International Journal of Foundations of Computer Science
Vol. 32, No. 2 (2021) 209–233

DOI: 10.1142/S012905412150012X

World Scientific
www.worldscientific.com

Graph Orientation with Edge Modifications*

Yuichi Asahiro†
Department of Information Science, Kyushu Sangyo University
Fukuoka 813-8503, Japan
asahiro@is.kyusan-u.ac.jp

Jesper Jansson
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
jesper.jansson@polyu.edu.hk

Eiji Miyano
Department of Artificial Intelligence
Kyushu Institute of Technology, Iizuka
Fukuoka 820-8502, Japan
miyano@ces.kyutech.ac.jp

Hirotaka Ono
Graduate School of Informatics
Nagoya University, Nagoya 464-8601, Japan
ono@i.nagoya-u.ac.jp

Sandhya T. P.
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
csstp@comp.polyu.edu.hk

Received 19 February 2020
Accepted 7 October 2020
Published 30 January 2021
Communicated by Francis Chin

The goal of an *outdegree-constrained edge-modification problem* is to find a spanning subgraph or supergraph H of an input undirected graph G such that either: *(Type I)* the number of edges in H is minimized or maximized and H can be oriented to satisfy some specified constraints on the vertices' resulting outdegrees; or: *(Type II)* among

*A preliminary version of this paper appeared in Proceedings of Frontiers in Algorithmics (FAW2019), Vol. 11458 of Lecture Notes in Computer Science, Springer, Cham, 2019, pp. 38–50.
†Corresponding author.

all subgraphs or supergraphs of G that can be constructed by deleting or inserting a fixed number of edges, H admits an orientation optimizing some objective involving the vertices' outdegrees. This paper introduces eight new outdegree-constrained edge-modification problems related to load balancing called *(Type I)* MIN-DEL-MAX, MIN-INS-MIN, MAX-INS-MAX, and MAX-DEL-MIN and *(Type II)* p-DEL-MAX, p-INS-MIN, p-INS-MAX, and p-DEL-MIN. In each of the eight problems, the input is a graph and the goal is to delete or insert edges so that the resulting graph has an orientation in which the maximum outdegree (taken over all vertices) is small or the minimum outdegree is large. We first present a framework that provides algorithms for solving all eight problems in polynomial time on unweighted graphs. Next we investigate the inapproximability of the edge-weighted versions of the problems, and design polynomial-time algorithms for six of the problems on edge-weighted trees.

Keywords: Graph orientation; maximum flow; computational complexity; inapproximability; load balancing.

1. Introduction

Graph modification problems are fundamental in graph theory and arise in many theoretical and practical settings, including computational biology [16] and machine learning [6]. Given a weighted or unweighted graph $G = (V, E)$ and a graph property Π, the general objective is to transform the graph G into a graph G' satisfying property Π by applying a shortest sequence of graph modification operations. There are two main types of graph modification problems: *vertex-modification problems* and *edge-modification problems*. In the former, one is allowed to add or remove vertices to or from the graph, while in the latter, the goal is typically to find a spanning subgraph or supergraph of G satisfying property Π.

A special case of edge-modification problems is when the property Π depends on the vertices' degrees. Such *degree-constrained edge-modification* problems are very general and include many natural problems such as the Maximum Weight Perfect Matching, Maximum r-Factor, Longest Cycle, and General Factor problems. Indeed, one can regard Maximum Weight Perfect Matching (or Maximum r-Factor) as the problem of finding a subgraph G' of G such that: (i) the degree of every vertex in G' is one (or exactly r); and (ii) the total weight of the deleted edges is minimized. Similarly, Longest Cycle is equivalent to the problem of finding a subgraph G' such that: (i) the degree of every vertex in G' is two; (ii) G' is connected; and (iii) the total weight of the deleted edges is minimized. General Factor [13, 14] asks if it is possible to delete edges from G so that the resulting graph G' is connected and every vertex v in G' has degree equal to a number belonging to a specified set $K(v)$. Many of these problems are NP-hard; e.g., Longest Cycle as well as General Factor are NP-hard even for unweighted graphs. Therefore, it is important to identify special cases of them that can be solved efficiently. One such special case of General Factor is the new problem MIN-DEL-MAX, introduced below.

