Shortest Longest-Path Graph Orientations

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

[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] Ruppin Academic Center, Kfar Monash, Israel
Zakov.Shay@ruppin365.net

Abstract. We consider a graph orientation problem that can be viewed as a generalization of Minimum Graph Coloring. Our problem takes as input an undirected graph $G = (V, E)$ in which every edge $\{u, v\} \in E$ has two (potentially different and not necessarily positive) weights representing the lengths of its two possible directions (u, v) and (v, u), and asks for an orientation, i.e., an assignment of a direction to each edge of G, such that the length of a longest simple directed path in the resulting directed graph is minimized. A longest path in a graph is not always a maximal path when some edges have negative lengths, so the problem has two variants depending on whether *all* simple directed paths or maximal simple directed paths *only* are taken into account in the definition. We prove that the problems are NP-hard to approximate even if restricted to subcubic planar graphs, and develop fast polynomial-time algorithms for both problem variants for three classes of graphs: path graphs, cycle graphs, and star graphs.

Keywords: Algorithm · Computational complexity · Graph orientation · Graph coloring · Path graph · Cycle graph · Star graph

Y. Asahiro—Funded by KAKENHI grant number JP22K11915.
E. Miyano—Funded by KAKENHI grant number JP21K11755.

W. Wu and G. Tong (Eds.): COCOON 2023, LNCS 14422, pp. 141–154, 2024.
https://doi.org/10.1007/978-3-031-49190-0_10

1 Introduction

1.1 Background

An *orientation* of an undirected graph is an assignment of a direction to each of its edges. Based on this natural concept, many kinds of algorithmic problems with applications to telecommunications, scheduling, data structures for supporting fast adjacency queries in sparse graphs, bioinformatics, etc. can be defined (see, e.g., [2,3,5,7,8,18,20]). Certain graph orientation problems have turned out to be equivalent to well-known classic graph algorithmic problems, and modifying the definitions of these graph orientation problems may then yield new generalizations of their classic counterparts. To illustrate, recall the Minimum Vertex Cover problem and the Maximum Independent Set problem, which take as input an undirected graph $G = (V, E)$ and ask for a smallest possible subset $V' \subseteq V$ such that every edge in E is incident to at least one vertex in V', and a largest possible subset $V' \subseteq V$ such that no two vertices in V' are adjacent in G, respectively. As shown in [1], to orient an undirected graph while minimizing the number of vertices with outdegree at least 1 is in fact the Minimum Vertex Cover problem; by replacing the number "1" by a parameter W, one obtains a relaxed variant of Minimum Vertex Cover in which every vertex of the input graph is allowed to cover up to $W-1$ of its incident edges without having to be placed in the output vertex cover. Similarly, maximizing the number of vertices that get outdegree at most W is the Maximum Independent Set problem when $W = 0$ (see [1]).

In this paper, we study a graph orientation problem that can be viewed as a generalization of another classic problem, namely Minimum Graph Coloring. Our starting point is the following problem, which we call Unweighted Shortest Longest-Path Orientation (USLPO): *Given an undirected, unweighted graph G, find an orientation of G 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. In the 1960s and 1970s, several researchers independently proved that $H(G)+1 = \chi(G)$ [10,13,19,21] and that this equality still holds when only acyclic orientations are allowed [7]. Note that since Minimum Graph Coloring is NP-hard [15], this immediately implies that USLPO is NP-hard. Moreover, known inapproximability results apply directly as well; e.g., Theorem 1.2 of [22] shows that Minimum Graph Coloring, and hence USLPO, cannot be approximated within a ratio of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ in polynomial time, where n is the number of vertices in the input graph, unless P = NP. Also, $H(G)+1 = \chi(G)$ implies that even if USLPO is restricted to 4-regular planar graphs, it cannot be approximated within a ratio of $(3/2-\varepsilon)$ for any constant $\varepsilon > 0$ in polynomial time, unless P = NP, because it is an NP-complete problem to determine if a 4-regular planar graph G satisfies $\chi(G) \leq 3$ or $\chi(G) = 4$ [6].

1.2 New Results

We generalize USLPO (thus making the problem even harder) to *bi-weighted* edges, which means that every edge $\{u, v\}$ in G has two (potentially different

and not necessarily positive) weights representing the lengths of its two possible directions (u, v) and (v, u). Our goal is then to determine whether the generalized problem, from here on simply referred to as Shortest Longest-Path Orientation (SLPO), becomes efficiently solvable in some special cases. Observe that in a directed graph, if some edge lengths are negative then a longest path is not necessarily a maximal path. For this reason, we consider two variants of the problem called SLPO_s and SLPO_m, in which the longest path is taken, respectively, among *all* simple directed paths and among maximal simple directed paths *only*.

An undirected graph G is *subcubic* if every vertex in G has degree at most three. We first prove that SLPO_s and SLPO_m are NP-hard to approximate even if restricted to subcubic planar graphs. This result is important in view of the close connection between USLPO and Minimum Graph Coloring described above and the fact that the latter is polynomial-time solvable for subcubic graphs [11].

