Journal of Combinatorial Optimization (2026) 52:20
https://doi.org/10.1007/s10878-026-01438-z

Shortest longest-path graph orientations

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

Received: 15 April 2024 / Accepted: 13 June 2026

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

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 interesting algorithmic problems can be defined. As a simple example, the problem of finding an *acyclic* orientation of an undirected graph, i.e., an orientation such that the resulting directed graph has no directed cycles, is trivially solvable in linear time: just enumerate the vertices in any order and then orient every edge toward its higher-numbered vertex (Deming 1979).

A preliminary version of this article appeared in *Proceedings of the Twenty-Ninth Annual International Computing and Combinatorics Conference* (COCOON 2023), Lecture Notes in Computer Science, Vol. 14422, pp. 141–154, Springer Nature Switzerland AG, 2024. Funded by KAKENHI grant numbers JP21K11755, JP22K11915, JP24K02902, and JP24K22294.

Extended author information available on the last page of the article

Published online: 31 August 2026

Another example is the problem of finding an orientation of an undirected graph that minimizes the maximum outdegree among all vertices. This problem has applications to job scheduling and load balancing (Asahiro et al. 2011), telecommunications (Borradaile et al. 2017; Venkateswaran 2004), and data structures for supporting fast adjacency queries in sparse graphs (Chrobak and Eppstein 1991). It can be solved in polynomial time (Asahiro et al. 2007; Chrobak and Eppstein 1991; Venkateswaran 2004), even if the orientation is required to be acyclic (Borradaile et al. 2017), but becomes NP-hard when generalized to edge-weighted graphs and the objective is to minimize (over all orientations) the maximum sum (over all vertices) of the outgoing edges' weights (Asahiro et al. 2011). A third example is the Maximum Pairs Orientation problem (MPO) from Bioinformatics for inferring protein-protein interaction networks from knockout experiments (Medvedovsky et al. 2008). Here, the input is an undirected graph $G = (V, E)$ and a set $P = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of ordered pairs from V, and the objective is to find an orientation of G that maximizes the number of pairs from P that are connected by directed paths of the form $s_i \to \ldots \to t_i$, where $(s_i, t_i) \in P$. MPO is NP-hard in general but admits polynomial-time approximation algorithms that work well in practice (Elberfeld et al. 2011; Medvedovsky et al. 2008).

Certain graph orientation problems have turned out to be equivalent to well-known classic graph algorithmic problems. 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 by Asahiro et al. (2015), 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 Asahiro et al. (2015)).

In this article, 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 $H(G) + 1 = \chi(G)$ (Gallai 1966; Hasse 1965; Roy 1967; Vitaver 1967) and that this equality still holds when only acyclic orientations are permitted (Deming 1979). Note that since Minimum Graph Coloring is NP-hard (Karp 1972), this immediately implies that USLPO is NP-hard. Moreover, known inapproximability results apply directly as well; e.g., Theorem 1.2 of Zuckerman (2007) 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 $\chi(G) = 4$ (Dailey 1980).

1.2 New results and organization of the article

We extend the definition of USLPO to allow *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). From here on, the generalized problem is simply referred to as Shortest Longest-Path Orientation (SLPO). 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*.

Since SLPO generalizes USLPO, which is computationally equivalent to Minimum Graph Coloring (in the sense explained above), it cannot be computationally easier than the latter. The big question then is how much harder it is. Our first main result (Theorem 2) states that both SLPO_s and SLPO_m are NP-hard to approximate even if restricted to subcubic planar graphs, where an undirected graph G is called *subcubic* if every vertex in G has degree at most three. This result is important in view of the fact that Minimum Graph Coloring is actually polynomial-time solvable for subcubic graphs (Garey and Johnson 1979).

Motivated by the hardness of the general case, we then focus on special cases that are solvable in linear time for Minimum Graph Coloring to see if their computational complexity changes for SLPO. Throughout the article, 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 vertex 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 vertex 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 article is devoted to developing much faster algorithms for solving SLPO_s and SLPO_m on path graphs, cycle graphs, and star graphs. (A *path graph* is an undirected, connected graph in which two vertices have degree one and all other vertices have degree two, a *cycle graph* is an undirected, connected graph in which

all vertices have degree two, and a *star graph* is an undirected, unrooted tree with one internal vertex.)

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 article 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. Section 7 gives some concluding remarks.

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 $w(u, v)$. A *directed path* (or *path*, for short) in a directed graph is a sequence of vertices $\langle v_1, v_2, \ldots, v_q \rangle$, $1 \leq q$, such that for $k \in \{1, 2, \ldots, q-1\}$, the graph contains the directed edge (v_k, v_{k+1}), and the *length* of such a path is

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

with $W(\vec{P}) = 0$ for an empty path $\vec{P} = \langle v_1 \rangle$.

Let $\vec{P}$ be a path in a directed graph. If all vertices in $\vec{P}$ are distinct then $\vec{P}$ is called *simple*. Also, $\vec{P}$ is said to be *maximal* if it is not contained in any path with more vertices. As pointed out in Sect. 1.2, a longest path in a directed graph need not be a maximal path if some of the edge lengths are negative. We therefore employ two ways of measuring the cost of orienting a graph.

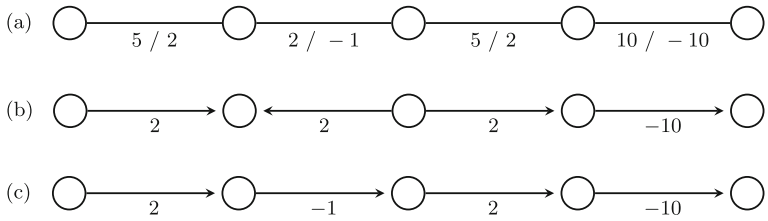

Fig. 1 An example. Let G be the undirected graph in (a). For every edge $\{u, v\}$, its two weights $w(u, v)$ and $w(v, u)$ are specified as $w(v, u)$ / $w(u, v)$. The orientation in (b) satisfies $h_s(\widetilde{G}) = h_m(\widetilde{G}) = 2$, while the one in (c) satisfies $h_s(\widetilde{G}) = 3$ and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

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:

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

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

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

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$_m$).

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 $H_s(G') \leq H_s(G)$.*

Proof By the definition (2) of h_s, we have $h_s(\vec{P}) \geq h_s(\vec{P}')$ for any subpath $\vec{P}'$ of $\vec{P}$. □

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.*

Proof From the definitions of H_m and H_s, it is clear that $h_m(\vec{P}) \leq h_s(\vec{P})$ for any directed path $\vec{P}$ and that the inequality is an equality if all the weights are nonnegative. This implies $h_s(\widetilde{G}) = h_m(\widetilde{G})$. □

3 NP-hardness for subcubic planar graphs

The main result of this section says that SLPO_s and SLPO_m are NP-hard to approximate even if restricted to subcubic planar graphs where all edge weights belong to $\{1, 2\}$ (Theorem 2). The NP-hardness is proved by a polynomial-time reduction from Planar One-In-Three 3-SAT (Dyer and Frieze 1986).

We first review the definition of Planar One-in-Three 3-SAT. The One-in-Three 3-SAT problem (see, e.g., Garey and Johnson (1979)) is a variant of 3-SAT that asks if there is a truth assignment to its input variables that makes each of its input clauses get exactly one true literal and two false literals. Planar One-in-Three 3-SAT is a further restriction of this problem in which a particular graph that is associated with the input CNF-formula is planar. More precisely, for any CNF-formula ϕ consisting of a collection $\mathcal{C}$ of clauses over a set X of variables, the *associated graph* $G_\phi = (V_\phi, E_\phi)$ is defined by:

$$V_\phi = X \cup \mathcal{C}$$
$$E_\phi = \{\{x, C\} \mid x \in C \text{ or } \overline{x} \in C \text{ for } x \in X \text{ and } C \in \mathcal{C}\}.$$

Planar One-in-Three 3-SAT takes as input a CNF-formula ϕ consisting of a collection of m clauses $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ over n variables $X = \{x_1, x_2, \ldots, x_n\}$ such that each clause contains exactly three variables and such that the associated graph G_ϕ admits a planar embedding (i.e., can be drawn in the plane with no edges intersecting each other except for in their endpoints), and asks whether there exists a truth assignment to X for which each clause in $\mathcal{C}$ has exactly one true literal. It is known that Planar One-in-Three 3-SAT is NP-hard (Dyer and Frieze 1986).

We now give a polynomial-time reduction from Planar One-in-Three 3-SAT to SLPO_s. Let ϕ be any instance of Planar One-in-Three 3-SAT. Fix a planar embedding of the associated graph G_ϕ. For every $1 \leq i \leq n$, let k_i be the number of clauses in ϕ that contain the literal x_i or $\overline{x_i}$ and assume without loss of generality that $k_i \geq 1$. Denote these k_i clauses by $C_{\ell(i,1)}, \ldots, C_{\ell(i,k_i)}$, where $C_{\ell(i,1)}$ is the first clause in ϕ that contains x_i or $\overline{x_i}$ and the rest of the sequence follows the counterclockwise order of the neighbors of the vertex x_i in the planar embedding of G_ϕ. Next, we describe

how to construct an undirected, edge-bi-weighted graph G consisting of (i) *variable gadgets* and (ii) *clause* gadgets that mimic variables and clauses of ϕ, respectively. The variable and clause gadgets are connected through (iii) a *variable-clause edge set*. As will be shown below, G has the property that $H_s(G) = 2$ if and only if ϕ is satisfiable.