An *orientation* of an undirected graph is an assignment of a direction to each of its edges. By an *outdegree-constrained edge-modification* problem, we mean an

edge-modification problem where the solution is required to admit an orientation in which the vertices' outdegrees satisfy some specified constraints.

This paper introduces eight new outdegree-constrained edge-modification problems that are related to load balancing. To illustrate the connection, consider the following example. Suppose there is a set of jobs to be completed and a set of available machines such that:

- each job can be processed by exactly one of two specified machines;
- for any pair of machines, at most one job is imposed; and
- the goal is to assign each job to a machine while minimizing the maximum load on the machines.

This situation can be represented as an undirected graph whose edges correspond to jobs and whose vertices correspond to machines; any orientation of such a graph then gives an assignment where the tail vertex (machine) of each directed edge has to process that edge (job). Next, suppose that unfortunately, after finding an optimal solution, it turns out that the maximum load is too high, and so we have to give up trying to complete all of the jobs. Instead, we compute the fewest jobs to abandon in order to decrease the resulting maximum load to within some reasonable amount. In other words, the new goal becomes *to find a smallest possible set of edges to delete from the graph so that the remaining graph can be oriented without the outdegree of any vertex having to go over some specified limit.* This problem is what we call MIN-DEL-MAX.

The other seven problems introduced below have similar interpretations. As another example, in the p-INS-MAX problem, the goal is to assign jobs to machines while maximizing the minimum load (rather than minimizing the maximum) and in addition to the jobs that can only be processed by one of their two specified machines, we are allowed to introduce p new jobs to help increase the minimum load.

We first provide algorithms for solving all of the eight new problems in polynomial time on unweighted graphs. We then prove that each of their generalizations to edge-weighted graphs cannot be approximated within a ratio of either $\rho(n)$, 1.5 or 2.0 in polynomial time unless P = NP, where n is the number of vertices in the input graph and $\rho(n) \geq 1$ is any polynomial-time computable function. These inapproximability results hold even for planar bipartite graphs. Finally, as a tractable subclass of the planar bipartite graphs, we consider the problems on edge-weighted trees.

1.1. *Problem definitions*

Let $G = (V, E, w)$ be a simple, undirected, edge-weighted graph, where V and E are the set of vertices and the set of edges, respectively, and w is a function assigning a positive integer (weight) to each edge. If $w(e) = c$ for every edge $e \in E$ and c is a positive integer, then G can be considered to be an unweighted graph. For

simplicity, any unweighted graph G is assumed to satisfy $w(e) = 1$ for every $e \in E$, and is defined by V and E only (that is, $G = (V, E)$).

The vertex set and the edge set of G are denoted by $V(G)$ and $E(G)$, respectively. For any $u, v \in V$, the undirected edge with endpoints in u and v is denoted by $\{u, v\}$, and the directed edge from u to v is denoted by (u, v). For G and $V' \subseteq V$, the subgraph of G induced by V' is denoted by $G[V']$. An *orientation* Λ of G is an assignment of a direction to each edge in E, i.e., every $\{u, v\} \in E$ is set to either (u, v) or (v, u). (Equivalently, Λ is a set of directed edges that consists of exactly one of the two directed edges (u, v) and (v, u) for every $\{u, v\} \in E$.) Let $\Lambda(G)$ denote the directed graph (V, Λ, w), where $w((u, v)) = w(\{u, v\})$ for $(u, v) \in \Lambda$ and $\{u, v\} \in E$. For any $v \in V$ and a fixed orientation Λ of G, define $d^+(v)$ as the weighted outdegree of v under Λ, i.e., $d^+(v)$ represents the total weight of v's outgoing edges. In an unweighted graph, $d^+(v)$ equals the number of edges directed outwards from v. Also, let $\Gamma(G)$ be the set of all orientations of G. Finally, for any unweighted graph $G = (V, E)$, its complement $(V, \overline{E})$ is denoted by G^c, where $\overline{E} = \{\{u, v\} \mid u, v \in V, u \neq v, \{u, v\} \notin E\}$.

We now define the first four new graph orientation problems. Each of them takes as input a simple, undirected graph $G = (V, E, w)$ and a positive integer k.