Motivated by the hardness of the general case, we then focus on special cases. Throughout the paper, n denotes the number of vertices in the input graph. As a first step, note that if G is a tree then one can root G in an arbitrarily selected vertex and apply dynamic programming over rooted subtrees: For every node v of G (in bottom-up order) and every possible triple (W_u, W_d, W_ℓ) of weights, check if there exists an orientation of the subtree rooted at v whose longest paths to, from, and not passing through v have lengths W_u, W_d, and W_ℓ, respectively. Since G has $\Theta(n^2)$ many paths, one can precompute the set of all possible path weights and use this set to prune the dynamic programming table, resulting in a polynomial-time algorithm. However, the degree of the polynomial will be large because the table can have $\Omega(n \cdot n^2 \cdot n^2 \cdot n^2) = \Omega(n^7)$ entries and computing any entry involving a node v may take $\Omega(n^6 \cdot \deg(v))$ time, where $\deg(v)$ is the number of children of v, if done by directly looking at the entries for (W_u, W_d, W_ℓ)-triples for each child of v. The rest of the paper is devoted to developing much faster algorithms for solving SLPO_s and SLPO_m on path graphs, cycle graphs, and star graphs. See the following table for a summary of our algorithms' time complexities.

Graph class	SLPO_s	SLPO_m	Section	Theorems
Tree	(Polynomial time)	(Polynomial time)	1.2	–
Path graph	$O(n)$	$O(n \log n)$	4	4 and 5
Cycle graph	$O(n)$	$O(n^2 \log n)$	5	6 and 7
Star graph	$O(n \log n)$	$O(n \log n)$	6	9 and 8

The paper is organized as follows. Section 2 defines SLPO_s and SLPO_m formally and lists some useful properties. The NP-hardness result is presented in Sect. 3. We describe our fast algorithms for path graphs in Sect. 4, for cycle graphs in Sect. 5, and for star graphs in Sect. 6. Due to space constraints, several correctness proofs have been omitted from the conference version of this paper. They will be available in the journal version.

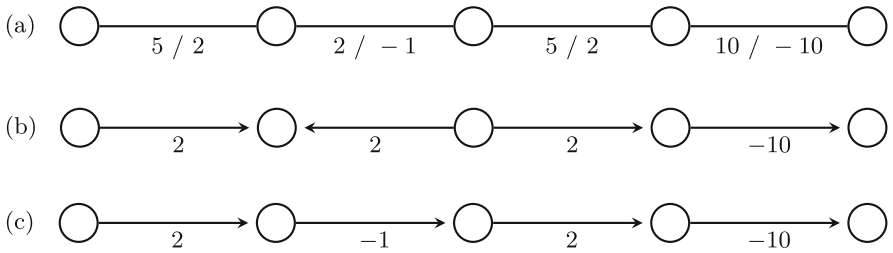

Fig. 1. An example. Let G be the undirected graph in (a). For every edge $\{u, v\}$, its two weights $w(u, v)$ and $w(v, u)$ are specified as $w(v, u)$ / $w(u, v)$. The orientation in (b) satisfies $h_s(\widetilde{G}) = h_m(\widetilde{G}) = 2$, while the one in (c) satisfies $h_s(\widetilde{G}) = 3$, $h_m(\widetilde{G}) = -7$. They are optimal under h_s and h_m, respectively, so $H_s(G) = 2$ and $H_m(G) = -7$.

2 Preliminaries

An *orientation* of an undirected graph $G = (V, E)$ is a directed graph $\widetilde{G}$ obtained by replacing each undirected edge $\{u, v\} \in E$ by either the directed edge (u, v) or the directed edge (v, u). Denote by $\mathcal{O}(G)$ the set of all orientations of G.

An *edge-bi-weighted graph* is an undirected graph $G = (V, E)$ in which every edge $\{u, v\} \in E$ has a pair of possibly nonpositive weights $w(u, v)$ and $w(v, u)$ associated with the two directions (u, v) and (v, u), respectively. The *length* of a directed edge (u, v) in an orientation of an edge-bi-weighted graph is $w(u, v)$, and the *length* of a path $\vec{P} = \langle v_1, \ldots, v_q \rangle$ is:

$$W(\vec{P}) = \sum_{k=1}^{q-1} w(v_k, v_{k+1}). \tag{1}$$

A path in a directed graph is said to be *maximal* if it is not contained in any path with more edges. As pointed out in Sect. 1.2, if some edge lengths are negative in a directed graph then a longest path is not necessarily a maximal path. We therefore employ two ways of measuring the cost of orienting a graph.