(i) The variable gadgets are n subgraphs, denoted G_{x_1} through G_{x_n}, that represent the n variables $\{x_1, x_2, \ldots, x_n\}$ in ϕ. For each $1 \leq i \leq n$, the ith variable gadget G_{x_i} is a path graph $\langle x_{i,1}, \overline{x_{i,1}}, \ldots, x_{i,k_i}, \overline{x_{i,k_i}} \rangle$ of $2k_i$ vertices. That is,

$$V(G_{x_i}) = \{x_{i,1}, \overline{x_{i,1}}, \ldots, x_{i,k_i}, \overline{x_{i,k_i}}\}, \text{ and}$$
$$E(G_{x_i}) = \{\{x_{i,1}, \overline{x_{i,1}}\}, \{\overline{x_{i,1}}, x_{i,2}\}, \ldots, \{x_{i,k_i}, \overline{x_{i,k_i}}\}\}.$$

For $1 \leq i \leq n$ and all $\{u, v\} \in E(G_{x_i})$, both $w(u, v)$ and $w(v, u)$ are set to 2. See Fig. 2 for an illustration.

(ii) The clause gadgets are m subgraphs, denoted G_{C_1} through G_{C_m}, that represent the m clauses $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ in ϕ. For each $1 \leq j \leq m$, the jth clause gadget G_{C_j} is a cycle of three vertices $c_{j,1}$, $c_{j,2}$, and $c_{j,3}$. That is,

$$V(G_{C_j}) = \{c_{j,1}, c_{j,2}, c_{j,3}\}, \text{ and}$$
$$E(G_{C_j}) = \{\{c_{j,1}, c_{j,2}\}, \{c_{j,2}, c_{j,3}\}, \{c_{j,3}, c_{j,1}\}\}.$$

For $1 \leq j \leq m$ and all $\{u, v\} \in E(G_{C_j})$, both $w(u, v)$ and $w(v, u)$ are set to 1. See Fig. 3 for an illustration.

(iii) The variable-clause edge set contains three edges for each clause C_j, where $1 \leq j \leq m$, defined as follows. Suppose that C_j contains the variables $x_{i_1}, x_{i_2}, x_{i_3}$ and that in the fixed planar embedding of G_ϕ, the counterclockwise ordering of these variables around C_j is precisely $x_{i_1}, x_{i_2}, x_{i_3}$. First, consider x_{i_1}. Define p to be the index for which $j = \ell(i_1, p)$ holds. If x_{i_1} occurs as a positive literal in C_j then we let the variable-clause edge set have the edge $\{x_{i_1,p}, c_{j,1}\}$; otherwise, x_{i_1} occurs as a negative literal in C_j and we let the variable-clause edge set have the edge $\{\overline{x_{i_1,p}}, c_{j,1}\}$ instead. Thus, in both cases, the added edge connects the variable gadget $G_{x_{i_1}}$ to the clause gadget G_{C_j}. Set the edge weights in the first case so that $w(x_{i_1,p}, c_{j,1}) = 2$ and $w(c_{j,1}, x_{i_1,p}) = 1$, and in the second case so that $w(\overline{x_{i_1,p}}, c_{j,1}) = 2$ and $w(c_{j,1}, \overline{x_{i_1,p}}) = 1$. Next, treat x_{i_2} and x_{i_3} in the same way to obtain an edge connecting $G_{x_{i_2}}$ to G_{C_j} and an edge connecting $G_{x_{i_3}}$ to G_{C_j}. See Figs. 2 and 3 again for some examples of variable-clause edges, and Fig. 4 for an example of the entire reduction.

The construction clearly takes polynomial time. Moreover, G is planar because the associated graph G_ϕ is planar. It is not difficult to see that adding the variable-clause edge set does not break the planarity of G since every added edge lies outside of all the variable and clause gadgets; to maintain the counterclockwise order of variable-clause edges, we can make small adjustments, e.g., by rotating a gadget, stretching an edge in a gadget, and shifting the position of a vertex in a gadget. Also, according to the construction, the maximum (unweighted) degree of G is three and all edge weights belong to $\{1, 2\}$.

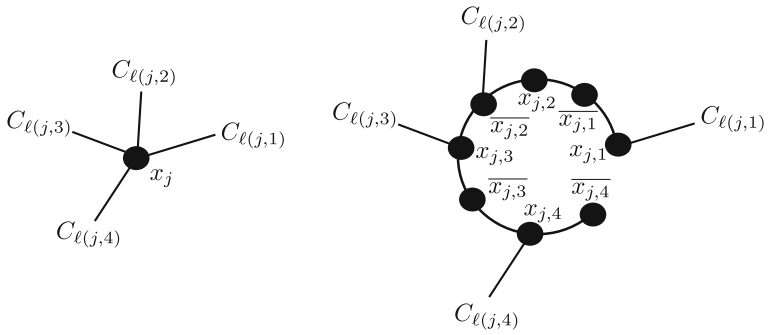

Fig. 2 Let x_j be a variable in ϕ with $k_j = 4$. (Left) The part of the fixed planar embedding of G_ϕ consisting of x_j and its neighborhood. The neighbors of x_j in G_ϕ are $C_{\ell(j,1)}, \ldots, C_{\ell(j,4)}$. (Right) The variable gadget G_{x_j} for x_j is a path graph with $2k_j = 8$ vertices and 7 edges. Four additional edges, not belonging to G_{x_j} itself, are also drawn here to demonstrate how G_{x_j} will be connected to the clause gadgets; in this example, $x_j \in C_{\ell(i,1)}$, $\overline{x_j} \in C_{\ell(j,2)}$, $x_j \in C_{\ell(j,3)}$, and $x_j \in C_{\ell(j,4)}$

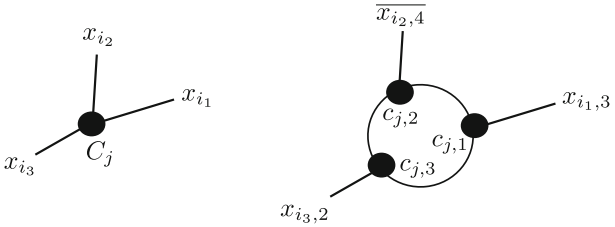

Fig. 3 Let C_j be a clause in ϕ. (Left) The part of the fixed planar embedding of G_ϕ consisting of C_j and its neighborhood. The neighbors of C_j in G_ϕ are $x_{i_1}, x_{i_2}, x_{i_3}$. (Right) The clause gadget G_{C_j} for C_j is a cycle graph with 3 vertices and 3 edges. Three additional edges, not belonging to G_{C_j} itself, are also drawn here to demonstrate how G_{C_j} will be connected to the variable gadgets; in this example, $C_j = x_{i_1} \vee \overline{x_{i_2}} \vee x_{i_3}$, with x_{i_1}, $\overline{x_{i_2}}$, and x_{i_3} being the third, the fourth, and the second occurrence of the variables x_{i_1}, x_{i_2}, and x_{i_3}, respectively

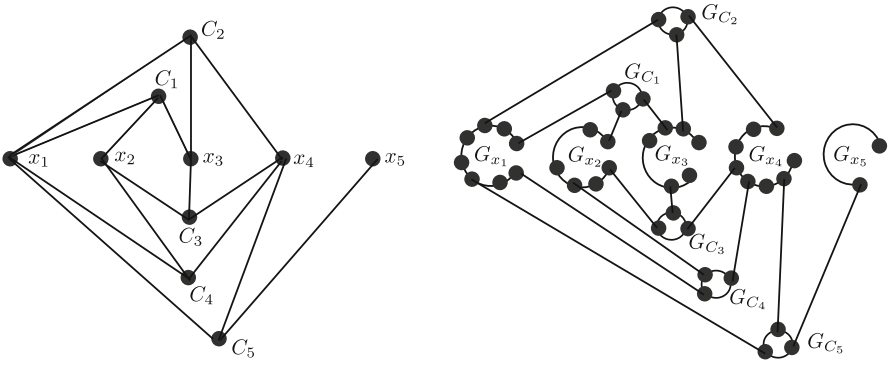

Fig. 4 An example of the reduction from PLANAR ONE-IN-THREE 3-SAT to SLPO$_s$. Suppose that $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_5})$ and that a planar embedding of the associated graph G_ϕ has been fixed as shown on the left. Here, $k_1 = 4$, $k_2 = 3$, $k_3 = 3$, $k_4 = 4$, and $k_5 = 1$, and, e.g., $\ell(1,1) = 1$, $\ell(1,2) = 2$, $\ell(1,3) = 5$, $\ell(1,4) = 4$. By combining the (i) variable gadgets G_{x_i}, (ii) the clause gadgets G_{C_j}, and (iii) the variable-clause edge set, we obtain the graph G on the right

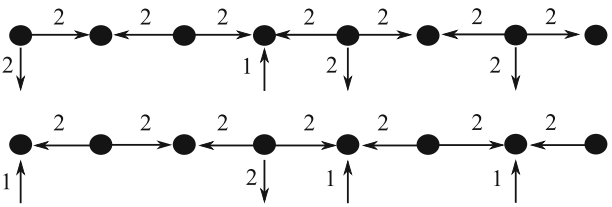

Fig. 5 The (Top) FB-orientation and (Bottom) BF-orientation of a variable gadget G_{x_i} with $k_i = 4$, drawn horizontally. The leftmost vertex is $x_{i,1}$ and the rightmost vertex is $\overline{x_{i,4}}$. Also shown here as vertical edges are four edges from the variable-clause edge set that do not belong to G_{x_i} itself but are connected to G_{x_i}. By point (iii) in the construction above, variable-clause edges get weight 2 when oriented away from G_{x_i} and weight 1 when oriented toward G_{x_i}

Next, we will present three lemmas (Lemmas 3–5) which are straightforward but play key roles in the proof that follows.