- MIN-DEL-MAX: (Assumes w.l.o.g. that $k \leq \min_{\Lambda \in \Gamma(G)} \max_{u \in V} d^+(u)$.) Find the minimum number of edges whose deletion results in a graph G' with $\min_{\Lambda \in \Gamma(G')} \max_{u \in V} d^+(u) \leq k$.
- MIN-INS-MIN: (Assumes w.l.o.g. that $k \geq \max_{\Lambda \in \Gamma(G)} \min_{u \in V} d^+(u)$.) Find the minimum number of edges whose addition results in a simple graph G' with $\max_{\Lambda \in \Gamma(G')} \min_{u \in V} d^+(u) \geq k$.
- MAX-INS-MAX: (Assumes w.l.o.g. that $k \geq \min_{\Lambda \in \Gamma(G)} \max_{u \in V} d^+(u)$.) Find the maximum number of edges whose addition results in a simple graph G' with $\min_{\Lambda \in \Gamma(G')} \max_{u \in V} d^+(u) \leq k$.
- MAX-DEL-MIN: (Assumes w.l.o.g. that $k \leq \max_{\Lambda \in \Gamma(G)} \min_{u \in V} d^+(u)$.) Find the maximum number of edges whose deletion results in a graph G' with $\max_{\Lambda \in \Gamma(G')} \min_{u \in V} d^+(u) \geq k$.

The above four problems optimize the number of edges to delete or insert while ensuring that the graph can be oriented to respect a given bound k on the vertices' outdegrees. Observe that in the problems MIN-INS-MIN and MAX-INS-MAX, the resulting graph must be simple. Furthermore, for the edge-weighted problem variants, the objective is still to optimize the number of deleted/inserted edges (rather than their total weight). Also, the weights of the inserted edges can be any positive integers.

Next, we define four additional problems. Each of them takes as input a simple, undirected graph $G = (V, E, w)$, but now the number of edges to be deleted/inserted

is a fixed integer p, and the objective is to compute the minimum/maximum value of the bound k that is achieved after deleting/inserting exactly p edges. For p-INS-MIN and p-INS-MAX, the weight of an inserted edge can be arbitrary as long as it is a positive integer.

- p-DEL-MAX: Find the smallest possible value of $\min_{\Lambda\in\Gamma((V,E\setminus E'))}\max_{u\in V} d^+(u)$ taken over all $E'\subseteq E$ with $|E'|=p$.
- p-INS-MAX: Find the smallest possible value of $\min_{\Lambda\in\Gamma((V,E\cup E'))}\max_{u\in V} d^+(u)$ taken over all $E'\subseteq E(G^c)$ with $|E'|=p$.
- p-INS-MIN: Find the largest possible value of $\max_{\Lambda\in\Gamma((V,E\cup E'))}\min_{u\in V} d^+(u)$ taken over all $E'\subseteq E(G^c)$ with $|E'|=p$.
- p-DEL-MIN: Find the largest possible value of $\max_{\Lambda\in\Gamma((V,E\setminus E'))}\min_{u\in V} d^+(u)$ taken over all $E'\subseteq E$ with $|E'|=p$.

We have just defined eight related problems. Their names follow the pattern A-B-C, where:

- A is MIN, MAX, or p, depending on whether the number of modified edges in the input graph is to be minimized, to be maximized, or equal to p.
- B is either DEL or INS, specifying whether edges are to be deleted or inserted.
- C is MIN or MAX, depending on whether the minimum outdegree (taken over all vertices) should be large or the maximum outdegree should be small.

Throughout the paper, let $n=|V|$ and $m=|E|$ for any given instance of the above eight problems. Δ is the (unweighted) maximum degree taken over all vertices in the input G. Any algorithm `ALG` is called a σ-approximation algorithm if the following inequality holds for every input graph G: $\max\left\{\frac{\#ALG(G)}{\#OPT(G)},\frac{\#OPT(G)}{\#ALG(G)}\right\}\leq\sigma$, where $\#ALG(G)$ and $\#OPT(G)$ are the number of deleted or inserted edges by `ALG` and an optimal algorithm, respectively. Note that the output of each of the eight problems defined above is a number. Consequently, the algorithms designed in this article output numbers. In case an actual set of edges to delete or insert (along with a corresponding orientation) is required, it is straightforward to modify the algorithms accordingly.