Definition 1. *Define the following two cost measures for an oriented graph* $\widetilde{G}$*:*

$$h_s(\widetilde{G}) = \max\{W(\vec{P}) \mid \vec{P} \text{ is a simple directed path in } \widetilde{G}\}, \tag{2}$$

$$h_m(\widetilde{G}) = \max\{W(\vec{P}) \mid \vec{P} \text{ is a maximal simple directed path in } \widetilde{G}\}. \tag{3}$$

The corresponding cost functions for orienting an undirected graph are:

$$H_s(G) = \min\{h_s(\widetilde{G}) \mid \widetilde{G} \in \mathcal{O}(G)\}, \tag{4}$$

$$H_m(G) = \min\{h_m(\widetilde{G}) \mid \widetilde{G} \in \mathcal{O}(G)\}. \tag{5}$$

Figure 1 illustrates the difference between $H_s(G)$ and $H_m(G)$.

The two problem variants that we consider in this paper 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).

In other words, SLPO_s and SLPO_m ask for orientations of G such that the length of a longest simple path in the resulting directed graph is minimized, using the two alternative definitions of a "longest path".

We now state some easily verified properties of the cost measures. Note that the empty path participates in determining the value of $h_s(\widetilde{G})$, so $h_s(\widetilde{G}) \geq 0$ for any $\widetilde{G}$ and $H_s(G) \geq 0$. Another useful property of H_s is its monotonicity:

Lemma 1. *If G' is a subgraph of an edge-bi-weighted graph G then $H_s(G') \leq H_s(G)$.*

H_m is not monotone in general, but if all weights are nonnegative then H_m is monotone as well. In fact, if all weights are nonnegative then the definitions of H_m and H_s coincide:

Lemma 2. *If all weights of an edge-bi-weighted graph G are nonnegative then $h_s(\widetilde{G}) = h_m(\widetilde{G})$ for any orientation $\widetilde{G}$ of G. In particular, $H_s(G) = H_m(G)$, and $\widetilde{G}$ is an optimal orientation of G with respect to h_s if and only if it is optimal with respect to h_m.*

3 NP-Hardness for Subcubic Planar Graphs

The following theorem is proved by a polynomial-time reduction from Planar 3-SAT restricted to instances where each variable occurs in at most four clauses, which is known to be NP-hard [14]. For the details of the reduction, the reader is referred to the journal version of this paper.

Theorem 1. *SLPO_s is NP-hard even if restricted to subcubic planar graphs where all edge weights belong to $\{0, 1, 2\}$.*

By Lemma 2, we immediately obtain:

Corollary 1. *SLPO_m is NP-hard even if restricted to subcubic planar graphs where all edge weights belong to $\{0, 1, 2\}$.*

The reduction in the proof of Theorem 1 constructs a subcubic planar graph G such that if one could determine whether $H_s(G) = 2$ or $H_s(G) \geq 3$ then one would know if the instance of Planar 3-SAT is satisfiable or not. This yields:

Theorem 2. *If, for any constant $\varepsilon > 0$, there exists a polynomial-time $(3/2-\varepsilon)$-approximation algorithm for SLPO_s or SLPO_m (even if restricted to subcubic planar graphs) then $\text{P} = \text{NP}$.*

4 Algorithms for Path Graphs

A *path graph* is an undirected, connected graph in which two vertices have degree 1 and all other vertices have degree 2. In Sect. 4.1 below, we first give a generic dynamic programming algorithm for finding the cost of an optimal orientation of an edge-bi-weighted path graph L. A straightforward implementation of the algorithm runs in $O(n^2)$ time. This high-level version makes no use of the details of the cost function, be it H_m or H_s. Then, Sects. 4.2 and 4.3 present faster implementations that take into account the specifics of H_s and H_m, thereby improving the time complexity to $O(n)$ and $O(n \log n)$, respectively.

4.1 A Generic Algorithm for Path Graphs

Given a path graph L with n vertices, assume without loss of generality that its vertices are numbered from 1 to n. For simplicity, we present an algorithm that computes the cost of an optimal orientation only; a corresponding optimal orientation can be found by adding a traceback step at no increase in the asymptotic time complexity. The algorithm is named BestOrientPath$_x(L)$ and its pseudocode is listed in Algorithm 1.

Algorithm 1: BestOrientPath$_x(L)$

Input: an edge-bi-weighted path graph L with n vertices
Output: an optimal orientation of L under H_x

if $n = 1$ **then**
  return 0;
$H_x^{\rightarrow}(1) = H_x^{\leftarrow}(1) = -\infty$;
for $j = 2$ *to* n **do**
  $H_x^{\rightarrow}(j) = \min_{1 \le i < j} \max\{H_x^{\leftarrow}(i), h_x(\vec{L}_{i,j})\}$;
  $H_x^{\leftarrow}(j) = \min_{1 \le i < j} \max\{H_x^{\rightarrow}(i), h_x(\overleftarrow{L}_{i,j})\}$;
return $\min\{H_x^{\rightarrow}(n), H_x^{\leftarrow}(n)\}$;

We use the following notation.

