leq j$, the distance between $S[i]$ and r is not smaller than that between $S[j]$ and r, where a tie is broken arbitrarily. Suppose that we apply DeleteLeaf to the first vertex $S[1]$ of S and its children in T, and then obtain a new tree T', One can see that children of $S[2]$ must be leaves in T', even if $S[1]$ is a child of $S[2]$ in T, since DeleteLeaf transforms a star formed by $S[1]$ and its children in T to a vertex. Hence we can apply DeleteLeaf to $S[2]$ for T' as the next step. Since the construction of S is done mainly by the breadth first search, we have:

Lemma 5. *The construction of the list S is done in $O(n)$ time.*

The algorithm OrientTree$_B$ is listed in Algorithm 3. It uses an algorithm named BestOrientStar$_m$ from [3], which solves SLPO$_m$ on stars exactly. The following lemma guarantees the correctness of OrientTree$_B$.

Lemma 6. *OrientTree$_B$ answers Question 1 in $O(n \log \Delta)$ time.*

3.5 Time Complexity and Cost Function H_s

The whole algorithm for cost function H_m is listed in Algorithm 4. Here it outputs the value $H_m(T)$ only. However, it is easy to output an orientation

with cost $H_m(T)$ by modifying OrientTree$_B$ so that it also outputs an obtained orientation when answering "Yes".

Algorithm 3: OrientTree$_B(T, S)$

Input: an edge-bi-weighted tree T which is not a star, and a list S of vertices
Output: "Yes" or "No"

while T *is not a star* **do**
  Suppose the first element $S[1]$ of S is v_i;
  $(\widetilde{T}_i^D, L_\downarrow^i) \leftarrow$ OrientStarDown$_B(T_i)$ and $(\widetilde{T}_i^U, L_\uparrow^i) \leftarrow$ OrientStarUp$_B(T_i)$;
  $T \leftarrow$ DeleteLeaf$(T, v_i, L_\downarrow^i, L_\uparrow^i)$, and delete $S[1]$ from S;
Apply BestOrientStar$_m$ in [3] to T. If the obtained orientation has cost at most B, output "Yes", otherwise, output "No";

Algorithm 4: OrientTree$_m(T)$

Input: an edge-bi-weighted tree T which is not a star
Output: $H_m(T)$

Construct the list S of T;
Find the minimum B such that OrientTree$_B(T, S)$ returns "Yes";
Output B;

Algorithm 5: AddLeaf(G)

Input: an edge-bi-weighted graph G
Output: an edge-bi-weighted graph G'

For each vertex v_i of degree at least two, add two vertices x_i and y_i to G;
To G, add an edge $\{v_i, x_i\}$ with weights $w(v_i, x_i) = \infty$ and $w(x_i, v_i) = 0$, and then add an edge $\{v_i, y_i\}$ with weights $w(v_i, y_i) = 0$ and $w(y_i, v_i) = \infty$;
return G (as G');

Theorem 1. *If the input is a tree, then SLPO$_m$ can be solved in $O(n^2 \log n)$ time. Moreover, if every edge weight of the input tree is an integer, SLPO$_m$ can be solved in $O(n \log \Delta \log Z)$ time.*

To solve SLPO$_s$, we reduce SLPO$_s$ to SLPO$_m$. To this end, we introduce an algorithm AddLeaf (listed in Algorithm 5). This algorithm transforms an edge-bi-weighted graph G to another edge-bi-weighted graph G' in linear time by adding leaves to G. The resulting graph G' does not always maintain the structural properties of G. For example, if G is a tree then so is G', but if G is a cycle then G' is not. However, the next lemma shows that the optimal cost of G for cost function H_s transfers to G'.

Lemma 7. $H_s(G) = H_m(G')$

The important point here is that AddLeaf transforms a tree to another tree. Thus, Theorem 1 and Lemma 7 immediately yield the following theorem.

Theorem 2. *If the input is a tree, then SLPO$_s$ can be solved in $O(n^2 \log n)$ time. Moreover, if every edge weight of the input tree is an integer, SLPO$_s$ can be solved in $O(n \log \Delta \log Z)$ time.*

4 Spiders with Cost Function H_s

This section presents an algorithm that solves SLPO$_s$ on spiders under the assumption that all edge weights are integers. It runs faster than the algorithm for trees when Z is sub-exponential. A *spider* is a tree with exactly one vertex r of degree greater than 2, referred to as the *root vertex*. Let G be a spider which has root vertex r with degree $\Delta \geq 3$ and undirected paths $P_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}, r)$ of length n_i for $1 \leq i \leq \Delta$. We call each P_i a *leg*. For simplicity, we let $v_{i,n_i+1} = r$.

Here we briefly sketch the main idea of the algorithm. For a leg P_i with n_i+1 vertices, the optimal cost $H_s(P_i)$ under H_s can be obtained in $O(n_i)$ time by the algorithm BestOrientPath$_s$ in [3]. However, just combining optimal orientations of $P_1, \ldots, P_\Delta$ may not give an optimal orientation for G. Instead, we proceed as follows.