1.2. *Related work*

Given an undirected unweighted/edge-weighted graph G, the objective of Minimum Maximum Outdegree Problem (MinMaxO) is to find an orientation of G such that the maximum outdegree of a vertex is minimized. In other words, MinMaxO computes $\min_{\Lambda\in\Gamma(G)}\max_{u\in V} d^+(u)$ for the input undirected graph. The problem MinMaxO was previously studied in, e.g., [4, 7, 8, 10, 17]. If the

input is restricted to unweighted graphs, then MinMaxO can be solved in linear time for planar graphs [10] and in polynomial time for general graphs [17]. The problem of computing $\max_{\Lambda \in \Gamma(G)} \min_{u \in V} d^+(u)$ for an undirected graph G is called Maximum Minimum Outdegree Problem (MaxMinO) in [2]: Given an undirected unweighted/edge-weighted graph G, the objective of MaxMinO is to find an orientation of G such that the minimum outdegree of a vertex is maximized. MaxMinO can also be solved in polynomial time for unweighted graphs (Theorem 8 in [2]). (This is why the input k to MIN-DEL-MAX, MIN-INS-MIN, MAX-INS-MAX, and MAX-DEL-MIN can be assumed w.l.o.g. to satisfy $k \leq \min_{\Lambda \in \Gamma(G)} \max_{u \in V} d^+(u)$ or $k \geq \max_{\Lambda \in \Gamma(G)} \min_{u \in V} d^+(u)$.) On the other hand, when the input is generalized to edge-weighted graphs, both MinMaxO and MaxMinO become NP-hard [2, 4].

A variant of MinMaxO in which one may perform p *split operations* on the vertices (corresponding to adding p extra machines in the load balancing setting described above) before orienting the edges was studied in [1]. That problem seems harder than the eight problems studied here, as it is NP-hard even for unweighted graphs when p is unbounded [1].

1.3. *Our contributions and organization of the paper*

Section 2 presents polynomial-time algorithms for the new problems on unweighted graphs. See Table 1 for a summary of their computational complexity. In Sec. 3 we develop polynomial-time algorithms for six of the eight problems on edge-weighted

Table 1. The computational complexity of the algorithms in Sec. 2 for unweighted graphs.

Problem	Time complexity	Reference
MIN-DEL-MAX	$O(m^2 \log n)$	Theorem 3
MIN-INS-MIN	$O(n^4 \log n)$	Theorem 5
MAX-INS-MAX	$O(n^4 \log n)$	Theorem 5
MAX-DEL-MIN	$O(m^{3/2} \log m \log^2 \Delta)$	Theorem 4
p-DEL-MAX	$O(m^2 \log n)$	Theorem 3
p-INS-MIN	$O(n^4 \log n)$	Theorem 5
p-INS-MAX	$O(n^4 \log n)$	Theorem 5
p-DEL-MIN	$O(m^2 \log n)$	Theorem 3

Table 2. The computational complexity of the algorithms in Sec. 3.1 for edge-weighted trees, where w_{max} is the maximum weight of edges.

Problem	Time complexity	Reference
MIN-DEL-MAX	$O(n)$	Theorem 8
MIN-INS-MIN	$O(n)$	Theorem 10
MAX-INS-MAX	unknown	
MAX-DEL-MIN	$O(n)$	Theorem 7
p-DEL-MAX	$O(n)$	Theorem 8
p-INS-MIN	$O(n \log w_{max} \Delta)$	Theorem 11
p-INS-MAX	unknown	
p-DEL-MIN	$O(1)$	Theorem 7

Table 3. The inapproximability in Sec. 3.2 for edge-weighted planar bipartite graphs, where $\rho(n) \geq 1$ is any polynomial-time computable function.

Problem	Inapproximability	Reference
MIN-DEL-MAX	$\rho(n)$	Theorem 12
MIN-INS-MIN	$\rho(n)$	Theorem 14
MAX-INS-MAX	$\rho(n)$	Theorem 15
MAX-DEL-MIN	$\rho(n)$	Theorem 16
p-DEL-MAX	1.5	Theorem 13
p-INS-MIN	2	Theorem 14
p-INS-MAX	1.5	Theorem 17
p-DEL-MIN	2	Theorem 17

trees that are summarized in Table 2. Then we show the polynomial-time inapproximability for planar bipartite edge-weighted graphs for all eight problems, as summarized in Table 3. Finally, we conclude the paper in Sec. 4.