- $L_{i,j}$ is the subgraph of L induced by the vertices $i, \ldots, j$.
- $\vec{L}_{i,j}$ and $\overleftarrow{L}_{i,j}$ are the oriented versions of $L_{i,j}$ in which all edges are directed towards larger numbered vertices (i.e., of the form $(i, i+1)$) and towards smaller numbered vertices (i.e., of the form $(i+1, i)$), respectively.
- The subscript x satisfies $x \in \{s, m\}$.
- $H_x^{\rightarrow}(i)$ is the value of an optimal orientation of $L_{1,i}$ under H_x assuming that edge $\{i-1, i\}$ is directed towards i. Analogously, $H_x^{\leftarrow}(i)$ is the value of an optimal orientation of $L_{1,i}$ under H_x assuming that edge $\{i-1, i\}$ is directed towards $i-1$. In particular, the cost of an optimal orientation of L under H_x is given by $\min\{H_x^{\rightarrow}(n),\ H_x^{\leftarrow}(n)\}$.

To compute $H_x^{\rightarrow}(n)$ and $H_x^{\leftarrow}(n)$, we compute $H_x^{\rightarrow}(j)$ and $H_x^{\leftarrow}(j)$ using dynamic programming from $j = 2$ up to $j = n$. The idea is to locate, in an optimal orientation of $L_{1,j}$, the largest vertex i at which there is a change in direction given that the last edge $\{j-1, j\}$ has a specified direction.

Theorem 3. *BestOrientPath$_x$ finds the cost of an optimal orientation of L.*

4.2 Running Time Under Cost Function H_s

Here, we show that Algorithm BestOrientPath$_s$ can be made to run in $O(n)$ time.

The first issue is computing $h_s(\vec{L}_{i,j})$ and $h_s(\overleftarrow{L}_{i,j})$ for given $1 \leq i < j \leq n$. The weight of an edge $\{i, i+1\}$ when oriented as $(i, i+1)$ is denoted by $w(i, i+1)$ and when oriented as $(i+1, i)$ by $w(i+1, i)$. Equations (1) and (2) give:

$$h_s(\vec{L}_{i,j}) = \max \left\{ \sum_{t=i'}^{j'-1} w(t, t+1) \mid i \leq i' \leq j' \leq j \right\}. \tag{6}$$

Computing $h_s(\vec{L}_{i,j})$ for a pair (i, j) is therefore an instance of the Range Maximum-sum Segment Online Query problem, RMSOQ for short [4]:

Problem 1 (Range Maximum-sum Segment Online Query).
Input to be preprocessed: A nonempty sequence $a_1, \ldots, a_n$ of real numbers.
Online query: respond to a query of the form $RMSOQ(i, j)$ by returning a pair of indices (i', j') that maximizes $\sum_{t=i'}^{j'} a_t$ over all $i \leq i' \leq j' \leq j$.

Chen and Chao [4] presented a method for answering each such query in constant time after A has been preprocessed in $O(n)$ time. This gives:

Lemma 3. *Suppose $w(i, i+1), 1 \leq i < n$ and $w(i+1, i), 1 \leq i < n$ have been preprocessed in linear time for the RMSOQ problem. After $H_s^{\leftarrow}(i)$ and $H_s^{\rightarrow}(i)$ have been computed for $1 \leq i < j$, each value of the form $\max\{H_s^{\leftarrow}(i), h_s(\vec{L}_{i,j})\}$ and $\max\{H_s^{\rightarrow}(i), h_s(\overleftarrow{L}_{i,j})\}$ appearing in steps 5 and 6 in iteration j can be evaluated in constant time.*

Next, we address the second issue: what is the time needed to find all minimum values in steps 5 and 6 in Algorithm BestOrientPath$_s(L)$? To answer the question, first define two $(n \times n)$-matrices $M^{\rightarrow}(i, j)$ and $M^{\leftarrow}(i, j)$ by:

- $M^{\rightarrow}[1, j] = h_s(\vec{L}_{1,j})$ and $M^{\leftarrow}[1, j] = h_s(\overleftarrow{L}_{1,j})$ for $2 \leq j \leq n$
- $M^{\rightarrow}[i, j] = M^{\leftarrow}[i, j] = \infty$ for $1 \leq j \leq i \leq n$
- $M^{\rightarrow}[i, j] = \max\{H_s^{\leftarrow}(i), h_s(\vec{L}_{i,j})\}$ and $M^{\leftarrow}[i, j] = \max\{H_s^{\rightarrow}(i), h_s(\overleftarrow{L}_{i,j})\}$ for $2 \leq i < j \leq n$

In these terms, the algorithm computes the minimum value in column j of the matrices $M^{\rightarrow}$ and $M^{\leftarrow}$ for $1 \leq j \leq n$. Two features of this computation deserve particular attention. The first is that before computing the minimum

value in column j of the matrix $M^{\rightarrow}$, or $M^{\leftarrow}$, the minimum values of all previous columns $j' < j$, $H_s^{\leftarrow}[j']$, respectively $H_s^{\rightarrow}[j']$, must have been computed already. The second feature we want to point out is that it follows from Eq. (6) that both matrices $M^{\rightarrow}$ and $M^{\leftarrow}$ are of the form $M[i,j] = \max\{f(i), g(i,j)\}$, with g non-increasing in i and non-decreasing in j. This leads to the next proposition, where a matrix M is called *totally r-monotone* if it has the following property: If $M[i_1, j_1] \geq M[i_2, j_1]$ then $M[i_1, j_2] \geq M[i_2, j_2]$ for all $i_1 < i_2$ and $j_1 < j_2$.