Lemma 3 *In any orientation $\widetilde{G}$ of G, $h_s(\widetilde{G}) \geq 2$.*

Proof Every edge in a variable gadget has both of its weights equal to 2. □

For any variable gadget G_{x_i}, the *FB-orientation* (short for "Forward-Backward") of G_{x_i} is defined as the orientation in which every edge in G_{x_i} is directed from the vertex of the form $x_{i,\ldots}$ to the vertex of the form $\overline{x_{i,\ldots}}$. Symmetrically, the *BF-orientation* ("Backward-Forward") of G_{x_i} is the orientation in which every edge is directed from the vertex of the form $\overline{x_{i,\ldots}}$ to the vertex of the form $x_{i,\ldots}$. See Fig. 5. In the special case where the number of occurrences of the variable x_i in ϕ is one, the variable gadget G_{x_i} only has a single edge $\{x_{i,1}, \overline{x_{i,1}}\}$, and this edge is oriented as $(x_{i,1}, \overline{x_{i,1}})$ in the FB-orientation and as $(\overline{x_{i,1}}, x_{i,1})$ in the BF-orientation.

Lemma 4 *To achieve $h_s(\widetilde{G}) = 2$, each variable gadget G_{x_i} must use either its FB-orientation or its BF-orientation. In addition, if it uses the FB-orientation then all variable-clause edges of the form $\{x_{i,\ldots}, c_{\ldots,\ldots}\}$ must be directed away from G_{x_i} and all variable-clause edges of the form $\{\overline{x_{i,\ldots}}, c_{\ldots,\ldots}\}$ toward G_{x_i}, and if it uses the BF-orientation then all variable-clause edges of the form $\{\overline{x_{i,\ldots}}, c_{\ldots,\ldots}\}$ must be directed away from G_{x_i} and all variable-clause edges of the form $\{x_{i,\ldots}, c_{\ldots,\ldots}\}$ toward G_{x_i}.*

Proof If two adjacent edges in G_{x_i}, say $\{u, v\}$ and $\{v, w\}$, were oriented in the same direction then the length of the resulting directed path $\langle (u, v), (v, w) \rangle$ or $\langle (w, v), (v, u) \rangle$ would be 4, which would give $h_s(\widetilde{G}) \geq 4$.

As for the second part of the lemma statement, in the case of an FB-orientation, a directed edge of the form $(c_{\ldots,\ldots}, x_{i,\ldots})$ would create a path of length $1 + 2 = 3$, and a directed edge of the form $(\overline{x_{i,\ldots}}, c_{\ldots,\ldots})$ would create a path of length $2 + 2 = 4$. Symmetrically, in the case of a BF-orientation, a directed edge of the form $(c_{\ldots,\ldots}, \overline{x_{i,\ldots}})$ would create a path of length 3, and a directed edge of the form $(x_{i,\ldots}, c_{\ldots,\ldots})$ would create a path of length 4. □

The three edges in the variable-clause edge set incident to any clause gadget G_{C_j} can be oriented according to either: (i) the *3I-orientation*, where all three edges are

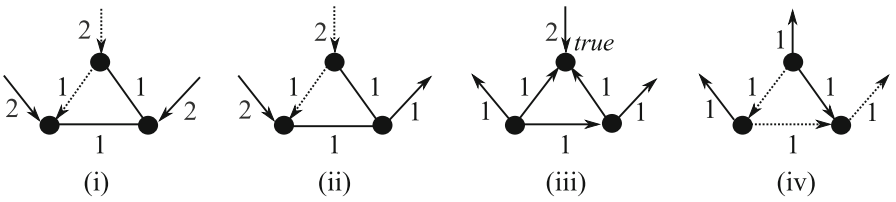

Fig. 6 Possible orientations of the edges in the variable-clause edge set incident to a clause gadget: (i) the 3I-orientation; (ii) a 2I1O-orientation; (iii) a 1I2O-orientation; and (iv) the 3O-orientation

oriented toward G_{C_j}; (ii) a *2I1O-orientation*, where two edges are oriented toward G_{C_j} and one edge away from G_{C_j}; (iii) a *1I2O-orientation*, where one edge is oriented toward G_{C_j} and two edges away from G_{C_j}; or: (iv) the *3O-orientation*, where all three edges are oriented away from G_{C_j}. (Here, "I" stands for "Inward" and "O" for "Outward".) See Fig. 6.

Lemma 5 *To achieve $h_s(\widetilde{G}) = 2$, it holds for each clause gadget that the three edges in the variable-clause edge set incident to it must be oriented according to a 1I2O-orientation.*

Proof Let G_{C_j} be any clause gadget. First, observe that if the variable-clause edges incident to G_{C_j} are oriented according to a 3I- or 2I1O-orientation then for some edge e in G_{C_j}, both of e's endpoints are incident to variable-clause edges oriented toward G_{C_j}. See Fig. 6 (i) and (ii). The directed path consisting of one of these two variable-clause edges and e will then have length $2 + 1 = 3$. Secondly, if the variable-clause edges incident to G_{C_j} are oriented according to a 3O-orientation then, since every orientation of G_{C_j} itself contains a directed path of length at least 2, there exists a directed path in $\widetilde{G}$ containing two edges from G_{C_j} followed by a variable-clause edge oriented away from G_{C_j} of total length at least $1 + 1 + 1 = 3$. See Fig. 6 (iv).

We conclude that a 1I2O-orientation as shown in Fig. 6 (iii) is the only possible way to obtain $h_s(\widetilde{G}) = 2$. □

We are now ready to prove the main result of this section.

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

Proof Let ϕ by any instance of PLANAR ONE-IN-THREE 3-SAT and let G be the undirected, edge-bi-weighted graph constructed from ϕ in the above polynomial-time reduction. By Lemma 3, $H_s(G) \geq 2$. We will show that if ϕ is satisfiable then $H_s(G) = 2$; otherwise, $H_s(G) \geq 3$.

(1) Suppose that ϕ is satisfiable. Let τ be a satisfying truth assignment to the variables in ϕ. We can orient G based on τ in such a way that every created directed path has length at most 2 as follows.

For each $1 \leq i \leq n$, if the variable x_i is true in τ then choose the FB-orientation for the variable gadget G_{x_i}; otherwise, take the BF-orientation. Next, define $t(C_j) \in \{1, 2, 3\}$ for each $C_j \in \mathcal{C}$ to be the index in C_j of the literal that made C_j true, e.g., if

$C_j = x_a \vee \overline{x_b} \vee \overline{x_c}$ and τ assigns $x_a =$ false, $x_b =$ false, and $x_c =$ true then $t(C_j) = 2$. For each $1 \leq j \leq m$, orient the clause gadget G_{C_j} so that two of its edges are directed toward the vertex $c_{j,t(C_j)}$ and its third edge is in the counterclockwise direction, and orient the variable-clause edges incident to G_{C_j} according to the 1I2O-orientation that makes $c_{j,t(C_j)}$ have outdegree 0. See Fig. 6 (iii).

As a result, for each variable x_i, if x_i is true then every variable-clause edge of the form $\{x_{i,\ldots}, c_{\ldots,\ldots}\}$ will be directed away from the vertex $x_{i,\ldots}$ in G_{x_i} whereas every variable-clause edge of the form $\{\overline{x_{i,\ldots}}, c_{\ldots,\ldots}\}$ will be directed toward the vertex $\overline{x_{i,\ldots}}$, which thus has outdegree 0 by the definition of the FB-orientation. This means that a directed path that starts in G_{x_i} can never include both an edge of G_{x_i} and a variable-clause edge at the same time; see Fig. 5 (Top). Also, because of the orientations chosen for the clause gadgets and the variable-clause edges, any directed path from G_{x_i} to a vertex in a clause gadget G_{C_j} ends at that vertex. This shows that every directed path starting somewhere in G_{x_i} has length at most 2. On the other hand, if x_i is false then all variable-clause edges of the form $\{\overline{x_{i,\ldots}}, c_{\ldots,\ldots}\}$ will be directed away from G_{x_i} while all variable-clause edges of the form $\{x_{i,\ldots}, c_{\ldots,\ldots}\}$ will be directed toward G_{x_i}. The definition of the BF-orientation (see Fig. 5 (Bottom)) now guarantees that every directed path that starts in G_{x_i} has length at most 2 in the same way as above.

Finally, observe that a directed path that starts in a clause gadget G_{C_j} can consist of at most two edges from G_{C_j} (in the counterclockwise direction), or at most one edge from G_{C_j} followed by at most one variable-clause edge and zero edges from a variable gadget, and that the length of any such path is at most $1 + 1 = 2$.

This gives $H_s(G) = 2$.

(2) Suppose that $H_s(G) = 2$ and consider an optimal orientation $\widetilde{G}$ of G. According to the first part of Lemma 4, each variable gadget uses the FB-orientation or the BF-orientation. We construct a truth assignment to the variables as follows: For each $1 \leq i \leq n$, if G_{x_i} uses the FB-orientation then let $x_i =$ true, but if G_{x_i} uses the BF-orientation then let $x_i =$ false.

By the second part of Lemma 4, if a variable x_i was set to true then all variable-clause edges that represent occurrences of the literal x_i in clauses are directed away from G_{x_i} in $\widetilde{G}$ and all variable-clause edges that represent occurrences of $\overline{x_i}$ are directed toward G_{x_i}. Similarly, if x_i was set to false then variable-clause edges representing $\overline{x_i}$ are directed away from G_{x_i} and variable-clause edges representing x_i are directed toward G_{x_i}. Now, Lemma 5 says that the three variable-clause edges incident to each clause gadget G_{C_j} must be oriented according to a 1I2O-orientation. It follows from the definition of a 1I2O-orientation that one literal in C_j is true and the other two literals are false. Hence, ϕ is satisfiable by a truth assignment that makes exactly one literal true in every clause. □

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* $\{1, 2\}$.