2. Unweighted Graphs

In this section, all graphs are assumed to be unweighted. We present polynomial-time algorithms for the new problems on unweighted graphs. Rather than developing a separate algorithm for each problem, our strategy is to give a unified framework from which each of the eight algorithms follows as a special case.

2.1. *The directed graphs N_G and N_H*

We take advantage of the problems' structural similarities by encoding the input graph G in all eight cases as a directed graph N_G, augmenting N_G with edge capacities as defined below to obtain a flow network, and then using binary search together with a fast algorithm for computing maximum flows. N_G is the same for all problems; only the edge capacities in the flow network depend on which of the problems is being solved. The encoding used here is an extension of the one in [4]; to be precise, the definition of N_G follows the basic construction in [4] and then adds auxiliary vertices and directed edges that can capture the deletion and insertion of edges in the input graph.

The formal definition of N_G is as follows. For any graph $G = (V, E)$, construct the directed graph $N_G = (V_G, E_G)$ with vertex set V_G and edge set E_G by defining:

$$\begin{aligned}
\overline{E} &= \{\{u, v\} \mid u, v \in V, u \neq v, \{u, v\} \notin E\} \\
V_G &= V \cup E \cup \overline{E} \cup \{x, y, r, s, t\} \\
E_G &= \{(s, v) \mid v \in V\} \cup \{(u, e), (v, e) \mid e = \{u, v\} \in E \cup \overline{E}\} \cup \{(r, e) \mid e \in E\} \\
&\quad \cup \{(e, x) \mid e \in E\} \cup \{(e, y) \mid e \in \overline{E}\} \cup \{(s, r), (x, t), (y, t)\}.
\end{aligned}$$

Note that if $e = \{u, v\} \in E$ (or $\overline{E}$), then N_G contains two directed edges (u, e) and (v, e) for e in the vertex subset E (or $\overline{E}$) of N_G. Note that the vertex r and the set

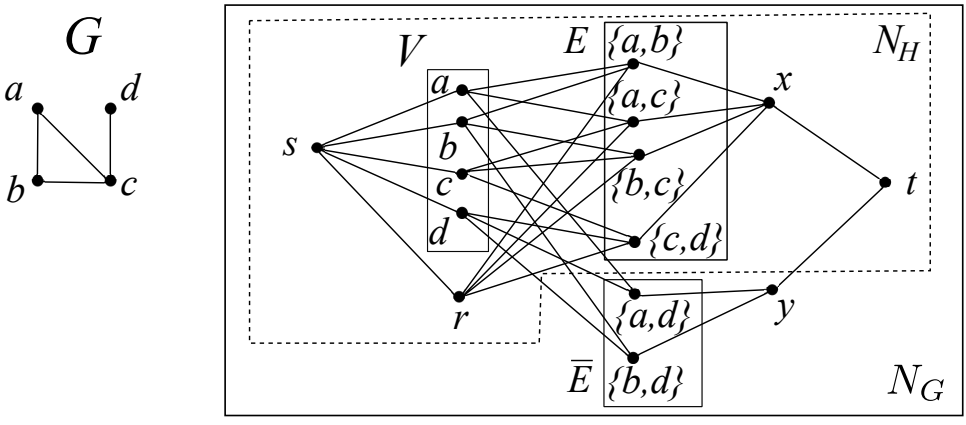

Fig. 1. An example of an unweighted graph G and the directed graph N_G constructed from G. All edges are directed from left to right in N_G. The part surrounded by the dashed lines represents N_H.

of vertices in $\overline{E}$ capture deletion and insertion of edges respectively, mentioned in the previous paragraph. See Fig. 1 for an illustration.