Proposition 1. *If $g(i,j)$ is non-increasing in i and non-decreasing in j and $M(i,j) = \max\{f(i), g(i,j)\}$ then M is totally r-monotone.*

Proof. Suppose that $M[i_1, j_1] \geq M[i_2, j_1]$ holds. Then $\max\{f(i_1), g(i_1, j_1)\} \geq \max\{f(i_2), g(i_2, j_1)\}$. In particular, we have $f(i_2) \leq \max\{f(i_1), g(i_1, j_1)\} \leq \max\{f(i_1), g(i_1, j_2)\}$ since g is non-decreasing in j. Moreover, $g(i_1, j_2) \geq g(i_2, j_2)$. Hence $\max\{f(i_1), g(i_1, j_2)\} \geq \max\{f(i_2), g(i_2, j_2)\}$. □

To find all minimum values in steps 5 and 6 of the algorithm, we apply a solution to the following problem.

Problem 2 (Online Column Minima of a Totally r-Monotone Matrix). For $1 \leq j \leq n$, compute $H(j) = \min\{M(i,j) \mid 1 \leq i \leq n\}$, where M is totally r-monotone and the values of $H(j'), j' < j$ have to be computed before $M(i,j)$ can be evaluated.

Several linear-time online algorithms for solving Problem 2 exist [9,16,17].

Theorem 4. *BestOrientPath$_x$ with cost function H_s runs in $O(n)$ time.*

4.3 Running Time Under Cost Function H_m

We now make BestOrientPath$_m$ run in $O(n \log n)$ time. Combining (1) and (3):

$$h_m(\vec{L}_{i,j}) = W_j - W_i = \sum_{t=i}^{j-1} w(t, t+1), \tag{7}$$

where $W_1 = 0$ and $W_j = \sum_{t=1}^{j-1} w(t, t+1)$ for $j \geq 2$. A similar equality holds for $h_m(\overleftarrow{L}_{i,j})$. Consequently, finding the minimum values in steps 5 and 6 takes on a form different from the one obtained for H_s; more precisely, Eq. (7) shows that for $2 \leq j \leq n$, $H_m^{\rightarrow}(j) = \min_{1 \leq i < j} \max\{H_m^{\leftarrow}(i), W_j - W_i\}$. We split the points of the interval $1 \leq i < j$ into those that satisfy $H_m^{\leftarrow}(i) + W_i \geq W_j$ and the remainder. Since $H_m^{\leftarrow}(1) = -\infty$, the point $i = 1$ belongs to the latter. Also, $H_m^{\rightarrow}(2) = W_2$, and for $j \geq 3$, $H_m^{\rightarrow}(j) = \min\{M_1(j), M_2(j)\}$ with:

$$M_1(j) = \min_{2 \leq i < j} \{H_m^{\leftarrow}(i) \mid H_m^{\leftarrow}(i) + W_i \geq W_j\}, \tag{8}$$

$$M_2(j) = \min_{1 \leq i < j} \{W_j - W_i \mid H_m^{\leftarrow}(i) + W_i < W_j\}. \tag{9}$$

Next, we consider how to efficiently compute the minima in Eq. (8). Equation (9) can be handled similarly. We rephrase the problem as:

Problem 3 (Minima of Sequence Prefixes under Key-Bounds).

Given: A sequence KV of n pairs of the form $(key, value)$, and a sequence of lower bounds W_j, $1 \leq j \leq n$,

Compute: $min_j(W_j) = \min\{value \mid (key, value) \in Pre_j \, and \, key \geq W_j\}$, for $1 \leq j \leq n$, where Pre_j is the prefix of length j of KV.

Problem 3 can be solved by red-black trees [12]; see the journal version for details. This gives:

Theorem 5. *BestOrientPath$_x$ with cost function H_m runs in $O(n \log n)$ time.*

5 Algorithms for Cycle Graphs

A *cycle graph* is an undirected, connected graph in which all vertices have degree 2. Given a cycle graph C on n vertices, we fix a numbering of its vertices by assigning the number 0 to an arbitrarily selected vertex and then assigning the numbers 1 through $n-1$ to the remaining vertices in the order that they are visited by a depth-first traversal starting at 0. Denote the weight of a directed edge $(i, i+1)$ by $w(i, i+1)$. When a vertex j with $j \geq n$ is referred to, it should be understood as referring to vertex $j \bmod n$. For example, $w(n-1, n) = w(n-1, 0)$.

5.1 An Algorithm for Cost Function H_s

Our algorithm for cycle graphs under H_s, BestOrientCycle$_s$, employs the following strategy. It first creates a special path graph L from the input cycle graph C and then applies BestOrientPath$_s$ from Sect. 4.2 to L to find an optimal orientation $\widetilde{L}^*$ of L. An optimal orientation of C will always be one of five possible orientations that depends on the structure of $\widetilde{L}^*$, so the algorithm simply checks which one of five conditions holds and outputs the corresponding orientation of C.