The reduction used to prove Theorem 1 also 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 where all edge weights belong to $\{1, 2\}$) then P = NP.*

Proof Suppose that a polynomial-time $(3/2 - \varepsilon)$-approximation algorithm ALG for SLPO$_s$ or SLPO$_m$ existed. Then PLANAR ONE-IN-THREE 3-SAT could be solved in polynomial time by constructing the undirected subcubic planar graph G for any given instance ϕ of PLANAR ONE-IN-THREE 3-SAT as described above, running ALG on G, and outputting "yes" if the obtained orientation's cost was < 3 and "no" otherwise. (This works because according to the proof of Theorem 1, if ϕ is satisfiable then $H_s(G) = 2$, so running ALG on G produces an orientation of cost at most $(3/2 - \varepsilon) \cdot 2 < 3$; on the other hand, if ϕ is not satisfiable then $H_s(G) \geq 3$, so the orientation output by ALG has cost ≥ 3.) Since PLANAR ONE-IN-THREE 3-SAT is NP-hard, this would imply P = NP. □

4 Algorithms for path graphs

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 direct 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.

The following notation is used:

- $L_{i,j}$ 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 toward larger numbered vertices (i.e., of the form $(i, i+1)$) and toward 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 cost of an optimal orientation of $L_{1,i}$ under H_x assuming that edge $\{i-1, i\}$ is directed toward i. Analogously, $H_x^{\leftarrow}(i)$ is the cost of an optimal orientation of $L_{1,i}$ under H_x assuming that edge $\{i-1, i\}$ is directed toward $i-1$. In particular, the cost of an optimal orientation of L under H_x is given by $\min\{H_x^{\rightarrow}(n), H_x^{\leftarrow}(n)\}$.

The algorithm computes $H_x^{\rightarrow}(j)$ and $H_x^{\leftarrow}(j)$ from $j = 2$ up to $j = n$ using dynamic programming. To obtain any such $H_x^{\rightarrow}(j)$ or $H_x^{\leftarrow}(j)$, the idea is to locate,

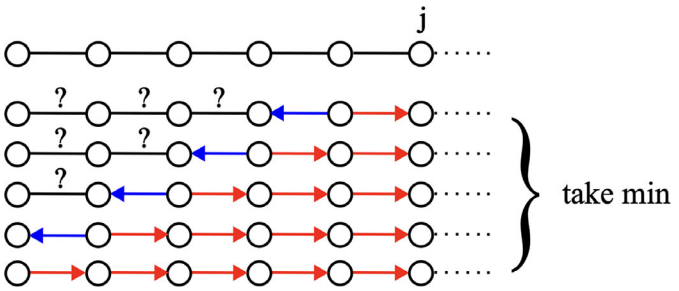

Fig. 7 Illustrating the dynamic programming solution. To compute $H_x^{\rightarrow}(j)$, the algorithm considers, for each i with $1 \leq i < j$, an orientation consisting of a single directed path between i and j (shown in red) together with an optimal orientation of $L_{1,i}$ in which the final edge (shown in blue) is oriented in the opposite direction of this path, and takes the best one found. Edges labeled by question marks have no restrictions on their directions when computing $H_x^{\rightarrow}(j)$. The computation of $H_x^{\leftarrow}(j)$ is analogous, with the directions of the red and blue edges in the figure reversed

in an optimal orientation of $L_{1,j}$, the largest i with $i < j$ at which there is a change in direction, given that the last edge $\{j-1, j\}$ has a specified direction, i.e., $(j-1, j)$ in case of $H_x^{\rightarrow}(j)$ and $(j, j-1)$ in case of $H_x^{\leftarrow}(j)$. See Fig. 7.

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

Proof By the discussion above, the recurrences for $H_x^{\rightarrow}$ and $H_x^{\leftarrow}$ can be written as $H_x^{\rightarrow}(j) = \min_{1 \leq i < j} \max\{H_x^{\leftarrow}(i),\ h_x(\vec{L}_{i,j})\}$ and $H_x^{\leftarrow}(j) = \min_{1 \leq i < j} \max\{H_x^{\rightarrow}(i),\ h_x(\overleftarrow{L}_{i,j})\}$. □

Algorithm 1: BestOrientPath$_x(L)$

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

1 **if** $n = 1$ **then**
2 **return** 0;
3 $H_x^{\rightarrow}(1) = H_x^{\leftarrow}(1) = -\infty$;
4 **for** $j = 2$ **to** n **do**
5 $H_x^{\rightarrow}(j) = \min_{1 \leq i < j} \max\{H_x^{\leftarrow}(i),\ h_x(\vec{L}_{i,j})\}$;
6 $H_x^{\leftarrow}(j) = \min_{1 \leq i < j} \max\{H_x^{\rightarrow}(i),\ h_x(\overleftarrow{L}_{i,j})\}$;
7 **return** $\min\{H_x^{\rightarrow}(n),\ H_x^{\leftarrow}(n)\}$;

To analyze the running time under each of the cost functions, we need to specify how two issues are addressed:

1. The computation of $h_x(\vec{L}_{i,j})$ and $h_x(\overleftarrow{L}_{i,j})$, for given $1 \leq i < j \leq n$.
2. The cumulative computation time of all minimum values in steps 5 and 6 in Algorithm BestOrientPath$_x(L)$. A point to remember is that $H_x^{\rightarrow}(j)$ and $H_x^{\leftarrow}(j)$ can be computed only after the values of $H_x^{\leftarrow}(i)$ and $H_x^{\rightarrow}(i)$, respectively, are known for $2 \leq i < j$. We will call this an *interwoven* computation.

In the next two subsections we will see that these computations can be implemented much more efficiently for H_s than for H_m.

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. For this cost function observe that equations (1) and (2) give:

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

Computing $h_s(\vec{L}_{i,j})$, and similarly $h_s(\overleftarrow{L}_{i,j})$, for a pair (i, j) is therefore an instance of the Range Maximum-sum Segment Online Query problem, RMSOQ for short (Chen and Chao 2007):

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 $i \le i' \le j' \le j$.

Chen and Chao (2007) presented a method for answering each such query in constant time after the input has been preprocessed in $O(n)$ time. This gives:

Lemma 6 *Suppose $w(i, i+1), 1 \le i < n$ and $w(i+1, i), 1 \le i < n$ have been preprocessed in linear time for the RMSOQ problem. Then each value of the form $h_s(\vec{L}_{i,j})$ and $h_s(\overleftarrow{L}_{i,j})$ can be evaluated in constant time.*

To address the second computation issue (finding the minima in steps 5 and 6 efficiently), first define the two $(n \times n)$-matrices $M^{\rightarrow}$ and $M^{\leftarrow}$ by:

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

In these terms, steps 5 and 6 of the algorithm compute the minimum value in column j of each of the two matrices $M^{\rightarrow}$ and $M^{\leftarrow}$. However, $M^{\rightarrow}[i, j]$ and $M^{\leftarrow}[i, j]$ are not given explicitly, and in fact are computed on the basis of the minima in earlier columns. They are therefore *implicitly defined* but computed online in constant time, using the method of Lemma 6.

Lemma 7 *Suppose $w(i, i+1), 1 \le i < n$ and $w(i+1, i), 1 \le i < n$ have been preprocessed for the RMSOQ problem. After $H_s^{\leftarrow}(i)$ and $H_s^{\rightarrow}(i)$ have been computed for $1 \le i < j$, each of the entries $M^{\rightarrow}[i, j]$ and $M^{\leftarrow}[i, j]$ can be evaluated in constant time.*

We show next that both of these matrices are *totally b-monotone*, where a matrix M is totally b-monotone if for all $i_1 < i_2$ and $j_1 < j_2$:

$$M[i_1, j_1] \geq M[i_2, j_1] \implies M[i_1, j_2] \geq M[i_2, j_2].$$

For the proof of this property observe first of all 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 according to equation (6).

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

Proof Suppose there are $i_1 < i_2$ and $j_1 < j_2$ such that $M[i_1, j_1] \geq M[i_2, j_1]$ but $M[i_1, j_2] < M[i_2, j_2]$. Spelling out the latter inequality, $\max\{f(i_1), g(i_1, j_2)\} < \max\{f(i_2), g(i_2, j_2)\}$, and combining it with $g(i_1, j_2) \geq \max\{g(i_1, j_1), g(i_2, j_2)\}$, leads to the conclusion that

$$f(i_2) > \max\{f(i_1), g(i_1, j_2)\} \geq \max\{f(i_1), g(i_1, j_1)\} = M[i_1, j_1],$$

a contradiction to $M[i_1, j_1] \geq M[i_2, j_1]$. □

The second computation issue can now rephrased as follows.

Problem 2 *(Online Interwoven Computation of Column Minima of Two Totally b-Monotone Matrices)*

For $1 \leq j \leq n$, compute $H^{\rightarrow}(j) = \min\{M^{\leftarrow}[i, j] \mid 1 \leq i \leq n\}$ and $H^{\leftarrow}(j) = \min\{M^{\rightarrow}[i, j] \mid 1 \leq i \leq n\}$, where $M^{\leftarrow}$ and $M^{\rightarrow}$ are implicitly defined totally b-monotone and the values of $H^{\leftarrow}(i)$ and $H^{\rightarrow}(i)$ with $i < j$ have to be computed before $M^{\leftarrow}[i, j]$ and $M^{\rightarrow}[i, j]$ can be evaluated.

Proposition 1 has two corollaries that are key to the computation of the column minima.

1. Denote the row index of the bottom-most minimum in column j of M by $BM[j]$, i.e., $BM[j] = \max\{i \mid M[i, j] = \min_{\ell} M[\ell, j]\}$. If M is totally b-monotone then

   $$BM[j] \leq BM[j + 1] \text{ for all } j.$$

   Thus, for example, if we happen to know the values of $BM[j]$ and $BM[j + 2]$ then we can find $BM[j + 1]$ simply by examining the entries $M[i, j + 1]$ for $BM[j] \leq i \leq BM[j + 2]$.