For the capacities of edges in N_G, we use three parameters α, β, and γ whose values are positive integers that will be specified by the algorithms. Intuitively, α, β, and γ respectively correspond to the number of edges deleted from G, the number of edges inserted into G, and maximum/minimum outdegree of a vertex in $\Lambda(G)$, where an orientation Λ of G is obtained from a maximum flow in N_G. For notational convenience, let $N_G(\alpha;\beta;\gamma) = (V_G, E_G, cap)$ be the flow network obtained by augmenting N_G with edge capacities cap:

$$cap(a) = \begin{cases} \gamma, & \text{if } a = (s,v) \text{ for } v \in V \\ \alpha, & \text{if } a = (s,r) \\ |E|, & \text{if } a = (x,t) \\ \beta, & \text{if } a = (y,t) \\ 1, & \text{otherwise.} \end{cases}$$

For each of the problems, we will fix two of the three parameters α, β, and γ, and then find minimum/maximum value of the remaining one parameter such that the size of a maximum flow in $N_G(\alpha;\beta;\gamma)$ satisfies a condition which will be given later. We remark that maximum flows in suitably defined flow networks were previously used to solve some other graph orientation problems in [1–4]. The definition of N_G that we present here is more general.

Section 2.2 shows two properties of N_G which will be used to develop the algorithms. In Sec. 2.3, we design the algorithms with N_G and analyze the time complexity of the obtained algorithms. To solve MIN-INS-MIN, p-INS-MIN, MAX-INS-MAX, and p-INS-MAX, we need to explicitly construct the entire directed graph N_G. However, for MIN-DEL-MAX, p-DEL-MAX, MAX-DEL-MIN, and p-DEL-MIN, the algorithms will only need the induced subgraph

Table 4. Summary of how to assign the edge capacities in the flow network to solve the different problems. A "–" indicates that the edge (y, t) is not included in the network.

Problem	Graph	$cap(x, v)$	$cap(s, r)$	$cap(y, t)$	$cap(x, t)$	others
MIN-DEL-MAX	N_H	k	α	–	$\|E\|$	1
MIN-INS-MIN	N_G	k	0	β	$\|E\|$	1
MAX-INS-MAX	N_G	k	0	β	$\|E\|$	1
MAX-DEL-MIN	N_H	k	α	–	$\|E\|$	1
p-DEL-MAX	N_H	γ	p	–	$\|E\|$	1
p-INS-MIN	N_G	γ	0	p	$\|E\|$	1
p-INS-MAX	N_G	γ	0	p	$\|E\|$	1
p-DEL-MIN	N_H	γ	p	–	$\|E\|$	1

$N_H = N_G[V \cup E \cup \{s, r, x, t\}]$ of N_G, and so these algorithms' running times will be lower when G is sparse. Table 4 summarizes how to set the edge capacities in N_G to solve the eight problems; please refer to the subsections below for details.

2.2. *Properties of N_G*

The following lemma relates the size of the maximum flow in $N_G(\alpha; \beta; \gamma)$ to the number of edges that need to be deleted or inserted to obtain an orientation with bounded outdegree. This lemma plays an important role in designing algorithms for the four problems MIN-DEL-MAX, MAX-INS-MAX, p-DEL-MAX, and p-INS-MAX.

Lemma 1. *There exists a flow in $N_G(\alpha; \beta; \gamma)$ with value $|E| + \beta$ if and only if there are two sets $S_D \subseteq E$ and $S_I \subseteq \overline{E}$ with $|S_D| = \alpha$ and $|S_I| = \beta$ such that modifying G by deleting S_D and inserting S_I yields a graph that can be oriented so that every vertex has outdegree at most γ.*

Proof. ($\Rightarrow$) Suppose there exists a flow in $N_G(\alpha; \beta; \gamma)$ with the value $|E| + \beta$. Since the edge capacities are integers, we can assume that every edge has a nonnegative integral flow by the integrality theorem (see, e.g., [11]). To obtain a total flow of $|E| + \beta$, both edges incident to t must be utilized to their maximum capacity, which means that: (i) exactly $|E|$ units of flow enter x; and (ii) exactly β units of flow enter y.