The pseudocode is given in Algorithm 2. It uses the following notation.

Definition 2. *Let C be a cycle graph, L the path graph constructed in step 1 of BestOrientCycle$_s(C)$, and $\widetilde{L}^*$ an optimal orientation of L. The five orientations of C denoted by $\widetilde{C}^{*1way}$, $\widetilde{C}^{2+xx}$, $\widetilde{C}^{i}$, $\widetilde{C}^{rflip_i}$, and $\widetilde{C}^{lflip_i}$ are defined as:*

- *$\widetilde{C}^{*1way}$ is a one-way orientation of C whose cost, $OneWayCost_s$, is the least.*
- *For a cycle graph of odd length, $\widetilde{C}^{2+xx}$ is the orientation of C obtained as follows: among all possible directed paths of length two find one, $\overleftrightarrow{L_2}$, whose weight is minimal; starting with this path, direct each successive edge of the cycle graph in the direction opposite to that of its predecessor.*

– $\widetilde{C}^{i}$ *is the orientation of* C *obtained by copying from* $\widetilde{L}^{*}$ *the directions of the edges* $\{k, k+1\},\ i \leq k < i+n$.

– $\widetilde{C}^{rflip_i}$ *is created when* $0 \leq i \leq 2n$ *and* $\widetilde{L}^{*}$ *has the directed edges* $(i, i+1), (i+1, i+2), (i+3, i+2)$. *Its first directed edge is* $(i+1, i)$, *and the directions of the following edges* $\{k, k+1\},\ i+1 \leq k < i+n$, *are copied from* $\widetilde{L}^{*}$.

– $\widetilde{C}^{lflip_i}$ *is created when* $n \leq i \leq 3n$ *and* $\widetilde{L}^{*}$ *has the directed edges* $(i, i-1), (i-1, i-2), (i-3, i-2)$. *Its first directed edge is* $(i-1, i)$, *and the directions of the following edges* $\{k, k-1\},\ i-n < k \leq i-1$, *are copied from* $\widetilde{L}^{*}$.

The algorithm's time complexity is linear because running BestOrientPath$_s$ in step 2 takes $O(n)$ time according to Theorem 4 and each of the other steps is easy to implement in $O(n)$ time.

Theorem 6. *BestOrientCycle$_s$ returns an orientation for a cycle graph that is optimal under the cost function* H_s *in* $O(n)$ *time.*

5.2 An Algorithm for Cost Function H_m

A simple method that works for cycle graphs under the cost function H_m is shown in Algorithm 3 (BestOrientCycle$_m$). It tries all ways of breaking the input cycle graph into a path graph by cutting one edge, applies BestOrientPath$_m$ to each such obtained path graph, and chooses one with the least cost. (This approach

Algorithm 2: BestOrientCycle$_s(C)$

Input: an edge-bi-weighted cycle graph C
Output: an optimal orientation of C under H_s

1 create a path graph L of length $3n$ by unrolling the cycle graph three times starting from vertex 0, numbering its vertices *0* to *3n*, and assigning each edge $\{i, i+1\}$ of L the same weights as edge $\{i, i+1\}$ of C;
2 let $\widetilde{L}^{*}$ be the oriented graph returned by BestOrientPath$_s(L)$;
3 **if** $h_s(\widetilde{L}^{*}) \geq OneWayCost_s$ **then** set $\widetilde{C}$ to $\widetilde{C}^{*1way}$;
4 **else if** *n is odd and every two consecutive edges in* $\widetilde{L}^{*}$ *have opposite directions* **then** set $\widetilde{C}$ to $\widetilde{C}^{2+xx}$;
5 **else if** *two edges* $\{i, i+1\}$ *and* $\{i+n-1, i+n\}$ *have opposite directions in* $\widetilde{L}^{*}$ **then** set $\widetilde{C}$ to $\widetilde{C}^{i}$;
6 **else if** *there is an* i, $0 \leq i \leq 2n$, *such that the directed edges* $(i, i+1), (i+1, i+2), (i+3, 1+2)$ *appear in* $\widetilde{L}^{*}$ **then** set $\widetilde{C}$ to $\widetilde{C}^{rflip_i}$;
7 **else** find i, $n \leq i \leq 3n$, such that $\widetilde{L}^{*}$ contains the directed edges $(i, i-1), (i-1, i-2), (i-3, i-2)$, and set $\widetilde{C}$ to $\widetilde{C}^{lflip_i}$;
8 **return** $\widetilde{C}$;

would also work for the cost function H_s, but the resulting time complexity would be worse than the one in Theorem 6.)

In the pseudocode, $\widetilde{C}^{*1way}$ denotes the one-way orientation of C whose cost, $OneWayCost_m$, is the least. Also, $L_{i+1,i-1}$ for $0 \leq i \leq n$ is the subgraph of C induced by vertices $i+1, i+2, \ldots, i-1$, with sums taken mod n.