2. Suppose we compare two entries $M[i_1, j]$ and $M[i_2, j]$, $i_1 < i_2$, in column j.
   - If $M[i_1, j] \geq M[i_2, j]$ then $BM[j'] \neq i_1$ for all $j' \geq j$, so that the values $M[i_1, j']$ are no longer of interest.
   - If $M[i_1, j] < M[i_2, j]$ then $BM[j'] \neq i_2$ for all $j' \leq j$.

To solve Problem 2 we take advantage of the fact that totally monotone matrices, which have the property that for all $i_1 < i_2$ and $j_1 < j_2$,

$$M[i_1, j_1] > M[i_2, j_1] \implies M[i_1, j_2] > M[i_2, j_2],$$

are well-studied. The two properties of totally monotone matrices that are key to the algorithms of interest here are analogous to the previously stated properties of b-monotone matrices:

1. Denote the row index of the topmost minimum in column j of a totally monotone matrix M by $TM[j]$, i.e., $TM[j] = \min\{i \mid M[i, j] = \min_\ell M[\ell, j]\}$. Then

$$TM[j] \leq TM[j+1] \text{ for all } j.$$

2. Suppose we compare two entries $M[i_1, j]$ and $M[i_2, j]$, $i_1 < i_2$, in column j.
   - If $M[i_1, j] > M[i_2, j]$ then $TM[j'] \neq i_1$ for all $j' \geq j$.
   - If $M[i_1, j] \leq M[i_2, j]$ then $TM[j'] \neq i_2$ for all $j' \leq j$.

An especially useful algorithm, the SMAWK algorithm (Aggarwal et al. 1987), computes in *linear* time all column minima of an implicitly defined totally monotone matrix M, provided each $M[i, j]$ can be computed in constant time. The only properties of the matrix that the algorithm uses are the ones just stated. Using instead of these properties the analogous ones for totally b-monotone matrices converts the algorithm into one, the b-SMAWK algorithm, that finds all column minima of a totally b-monotone matrix.

Note that the b-SMAWK algorithm does not solve Problem 2 because it requires constant time access to *all* $M[i, j]$. To the rescue comes the algorithm of Galil and Park (1990) which finds the column minima of a totally monotone matrix with the restriction that $M[i, j]$ can be computed only after the column minima for columns $1, \ldots, i$ have been found. Their algorithm, like the SMAWK algorithm, uses only the two properties of M stated above. To adapt it for use here all that needs to be done is again substituting the b-monotone properties for the two monotone ones mentioned above, and to replace each call to the SMAWK algorithm by a call to the b-SMAWK algorithm. In conclusion, to solve Problem 2 we adapt the algorithm of Galil and Park by substituting the totally b-monotone properties for the totally monotone ones, using b-SMAWK instead of SMAWK, and computing $H^{\rightarrow}(j)$ and $H^{\leftarrow}(j)$ together in iteration j.

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

4.3 Running time under cost function H_m

Here we show how to make BestOrientPath$_m$ run in $O(n \log n)$ time. Combining (1) and (3) yields $h_m(\vec{L}_{i,j}) = \vec{W}_j - \vec{W}_i = \sum_{t=i}^{j-1} w(t, t+1)$, where $\vec{W}_1 = 0$ and

$\vec{W}_j = \sum_{t=1}^{j-1} w(t, t+1)$ for $j \geq 2$. Substituting these equalities into step 5 results in

$$H_m^{\rightarrow}(j) = \min_{1 \leq i < j} \max\{H_m^{\leftarrow}(i), \vec{W}_j - \vec{W}_i\}, 2 \leq j \leq n.$$

Similar equalities hold for $h_m(\overleftarrow{L}_{i,j})$ and $H_m^{\leftarrow}(j)$, in fact so similar that we will go into the computation details only for $H_m^{\rightarrow}(j)$.

The task at hand can be rephrased as the computation of $H_m^{\rightarrow}(j) = \min\{M_1(j), M_2(j)\}$ for $j \geq 2$, with

$$M_1(j) = \min_{1 \leq i < j} \{H_m^{\leftarrow}(i) \mid H_m^{\leftarrow}(i) + W_i \geq W_j\}, \tag{7}$$

$$M_2(j) = \min_{1 \leq i < j} \{W_j - W_i \mid H_m^{\leftarrow}(i) + W_i < W_j\}, \tag{8}$$

where $M_1(j)$ or $M_2(j)$ are defined to be ∞ if the minimum is taken over an empty set. We focus on how to efficiently compute the minima in equation (7), since equation (8) can be handled similarly.

We abstract the problem, replacing $H_m^{\leftarrow}(i)$ by $value_i$ and $H_m^{\leftarrow}(i) + W_i$ by key_i, and asking for a data structure that supports the operations in greater generality than required for the computation of equation (7).

Problem 3 *(Implementing the operations Insert, and FindMin among lower-bounded keys)*

Design: A data structure containing items $(key_i, value_i)$ that efficiently supports two arbitrarily interspersed operations:

- Insert(key, $value$), which inserts that item.
- FindMin(W), that finds, among those items which were inserted into the data structure and have a key not less than W, an item whose value is least; if no such item exists, it returns ∞.

To address Problem 3, we maintain a set of items S, denoting by S_k its state after k items have been inserted. Thinking of the items as points in the $(Key, Value)$ plane, S_k is a sorted set consisting only of those items that are the breakpoints on the step-function that is the lower envelope of the first k items, as illustrated in Fig. 8 by the red points on the step function, while the scattered blue points are $(key, value)$ pairs that were discarded. For convenience, we also add a zeroth item $(-\infty, -\infty)$ and a last item (∞, ∞). Thus, $S_k = \langle (K_0, V_0), \ldots, (K_{\ell_k+1}, V_{\ell_k+1}) \rangle$, and the operations on it are defined as follows:

- FindMin(W):
  find the smallest i such that $W \leq K_i$ and **return** V_i.
- Insert(key, $value$):
  1. find the smallest i such that $key \leq K_i$;
  2. **if** $value \geq V_i$ **then** discard $(key, value)$ {comment: $S_{k+1} = S_k$};
  3. **else**

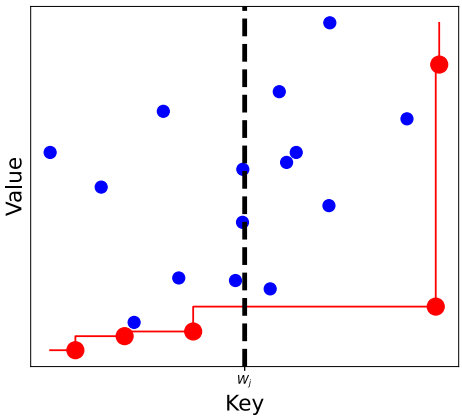

Fig. 8 Computing $FindMin(W_j)$

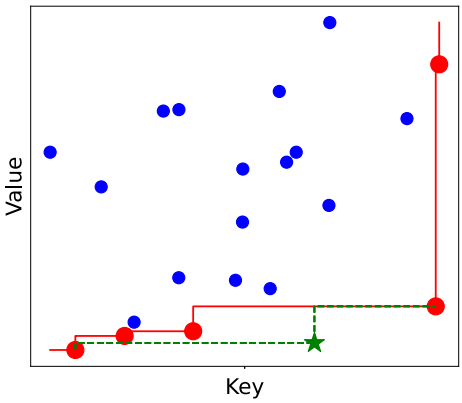

Fig. 9 Inserting the star item into S

(a) **if** $key = K_i$ **then** set V_i to $value$ **else** insert $(key, value)$ between items $i-1$ and i;
(b) set j to $i-1$;
(c) **while** $V_j \geq value$ **do** delete (K_j, V_j) from S_k and set j to $j-1$;

Fig. 8 illustrates FindMin where the points to the right of the broken vertical line are the ones whose minimum value is to be found. Fig. 9 illustrates Insert, where the star point is the item that is to be inserted into S, and the points on the step function that are above the horizontal line to the left of the star point are discarded.

Lemma 8 *The Insert procedure preserves the following invariants of* S_k.

1. $K_{h-1} < K_h$ *and* $V_{h-1} < V_h$, *for* $1 \leq h$.
2. *If* $(key, value)$ *is one of the* k *items that were inserted, and* $K_{h-1} < key \leq K_h$ *then* $V_h \leq value$.

Taken together the two invariants imply the correctness of the $FindMin(W)$ *computation: if* $K_{i-1} < W \leq K_i$ *then the value returned by* $FindMin(W)$, V_i, *is correct.*

Proof By induction on k. When $k = 1$, both invariants are clearly true.

For general k, if, in step 2 of $Insert(key, value)$ applied to S_k, the item $(key, value)$ discarded then both invariants continue to hold for all items because they did for S_k and because the second invariant clearly holds for the discarded item.

If the item $(key, value)$ is not discarded, i.e., $value < V_i$, then in step 3(a) $(key, value)$ either replaces the i-th item (when $key = K_i$), or is inserted immediately before it. In either case the first invariant holds from thereon. The execution of the while-loop 3(c) then removes all items that violate the first invariant. Moreover, all items (K, V) that were so deleted satisfy the second invariant because $K_j < K \leq key$ and $V \geq value$, while items (K, V) with $K_j < K \leq K_i$ that were deleted previously satisfied $V \geq V_{j+1}$ and $V_{j+1} \geq value$. □

Theorem 5 *The Sorted Set data structure supports each of the operations of Problem 3 in amortized time $O(\log n)$. Consequently, BestOrientPath$_x$ with cost function H_m runs in $O(n \log n)$ time.*

Proof The sorted set data structure can be implemented using red-black trees (Guibas and Sedgewick 1978). Each individual operation (find, insert, delete) used in our implementation of FindMin and Insert takes $O(\log n)$ time. Only the while-loop 3(c) of Insert may use more than one delete operation, but each such operation removes the item forever, so that the total number of delete operations does not exceed n and the amortized cost per Insert is $O(1)$.