1. For each leg P_i, we obtain an orientation such that $\{v_{i,n_i}, r\}$ is directed toward r (or toward v_{i,n_i}) and any maximal di-path not including (v_{i,n_i}, r) (or (r, v_{i,n_i})) inside P_i has length at most B, utilizing BestOrientPath$_s$.
2. Based on these two orientations for each leg, we construct a star G' from an input spider G, which preserves the lengths of longest di-paths, ending at, starting from, and passing r in G.
3. From an orientation of G' obtained by the algorithm BestOrientStar$_s$ in [3], we construct an orientation of G.

The above procedure answers Question 1 for a fixed B in $O(n + \Delta \log \Delta)$ time. Then, by utilizing the above procedure in a binary search manner on B having $O(Z)$ candidates, we obtain the main result of this section.

Theorem 3. *Suppose that the input is a spider in which every edge weight is an integer. Then, SLPO$_s$ can be solved in $O((n + \Delta \log \Delta) \log Z)$ time.*

5 Concluding Remarks

In this paper, we presented efficient algorithms for SLPO$_m$ and SLPO$_s$ on trees and spiders. Some open questions are:

– Are faster algorithms possible? For example, linear-time algorithms to solve SLPO$_m$ for paths, cycles, and stars are plausible targets.

- Can we remove the assumption that all edge weights are integers and Z is sub-exponential, imposed to obtain faster algorithms for trees and spiders?
- Is it possible to design polynomial-time algorithms for other graph classes, such as unicyclic graphs, cactus graphs, and graphs with bounded treewidth?
- Is there a polynomial-time approximation algorithm for subcubic planar graphs?
- Is there any graph class, for which the (in)tractability of SLPO_s differs from SLPO_m, e.g., SLPO_m is polynomial-time solvable while SLPO_s is NP-hard?

Acknowledgments. This work was supported by JSPS KAKENHI Grant Numbers JP22K11915 and JP24K02902.

References

1. Asahiro, Y., Jansson, J., Miyano, E., Ono, H.: Graph orientations optimizing the number of light or heavy vertices. J. Graph Algor. Appl. **19**(1), 441–465 (2015)
2. Asahiro, Y., Jansson, J., Miyano, E., Ono, H., Zenmyo, K.: Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. J. Comb. Optim. **22**(1), 78–96 (2011)
3. Asahiro, Y., et al.: Shortest longest-path graph orientations. In: Proceedings of the 29th International Computing and Combinatorics Conference (COCOON 2023). LNCS, vol. 14422 , pp. 141–154. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-49190-0_10
4. Asahiro, Y., Miyano, E., Ono, H., Zenmyo, K.: Graph orientation algorithms to minimize the maximum outdegree. Int. J. Found. Comput. Sci. **18**(2), 197–215 (2007)
5. Borradaile, G., Iglesias, J., Migler, T., Ochoa, A., Wilfong, G., Zhang, L.: Egalitarian graph orientations. J. Graph Algor. Appl. **21**(4), 687–708 (2017)
6. Chrobak, M., Eppstein, D.: Planar orientations with low out-degree and compaction of adjacency matrices. Theor. Comput. Sci. **86**(2), 243–266 (1991)
7. Dailey, D.P.: Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. Disc. Math. **30**(3), 289–293 (1980)
8. Deming, R.W.: Acyclic orientations of a graph and chromatic and independence numbers. J. Comb. Theory Ser. B **26**(1), 101–110 (1979)
9. Elberfeld, M., et al.: On the approximability of reachability-preserving network orientations. Internet Math. **7**(4), 209–232 (2011)
10. Gallai, T.: On directed graphs and circuits. In: Theory of Graphs (Proceedings of the Colloquium held at Tihany 1966), pp. 115–118. Akadémiai Kiadó (1968)
11. Hasse, M.: Zur algebraischen Begründung der Graphentheorie. I. Mathematische Nachrichten **28**(5–6), 275–290 (1965)
12. Hörsch, F.: On orientations maximizing total arc-connectivity. Theor. Comput. Sci. **978**, 114176 (2023)
13. Karp, R.M.: Reducibility among combinatorial problems. In: Proceedings of Complexity of Computer Computations. The IBM Research Symposia Series, pp. 85–103. Plenum Press (1972)

14. Medvedovsky, A., Bafna, V., Zwick, U., Sharan, R.: An algorithm for orienting graphs based on cause-effect pairs and its applications to orienting protein networks. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS, vol. 5251, pp. 222–232. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87361-7_19
15. Roy, B.: Nombre chromatique et plus longs chemins d'un graphe. Revue française d'informatique et de recherche opérationnelle **1**(5), 129–132 (1967)
16. Venkateswaran, V.: Minimizing maximum indegree. Disc. Appl. Math. **143**(1–3), 374–378 (2004)
17. Vitaver, L.M.: Determination of minimal coloring of vertices of a graph by means of Boolean powers of the incidence matrix. In: Proceedings of the USSR Academy of Sciences, vol. 147, pp. 758–759. Nauka (1967). (in Russian)