Theorem 7. *Algorithm BestOrientCycle$_m(C)$ returns an optimal orientation of C in $O(n^2 \log n)$ time.*

Proof. Let $\widetilde{C}^*$ be an optimal orientation of C. If $\widetilde{C}^*$ is a directed cycle then an optimal solution will be found in step 1. Otherwise, $\widetilde{C}^*$ has at least one vertex i such that both edges $\{i-1, i\}$ and $\{i, i+1\}$ are oriented away from i. Consider the path graph L^i constructed in iteration i. Denote by $\widetilde{L}^i$ the orientation of L^i induced by breaking the cycle at vertex i, and note that $h_m(\widetilde{L}^i) = h_m(\widetilde{C}^*)$. Let $\widetilde{L}^{i*}$ be an optimal orientation of L^i. $\widetilde{L}^{i*}$ induces an orientation $\widetilde{C}$ of C by identifying i' with i''. Then $h_m(\widetilde{C}) = h_m(\widetilde{L}^{i*}) \leq h_m(\widetilde{L}^i) = h_m(\widetilde{C}^*)$. Since $h_m(\widetilde{C}) \geq h_m(\widetilde{C}^*)$, it follows that $h_m(\widetilde{L}^{i*}) = h_m(\widetilde{C}^*)$.

Because one call to BestOrientPath$_m$ takes $O(n \log n)$ time by Theorem 5, the algorithm runs in $O(n^2 \log n)$ time. □

Algorithm 3: BestOrientCycle$_m(C)$

Input: an edge-bi-weighted cycle graph C
Output: an optimal orientation of C under H_m

1 set $\widetilde{C}$ to $\widetilde{C}^{*1way}$ and set *BestCost* to $OneWayCost_m$;
2 **for** $i = 0$ *to* $n-1$ **do**
3 construct a path graph L^i by adding two vertices i' and i'' and two edges $\{i+1, i'\}$ and $\{i-1, i''\}$ to $L_{i+1,i-1}$ with edge weights $w_{L^i}(i+1, i') = \infty$, $w_{L^i}(i', i+1) = w(i, i+1)$, $w_{L^i}(i-1, i'') = \infty$, and $w_{L^i}(i'', i-1) = w(i, i-1)$;
4 let $\widetilde{L}^i$ be the graph returned by BestOrientPath$_m(L^i)$;
5 **if** $h_m(\widetilde{L}^i) <$ *BestCost* **then**
6 set $\widetilde{C}$ to the orientation of C induced by $\widetilde{L}^i$ and *BestCost* to $h_m(\widetilde{L}^i)$;
7 **return** $\widetilde{C}$;

6 Algorithms for Star Graphs

A *star graph* is a tree with exactly one internal node and at least two leaves. In this section, the internal node of a given star graph is denoted by c.

6.1 An Algorithm for Cost Function H_m

Algorithm BestOrientStar$_m$ for SLPO$_m$ on star graphs is given in Algorithm 4. It initially orients each edge so that it points in its lighter direction and then refines this solution to obtain an optimal orientation. To do so, it either flips edges that were initially pointing inwards only or edges that were initially pointing outwards only. The correctness of this approach is guaranteed by:

Lemma 4. *There is no optimal orientation of a star graph in which both the largest inward edge weight is less than $w(u_1, c)$ and the largest outward edge weight is less than $w(c, v_1)$.*

Proof. The cost of the initial orientation is $h_m(\widetilde{S}) = w(u_1, c) + w(c, v_1)$. For the purpose of obtaining a contradiction, suppose that there is an optimal orientation $\widetilde{S}^*$ with largest inward weight $w(x, c) < w(u_1, c)$ and largest outward weight $w(c, y) < w(c, v_1)$. Then $h_m(\widetilde{S}^*) = w(x, c) + w(c, y)$. The initial orientation implies that $w(c, u_1) > w(u_1, c)$, and $w(v_1, c) \geq w(c, v_1)$. Since $\{u_1, c\}$ is an outward edge in $\widetilde{S}^*$ and $\{v_1, c\}$ an inward edge, $h_m(\widetilde{S}^*) \geq w(v_1, c) + w(c, u_1) > w(u_1, c) + w(c, v_1) = h_m(\widetilde{S})$, which is impossible. □

Theorem 8. *BestOrientStar$_m$ solves SLPO$_m$ on star graphs in $O(n \log n)$ time.*

6.2 An Algorithm for Cost Function H_s

Based on the observation in the next lemma, we can use BestOrientStar$_m$ from Sect. 6.1 as a subroutine to obtain a solution for SLPO$_s$ on star graphs, as shown in Algorithm 5 (BestOrientStar$_s$).