Noting that the computation steps of the minima in equation (7) alternate n times between Insert and FindMin, the resulting overall running time is $O(n \log n)$. □

5 Algorithms for cycle graphs

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. (Here we begin the vertex numbering at 0 instead of 1 as in the previous section for notational convenience.) 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 path graph L by unrolling the input cycle graph C three times. Next it finds an optimal orientation $\widetilde{L}^*$ of L using BestOrientPath$_s$ from Sect. 4.2. Finally, it fashions from $\widetilde{L}^*$ an optimal orientation of C. Unless the optimal orientation is a one-way one, which is easily checked, it is constructed by copying the directions of the edges in a contiguous segment of $\widetilde{L}^*$ to C, possibly reversing the direction of a single edge. The pseudocode is given in Algorithm 2.

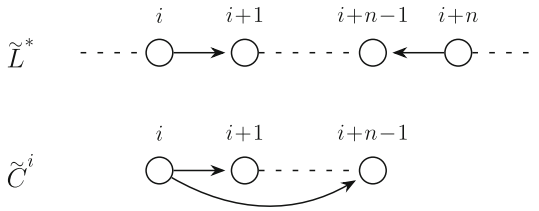

Fig. 10 The construction of $\widetilde{C}^i$ in statement 4 of BestOrientCycle$_s(C)$

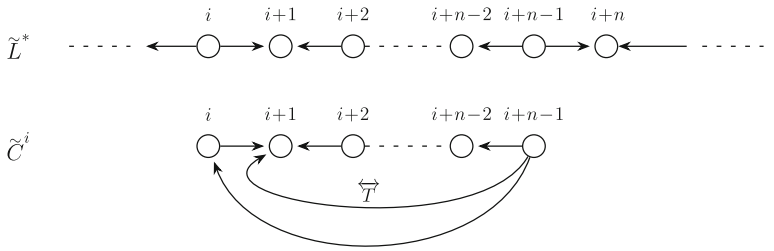

Fig. 11 The construction of $\widetilde{C}^i$ in statement 5 of BestOrientCycle$_s(C)$

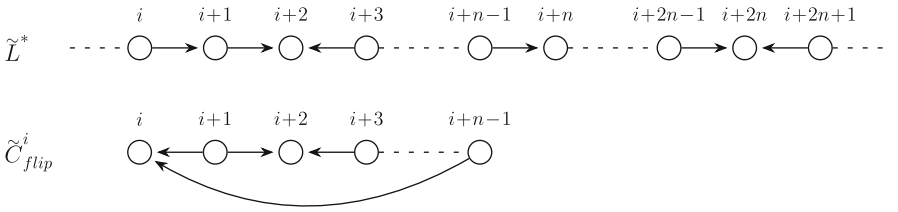

Fig. 12 The construction of $\widetilde{C}^i_{flip}$ in statement 6 of BestOrientCycle$_s(C)$

Definition 2 The types of orientations used in Algorithm BestOrientCycle$_s(C)$ are the following.

- $\widetilde{C}_{1way}$ is that one-way orientation of C whose cost, $1WayCost_s$, is the least.
- $\widetilde{C}^i$ is the orientation of C obtained by copying from $\widetilde{L}^*$ the directions of the edges $\{k, k+1\},\ i \le k < i+n$. This orientation is illustrated in Figs. 10 and 11.
- $\widetilde{C}^i_{flip}$ is the orientation of C obtained by flipping the first edge of $\widetilde{C}^i$. This orientation is illustrated in Fig. 12.

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 with $3n+1$ vertices 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 in C;
2 let $\widetilde{L}^*$ be the oriented graph returned by BestOrientPath$_s(L)$;
3 **if** $h_s(\widetilde{L}^*) \geq 1WayCost_s$ **then** set $\widetilde{C}$ to $\widetilde{C}_{1way}$;
4 **else if** *two edges* $\{i, i+1\}$ *and* $\{i+n-1, i+n\}$ *have opposite directions in* $\widetilde{L}^*$ **then** set $\widetilde{C}$ to $\widetilde{C}^i$;
5 **else if** *n is odd and every two consecutive edges in* $\widetilde{L}^*$ *have opposite directions* **then** let $\vec{T}$ be a directed path in C with two edges whose weight $h_s(\vec{T})$ is least among all such paths, and let j be its middle vertex;
if the edge $\{j, j+1\}$ has the same direction in $\vec{T}$ and $\widetilde{L}^*$ then let $i = j$ else let $i = j+n$, and set $\widetilde{C}$ to $\widetilde{C}^i$;
6 **else** find an i, $0 \leq i < n$, such that $(i, i+1)$, $(i+1, i+2)$, $(i+3, i+2)$ appear in $\widetilde{L}^*$ or an i, $2n \leq i < 3n$, such that $(i-2, i-1)$, $(i, i-1)$, $(i+1, i)$ appear in $\widetilde{L}^*$;
set $\widetilde{C}$ to $\widetilde{C}^i_{flip}$;
7 **return** $\widetilde{C}$;

Turning now to prove the correctness of Algorithm 2, we do so in a series of lemmas corresponding to statements 3-6. But first a definition and a useful observation:

Definition 3 The orientation $\widetilde{L}$ of L, *induced* by an orientation $\widetilde{C}$ of C, is obtained by assigning each edge $\{i, i+1\}$ of L the direction that edge $\{i \bmod n, (i+1) \bmod n\}$ has in $\widetilde{C}$.

Lemma 9 *If* $\widetilde{C}$ *includes a direction change then the induced orientation satisfies* $h_s(\widetilde{C}) = h_s(\widetilde{L})$.

Proof Any orientation of the cycle graph consists of direction reversals at an even number of vertices (including possibly no reversals at all). In particular, if $\widetilde{C}$ includes a change in direction then its longest directed path has at most n edges. Each directed path of the induced orientation $\widetilde{L}$ is therefore a subpath of some directed path in $\widetilde{C}$ so that $h_s(\widetilde{L}) \leq h_s(\widetilde{C})$, by the monotonicity of h_s. Since L is longer than $2n$, any directed path of $\widetilde{C}$ appears in its entirety in $\widetilde{L}$, whence also $h_s(\widetilde{C}) \leq h_s(\widetilde{L})$. □

Remark 1 If $\widetilde{C}$ is a one-way orientation then $h_s(\widetilde{L}) > h_s(\widetilde{C})$ is possible. For example, if $n = 3$ and $w(i, i+1) = 1$ for $0 \leq i \leq 2$, and $\widetilde{C}$ is the orientation (0, 1), (1, 2), (2, 0), then $h_s(\widetilde{L}) = 9$ and $h_s(\widetilde{C}) = 3$.

Lemma 10 *If* $H_s(L) \geq 1WayCost_s$ *then* $1WayCost_s$ *is the cost of an optimal orientation of* C.

Proof Suppose to the contrary that there exists a $\tilde{C}^*$ such that $h_s(\widetilde{C}^*) = H_s(C) < 1WayCost_s$. Then there is a direction change in $\widetilde{C}^*$, and the induced orientation $\widetilde{L}$ satisfies $H_s(L) \le h_s(\widetilde{L}) = h_s(\widetilde{C}^*) < 1WayCost_s$ according to Lemma 9, which is a contradiction. □

Remark 2 The orientation $\widetilde{L}$ induced by $\widetilde{C}^*$ may not be optimal for L, even if $\widetilde{C}^*$ includes a change in direction. Here is an example: $n = 3$ and $w(i, i+1) = w(i+1, i) = 2$ for $i = 0, 1$, and $w(2, 0) = w(0, 2) = 3$. Then $h_s(\widetilde{C}^*) = 4 > h_s(\widetilde{L}^*) = 3$. It follows from Corollary 2 at the end of this subsection that this can happen only when n is odd and the optimal orientation of L reverses direction at each vertex.

Lemma 11 *Suppose $h_s(\widetilde{L}^*) = H_s(L) < 1WayCost_s$. If there is an i, $0 \le i \le n$, such that the edges $\{i, i+1\}$ and $\{i+n-1, i+n\}$ have opposite directions in $\widetilde{L}^*$, then its subgraph $\widetilde{L}^*_{i,i+n}$ induces an orientation $\widetilde{C}^i$ of the cycle that is optimal, and $H_s(C) = h_s(\widetilde{C}^i) = H_s(L)$.*

Proof (See also Fig. 10.) There is a change of direction at vertex i in $\widetilde{C}^i$, because the edges $\{i, i+1\}$ and $\{i+n-1, i+n\}$ have opposite directions. Therefore $h_s(\widetilde{C}^i) = h_s(\widetilde{L}_{i,i+n}) \le h_s(\widetilde{L}^*)$.

There is also a change of direction in any optimal $\widetilde{C}^*$, because $h_s(\widetilde{C}^*) \le h_s(\widetilde{C}^i) \le h_s(\widetilde{L}^*) < 1WayCost_s$. By Lemma 9, the orientation $\widetilde{L}'$ induced by $\widetilde{C}^*$ satisfies $h_s(\widetilde{L}') = h_s(\widetilde{C}^*)$. Hence

$$h_s(\widetilde{L}^*) \le h_s(\widetilde{L}') = h_s(\widetilde{C}^*) \le h_s(\widetilde{L}^*).$$

Thus $H_s(C) = h_s(\widetilde{C}^*) = h_s(\widetilde{C}^i) = h_s(\widetilde{L}^*) = H_s(L)$. □

Remark 3 Note that if n is even and every two consecutive edges in $\widetilde{L}^*$ have opposite directions then all edges $\{i, i+1\}$ and $\{i+n-1, i+n\}$ have opposite directions.

Lemma 12 *Suppose that $n > 1$ is odd, $h_s(\widetilde{L}^*) = H_s(L) < 1WayCost_s$, and every two consecutive edges in $\widetilde{L}^*$ have opposite directions. Let $\vec{T}$ be a directed path in C with two edges whose weight $h_s(\vec{T})$ is least among all such paths, and let j be its middle vertex. If the edge $\{j, j+1\}$ has the same direction in $\vec{T}$ and $\widetilde{L}^*$ then let $i = j$ else let $i = j + n$. Then the cycle orientation $\widetilde{C} = \widetilde{C}^i$ is optimal, and*

$$H_s(C) \le \max\{h_s(\widetilde{L}^*), h_s(\vec{T})\}.$$

Proof (See also Fig. 11.) Since L contains $3n$ edges, n is odd, and $\widetilde{L}^*$ reverses direction at each vertex, any edge $(j, j+1)$ (edge $(j+1, j)$, respectively), where $0 \le j \le 2n-1$,

appears in $\widetilde{L}^*$ if and only if the edge $(j+n+1, j+n)$ ($(j+n, j+n+1)$, respectively) does so. Therefore

$$h_s(\widetilde{L}^*) = \max_{0 \le i \le n-1} \{w(i, i+1), w(i+1, i)\}.$$

For ease of notation, assume that $\vec{T}$ is the directed path from 0 to 2, so that $i = 1$. Then $\widetilde{C}=\widetilde{C}^1$ consists of the edges $(1, 2), (3, 2), (3, 4), \ldots, (0, n-1), (0, 1)$, i.e., it is a chain of alternating direction edges beginning and ending at $\vec{T}$. Consequently, $H_s(C) \le h_s(\widetilde{C}) \le \max\{h_s(\widetilde{L}^*), h_s(\vec{T})\}$.

Let $\widetilde{C}^*$ be an optimal orientation of the cycle graph. If there is no change of direction in $\widetilde{C}^*$ then $h_s(\widetilde{C}^*) \ge h_s(\vec{T})$. Moreover, $1WayCost_s = h_s(\widetilde{C}^*) > h_s(\widetilde{L}^*)$. It follows that $h_s(\widetilde{C}^*) \ge h_s(\widetilde{C})$, proving that $\widetilde{C}$ is optimal.

If $\widetilde{C}^*$ does contain a change of direction then Lemma 9 ensures that the orientation $\widetilde{L}'$ of L induced by $\widetilde{C}^*$ satisfies $h_s(\widetilde{C}^*) = h_s(\widetilde{L}') \ge h_s(\widetilde{L}^*)$. Furthermore, since the cycle is odd, $\widetilde{C}^*$ contains a directed path consisting of two edges that have the same direction so that $h_s(\widetilde{C}^*) \ge h_s(\vec{T})$. It follows again that $h_s(\widetilde{C}^*) \ge h_s(\widetilde{C})$, □

Finally we consider the orientations constructed when the conditions of steps 3, 4, and 5 do not hold. What we know then about $\widetilde{L}^*$ is:

1. $h_s(\widetilde{L}^*) < 1\text{WayCost}_s$, implying that the number of edges on each directed path in $\widetilde{L}^*$ is less than n;
2. for $0 \le i \le 2n$, the edges $\{i, i+1\}$ and $\{i+n-1, i+n\}$ have the same direction;
3. not all successive edges of $\widetilde{L}^*$ alternate in direction, and in particular there is a directed path of two or more edges.

Lemma 13 *When the conditions of steps 3, 4, and 5 do not hold then $\widetilde{L}^*$ contains at least one of the following:*

- *a triplet of edges $(i, i+1), (i+1, i+2), (i+3, i+2)$ with $0 \le i \le n-1$;*
- *a triplet of edges $(i-2, i-1), (i, i-1), (i+1, i)$ with $2n \le i \le 3n-1$.*

In both cases $\widetilde{C}^i_{flip}$ is an optimal orientation of C and $h_s(\widetilde{L}^*) = h_s(\widetilde{C}^i_{flip}) = H_s(C)$.

Proof That $\widetilde{L}^*$ contains at least one of the triplets for some i follows directly from item 3 above. Suppose it contains $(i, i+1), (i+1, i+2), (i+3, i+2)$ with $n \le i$. Then by item 2 above it also contains the triplet of edges $(j, j+1), (j+1, j+2), (j+3, j+2)$

1. where $1 \le j = i - n + 1 \le n-1$, if $n \le i \le 2n-2$,
2. where $1 \le j = i - 2n + 2 < n$, if $2n-1 \le i \le 3n-3$.

This proves that $\widetilde{L}^*$ contains a triplet of edges $(i, i+1), (i+1, i+2), (i+3, i+2)$ with $0 \le i \le n-1$.

Similarly, if $\widetilde{L}^*$ contains some triplet $(i-2, i-1), (i, i-1), (i+1, i)$ then it also contains such a triplet with $2n \le i \le 3n-1$.

To prove the optimality of $\widetilde{C}^i_{flip}$ in case $0 \le i \le n-1$, assume for simplicity that $i = 0$, i.e., the triplet is $(0, 1), (1, 2), (3, 2)$. Note that the fact that $(0, 1)$ belongs to $\widetilde{L}^*$ implies, according to item 2 above, that so does $(n-1, n) = (n-1, 0)$ so that $\widetilde{C}^i_{flip}$ contains the edges $(1, 0), (1, 2), (3, 2)$ and $(n-1, 0)$, among others. In addition, the fact that $(3, 2)$ belongs to $\widetilde{L}^*$ implies that so does $(2n+1, 2n)$. See Fig. 12.

We show next that any directed path in $\widetilde{C}^i_{flip}$ appears in $\widetilde{L}^*$. Consider first a directed path $\vec{P}$ that contains the edge $(1, 0)$. Since this edge is surrounded by the edges $(1, 2), (n-1, 0)$, it is the only one in $\vec{P}$. Moreover, edge $(1, 0)$ is identical to edge $(2n+1, 2n)$, which we saw belongs to $\widetilde{L}^*$, proving that $\vec{P}$ is a path in $\widetilde{L}^*$. Any other directed path in $\widetilde{C}^i_{flip}$ does not contain the edge $(1, 0)$ and is therefore also a directed path in $\widetilde{L}^*$. Consequently, $h_s(\widetilde{C}^i_{flip}) \le h_s(\widetilde{L}^*) < 1WayCost_s$.

Let $\widetilde{C}^*$ be an optimal orientation of the cycle graph. It contains a direction change because $h_s(\widetilde{C}^*) \le h_s(\widetilde{C}^i_{flip}) < 1WayCost_s$. Hence the orientation $\widetilde{L}'$ of L it induces satisfies $h_s(\widetilde{C}^*) = h_s(\widetilde{L}') \ge h_s(\widetilde{L}^*) \ge h_s(\widetilde{C}^i_{flip})$.

The proof for the other case is analogous. □

This proves the correctness of Algorithm BestOrientCycle$_s(C)$. 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. In summary:

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

Another conclusion from Lemmas 11, 12, and 13 is:

Corollary 2 *If $H_s(L) < 1WayCost_s$ then $H_s(L)$ is also the cost of an optimal orientation of the cycle graph, unless n is odd and $\widetilde{L}^*$ reverses direction at each vertex.*

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 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, $1WayCost_m$, is the least. Also, $L_{i+1,i-1}$ for $0 \le i \le n$ is the subgraph of C induced by vertices $i+1, i+2, \ldots, i-1$, with sums taken mod n.

Algorithm 3: BestOrientCycle$_m(C)$

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

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

Theorem 7 *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. □

6 Algorithms for star graphs

For any $n \geq 3$, an *n-star graph* is a tree with one internal vertex c and $n-1$ leaves. In this section, the internal vertex 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.

Lemma 14 *There is an optimal orientation of the star graph in which at least one of the edges (u_1, c) or (c, v_1) participates. Consequently, there is an optimal orientation whose edge set $\tilde{E}^*$ has one of the following forms:*

1. *$\tilde{E}^*$ contains all of $\widetilde{E_{in}}$ as well as the edges (v_i, c), $1 \le i \le j-1$ and the edges (c, v_i), $j \le i \le r$, for some $1 \le j \le r+1$.*
2. *$\tilde{E}^*$ contains all of $\widetilde{E_{out}}$ as well as the edges (c, u_i), $1 \le i \le j-1$ and the edges (u_i, c), $j \le i \le \ell$, for some $1 \le j \le \ell+1$.*

Proof Note that the cost of the initial orientation is $h_m(\widetilde{S}) = w(u_1, c) + w(c, v_1)$. Suppose that both (c, u_1) and (v_1, c) participate in an optimal orientation, $\widetilde{S}^*$. Then $H_m(S) = h_m(\widetilde{S}^*) \ge w(c, u_1) + w(v_1, c) \ge w(u_1, c) + w(c, v_1) \ge h_m(\widetilde{S})$, proving that $\widetilde{S}$ is an optimal orientation as well.

Let $\tilde{E}^*$ be the edge set of $\widetilde{S}^*$. Suppose $(u_1, c) \in \tilde{E}^*$, and let j be the smallest index such that $(c, v_j) \in \tilde{E}^*$. Then $h_m(\widetilde{S}^*) \ge w(u_1, c) + w(c, v_j)$. Thus, if (c, u_i), $i > 1$, happens to be in $\tilde{E}^*$ then the cost of the orientation will not be increased by flipping this edge to (u_i, c) because $w(u_i, c) \le w(u_1, c)$. Similarly, if (v_i, c), $i > j$, happens to be in $\tilde{E}^*$ then the cost of the orientation will not be increased by flipping the edge to (c, v_i) because $w(c, v_i) \le w(c, v_j)$. In conclusion, if $(u_1, c) \in \tilde{E}^*$ then there is an optimal orientation of the form 1.