Algorithm 4: BestOrientStar$_m(S)$

Input: an edge-bi-weighted star graph S
Output: an optimal orientation of S under H_m

orient each edge $\{u, c\}$ inwards to c if $w(u, c) < w(c, u)$, and outwards from c otherwise;
denote by $\widetilde{S}$ the resulting directed graph, by *BestCost* its cost,
by $E_{in} = \{(u_1, c), \ldots, (u_\ell, c)\}$ the list of its inward edges,
and by $E_{out} = \{(c, v_1), \ldots, (c, v_r)\}$ the list of its outward edges;
reorder each of E_{in} and E_{out} so that the weights of its edges are in non-increasing order;
set $\widetilde{S}'$ to $\widetilde{S}$;
for $k = 1$ *to* ℓ **do**
    flip the direction of edge (u_k, c) in $\widetilde{S}'$ and update *BestCost* if necessary;
set $\widetilde{S}'$ to $\widetilde{S}$;
for $k = 1$ *to* r **do**
    flip the direction of edge (c, v_k) in $\widetilde{S}'$ and update *BestCost* if necessary;
return an orientation whose cost is *BestCost*;

Lemma 5. *Suppose S has a vertex u with an edge to or from c with non-positive weight. Let S' be the star graph obtained by removing u from S. Then an optimal orientation of S is obtained by first optimally orienting S', and then adding to it the vertex u with its edge $\{u, c\}$ oriented in a direction with non-positive weight.*

Proof. The number of edges on a directed path in an oriented star graph is no more than two. An edge with non-positive weight is therefore either the first or the last edge on any directed path, and does not contribute to its cost. □

Theorem 9. *BestOrientStar$_s$ solves SLPO$_s$ on star graphs in $O(n \log n)$ time.*

Algorithm 5: BestOrientStar$_s(S)$

Input: an edge-bi-weighted star graph S
Output: an optimal orientation of S under H_s

1 remove from S every edge with a direction of non-positive weight, and denote the resulting star graph S';
2 set $\widetilde{S}'$ to BestOrientStar$_m(S')$;
3 direct all edges removed in step 1 in a direction of non-positive weight and add them to $\widetilde{S}'$, and denote the resulting directed star graph $\widetilde{S}$;
4 **return** $\widetilde{S}$;

References

1. Asahiro, Y., Jansson, J., Miyano, E., Ono, H.: Graph orientations optimizing the number of light or heavy vertices. J. Graph Algorithms 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. Borradaile, G., Iglesias, J., Migler, T., Ochoa, A., Wilfong, G., Zhang, L.: Egalitarian graph orientations. J. Graph Algorithms Appl. **21**(4), 687–708 (2017)
4. Chen, K.Y., Chao, K.M.: On the range maximum-sum segment query problem. Discret. Appl. Math. **155**(16), 2043–2052 (2007)
5. Chrobak, M., Eppstein, D.: Planar orientations with low out-degree and compaction of adjacency matrices. Theoret. Comput. Sci. **86**(2), 243–266 (1991)
6. Dailey, D.P.: Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. Discret. Math. **30**(3), 289–293 (1980)
7. Deming, R.W.: Acyclic orientations of a graph and chromatic and independence numbers. J. Comb. Theory B **26**(1), 101–110 (1979)
8. Elberfeld, M., et al.: On the approximability of reachability-preserving network orientations. Internet Math. **7**(4), 209–232 (2011)
9. Galil, Z., Park, K.: A linear-time algorithm for concave one-dimensional dynamic programming. Inf. Process. Lett. **33**(6), 309–311 (1990)
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. Garey, M., Johnson, D.: Computers and Intractability - A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
12. Guibas, L.J., Sedgewick, R.: A dichromatic framework for balanced trees. In: Proceedings of the Nineteenth Annual Symposium on Foundations of Computer Science (FOCS 1978), pp. 8–21. IEEE Computer Society (1978)
13. Hasse, M.: Zur algebraischen Begründung der Graphentheorie. I. Mathematische Nachrichten **28**(5–6), 275–290 (1965)
14. Jansen, K., Müller, H.: The minimum broadcast time problem for several processor networks. Theoret. Comput. Sci. **147**(1–2), 69–85 (1995)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Proceedings of Complexity of Computer Computations, pp. 85–103. The IBM Research Symposia Series, Plenum Press (1972)
16. Klawe, M.M.: A simple linear time algorithm for concave one-dimensional dynamic programming. Technical Report 89–16, Department of Computer Science, University of British Columbia (1989)
17. Larmore, L.L., Schieber, B.: On-line dynamic programming with applications to the prediction of RNA secondary structure. J. Algorithms **12**(3), 490–515 (1991)
18. 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
19. Roy, B.: Nombre chromatique et plus longs chemins d'un graphe. Revue française d'informatique et de recherche opérationnelle **1**(5), 129–132 (1967)
20. Venkateswaran, V.: Minimizing maximum indegree. Discret. Appl. Math. **143**(1–3), 374–378 (2004)
21. Vitaver, L.M.: Determination of minimal coloring of vertices of a graph by means of Boolean powers of the incidence matrix (in Russian). In: Proceedings of the USSR Academy of Sciences, vol. 147, pp. 758–759. Nauka (1967)
22. Zuckerman, D.: Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. Theory Comput. **3**(1), 103–128 (2007)