A similar argument shows that there is an optimal orientation of the form 2 if $(v_1, c) \in \tilde{E}^*$. □

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 $\{u, c\}$ inward to c if $w(u, c) < w(c, u)$, and outward from c otherwise;
denote by $\widetilde{S}$ the resulting directed graph,
by $\widetilde{E_{in}} = \{(u_1, c), \ldots, (u_\ell, c)\}$ the list of its inward edges,
and by $\widetilde{E_{out}} = \{(c, v_1), \ldots, (c, v_r)\}$ the list of its outward edges;
reorder each of $\widetilde{E_{in}}$ and $\widetilde{E_{out}}$ so that the weights of its edges are in non-increasing order;
set $\widetilde{S_{right}}$ and $\widetilde{S'}$ to $\widetilde{S}$ and set $BestCost_{right}$ to $h_m(\widetilde{S})$;
for $k = 1$ **to** ℓ **do**
    flip the direction of edge (u_k, c) in $\widetilde{S'}$;
    if $h_m(\widetilde{S'}) < BestCost_{right}$ **then** set $\widetilde{S_{right}}$ to S' and set $BestCost_{right}$ to $h_m(\widetilde{S'})$
set $\widetilde{S_{left}}$ and $\widetilde{S'}$ to $\widetilde{S}$ and set $BestCost_{left}$ to $h_m(\widetilde{S})$;
for $k = 1$ **to** r **do**
    flip the direction of edge (c, v_k) in $\widetilde{S'}$;
    if $h_m(\widetilde{S'}) < BestCost_{left}$ **then** set $\widetilde{S_{left}}$ to S' and set $BestCost_{left}$ to $h_m(\widetilde{S'})$
if $BestCost_{left} < BestCost_{right}$ **then**
    return $\widetilde{S_{left}}$
else
    return $\widetilde{S_{right}}$

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

Proof Let $\widetilde{S}^*$ be an optimal orientation of the input n-star graph S. If the edge (u_1, c) participates in the orientation, i.e., there is an optimal orientation that has form 1, then loop 7 will find it. Similarly, if (c, v_1) participates, i.e., there is an optimal orientation of the form 2, then that orientation will be found by the loop 11.

The running time analysis is straightforward; the reordering of step 5 takes $O(n \log n)$ time and each update of $BestCost$ on lines 9 and 13 takes constant time if auxiliary variables are maintained for the values of $\max\{w(c, u_i) : 1 \le i < j\}$ and $\max\{w(v_i, c) : 1 \le i < j\}$. □

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$_s$).

Lemma 15 *Suppose S has a vertex u with an edge to or from c with nonpositive 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 nonpositive weight.*

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

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 nonpositive 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 nonpositive weight and add them to $\widetilde{S}'$, and denote the resulting directed star graph $\widetilde{S}$;
4 **return** $\widetilde{S}$;

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

Proof Lines 1 and 3 take $O(n)$ time each. By Theorem 8, line 2 takes $O(n \log n)$ time. □

7 Concluding remarks

We have introduced a new graph orientation problem called SLPO that generalizes graph coloring and analyzed its time complexity for various graph classes. Regarding

polynomial-time approximation ratios, we remark here that our problem formulation allows optimal costs to be nonpositive; for example, this happens in the extreme case where all edge weights in the input graph are negative. In such cases, the (standard) relative performance guarantee may not be appropriate, and an absolute performance guarantee could be used instead.

We hope that this article will inspire researchers to think about graph coloring problems in terms of graph orientations, find real-world applications of the theory, and continue to develop fast algorithms. Our findings suggest several promising topics for further investigation:

- How fast can the two variants SLPO_s and SLPO_m be solved for trees? As pointed out in Sect. 1.2, they can be solved in polynomial time, but the degree of the polynomial is large using that method directly. More generally, can dynamic programming give efficient algorithms for SLPO_s and SLPO_m on graphs of bounded treewidth?
- What is the time complexity of SLPO restricted to bipartite graphs? Suppose that $G = (U, V, E)$ is an edge-bi-weighted bipartite graph. The special case where all edge weights belong to $\{p, q\}$ for two integers p and q such that $0 < p \leq q \leq 2p$ is linear-time solvable as follows: Take the best of the two orientations obtained by orienting all edges from U to V or orienting all edges from V to U. Similarly, if all edge weights belong to $\{0, q\}$ for an integer $q > 0$, the problem can be solved by taking an orientation that uses weight-0 edges only if one exists, and if not, by orienting all edges from U to V. We conjecture that the problem becomes NP-hard for bipartite graphs even when all edge weights belong to $\{1, q\}$ for an integer $q \geq 3$ or all edge weights belong to $\{0, 1, 2\}$.
- Suppose that the edges of the input graph are allowed to have different weights, but for each edge, its two weights have to be the same. What is the polynomial-time inapproximability of SLPO restricted to such inputs in the planar graph case?

Acknowledgements We would like thank an anonymous reviewer of a previous version of this article for suggesting the polynomial-time dynamic programming solution mentioned in Sect. 1.2.

References

Aggarwal A, Klawe MM, Moran S, Shor P, Wilber R (1987) Geometric applications of a matrix-searching algorithm. Algorithmica 2:195–208

Asahiro Y, Jansson J, Miyano E, Ono H (2015) Graph orientations optimizing the number of light or heavy vertices. Journal of Graph Algorithms and Applications 19(1):441–465

Asahiro Y, Jansson J, Miyano E, Ono H, Zenmyo K (2011) Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. J Comb Optim 22(1):78–96

Asahiro Y, Miyano E, Ono H, Zenmyo K (2007) Graph orientation algorithms to minimize the maximum outdegree. Int J Found Comput Sci 18(2):197–215
Borradaile G, Iglesias J, Migler T, Ochoa A, Wilfong G, Zhang L (2017) Egalitarian graph orientations. Journal of Graph Algorithms and Applications 21(4):687–708
Chen KY, Chao KM (2007) On the range maximum-sum segment query problem. Discret Appl Math 155(16):2043–2052
Chrobak M, Eppstein D (1991) Planar orientations with low out-degree and compaction of adjacency matrices. Theoret Comput Sci 86(2):243–266
Dailey DP (1980) Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. Discret Math 30(3):289–293
Deming RW (1979) Acyclic orientations of a graph and chromatic and independence numbers. Journal of Combinatorial Theory Series B 26(1):101–110
Dyer ME, Frieze AM (1986) Planar 3DM is NP-complete. J Algorithms 7(2):174–184
Elberfeld M, Bafna V, Gamzu I, Medvedovsky A, Segev D, Silverbush D, Zwick U, Sharan R (2011) On the approximability of reachability-preserving network orientations. Internet Mathematics 7(4):209–232
Galil Z, Park K (1990) A linear-time algorithm for concave one-dimensional dynamic programming. Inf Process Lett 33(6):309–311
Gallai T (1968) On directed graphs and circuits. In: Theory of Graphs (Proceedings of the Colloquium held at Tihany 1966). pp. 115–118. Akadémiai Kiadó
Garey M, Johnson D (1979) Computers and Intractability - A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York
Guibas LJ, Sedgewick R (1978) A dichromatic framework for balanced trees. Proceedings of the Nineteenth Annual Symposium on Foundations of Computer Science (FOCS 1978). IEEE Computer Society, pp 8–21
Hasse M (1965) Zur algebraischen Begründung der Graphentheorie. I Mathematische Nachrichten 28(5–6):275–290
Karp RM (1972) Reducibility among combinatorial problems. Proceedings of Complexity of Computer Computations. Plenum Press, pp 85–103 (The IBM Research Symposia Series)
Medvedovsky A, Bafna V, Zwick U, Sharan R (2008) An algorithm for orienting graphs based on cause-effect pairs and its applications to orienting protein networks. In: Proceedings of the Eighth International Workshop on Algorithms in Bioinformatics (WABI 2008). Lecture Notes in Bioinformatics, vol. 5251, pp. 222–232. Springer-Verlag
Roy B (1967) Nombre chromatique et plus longs chemins d'un graphe. Revue française d'informatique et de recherche opérationnelle 1(5):129–132
Venkateswaran V (2004) Minimizing maximum indegree. Discret Appl Math 143(1–3):374–378
Vitaver LM (1967) Determination of minimal coloring of vertices of a graph by means of Boolean powers of the incidence matrix (in Russian). Proceedings of the USSR Academy of Sciences, vol 147. Nauka, pp 758–759
Zuckerman D (2007) Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. Theory of Computing 3(1):103–128

Authors and Affiliations

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

✉ Jesper Jansson
jj@i.kyoto-u.ac.jp

Yuichi Asahiro
asahiro@is.kyusan-u.ac.jp

Avraham A. Melkman
melkmana@gmail.com

Eiji Miyano
miyano@ai.kyutech.ac.jp

Hirotaka Ono
ono@nagoya-u.jp

Quan Xue
quan.xue@connect.polyu.hk

Shay Zakov
zakovs@gmail.com

1 Kyushu Sangyo University, Fukuoka, Japan

2 Kyoto University, Kyoto, Japan

3 Stein Faculty of Computer and Information Science, Ben Gurion University of the Negev, Be'er Sheva, Israel

4 Kyushu Institute of Technology, Iizuka, Japan

5 Nagoya University, Nagoya, Japan

6 The University of Hong Kong, Hong Kong, China

7 Ruppin Academic Center, Kfar Monash, Israel