Define the set S_D as follows. Let j be the amount of flow through the vertex r. Then there are j directed edges in $N_G(\alpha; \beta; \gamma)$ of the form (r, e), where $e \in E$, through which a unit of flow passes. For each such (r, e), include the corresponding edge e from G in S_D. Since the capacity of the edge (s, r) was set to α, it holds that $j \leq \alpha$. Also insert $\alpha - j$ additional arbitrarily chosen edges from G into S_D. This gives $|S_D| = \alpha$. Similarly, let S_I be the subset of $\overline{E}$ such that for every $e \in S_I$, a unit of flow passes through (e, y) in $N_G(\alpha; \beta; \gamma)$. By property (ii), $|S_I| = \beta$.

Now let G' be the graph obtained from G by deleting S_D and inserting S_I. Construct an orientation of G' as follows: For every edge of the form (v, e) in $N_G(\alpha; \beta; \gamma)$ that receives a unit of flow, where $v \in V$ and $e \in E$, orient the edge e in G' away from v. This gives an orientation of G' because by property (i), every

original edge e in E receives a unit of flow from either r, in which case e does not exist in G', or from a vertex in V, in which case e is assigned a direction in G'. Finally, for each $u \in V$, at most γ units of flow can exit u, so $d^+(u) \leq \gamma$ and hence $\min_{\Lambda \in \Gamma(G')} \max_{u \in V} d^+(u) \leq \gamma$.

($\Leftarrow$) Conversely, suppose there is a set S_D of α edges in E and a set S_I of β edges in $\overline{E}$ such that deletion of S_D from G with insertion of S_I to G yields a graph G' with $\min_{\Lambda \in \Gamma(G')} \max_{u \in V} d^+(u) \leq \gamma$. Let Λ be a fixed orientation of G' with $\max_{u \in V} d^+(u) \leq \gamma$. For each $e = \{u, v\} \in (E \setminus S_D) \cup S_I$, if e is oriented as (u, v) in Λ then define $e_0 = u$; otherwise, define $e_0 = v$. Construct a flow in $N_G(\alpha; \beta; \gamma)$ as follows:

- For each $e \in E \setminus S_D$, send one unit of flow from s to e_0, from e_0 to e, from e to x, and from x to t. The contribution to the total flow is $|E| - \alpha$.
- For each $e \in S_D$, send one unit of flow from s to r, from r to e, from e to x, and from x to t. The contribution to the total flow is α.
- For each $e \in S_I$, send one unit of flow from s to e_0, from e_0 to e, from e to y, and from y to t. The contribution to the total flow is β.

No edge capacity constraints are violated and the total flow is thus $|E| - \alpha + \alpha + \beta = |E| + \beta$. □

The following lemma for the problems MIN-INS-MIN, MAX-DEL-MIN, p-INS-MIN, and p-DEL-MIN is very similar to Lemma 1, which again relates the size of the maximum flow in $N_G(\alpha; \beta; \gamma)$ to the number of edges that need to be deleted or inserted.

Lemma 2. *There exists a flow in $N_G(\alpha; \beta; \gamma)$ with value $n\gamma + \alpha$ if and only if there are two sets $S_D \subseteq E$ and $S_I \subseteq \overline{E}$ with $|S_D| = \alpha$ and $|S_I| = \beta$ such that modifying $G = (V, E)$ by deleting S_D and inserting S_I yields a graph that can be oriented so that every vertex has outdegree at least γ.*

Proof. ($\Rightarrow$) The proof is analogous to the proof of Lemma 1. The main difference is that all the edges incident to s (instead of t) must be utilized to their maximum capacity. The set S_D includes α edges from G corresponding to the α directed edges of the form (r, e) through which a unit of flow passes. Then, S_I similarly includes the set S of edges from G corresponding to the directed edges of the form (e, y) through which a unit of flow passes, and also includes an arbitrary set of $\beta - |S|$ edges in $\overline{E} \setminus S$ if $|S| < \beta$. Using a similar argument as in the proof of Lemma 1, we can define G' as the graph obtained from G by deleting S_D and inserting S_I, and find an orientation of G' with $d^+(u) \geq \gamma$ for all $u \in V$ and hence $\max_{\Lambda \in \Gamma(G')} \min_{u \in V} d^+(u) \geq \gamma$. (Note that the maximum flow may not determine the directions of all edges in G'. Such edges can be oriented arbitrarily because they will never decrease the outdegree of a vertex.)

($\Leftarrow$) As in the second part of the proof of Lemma 1, let G' be the modified graph and let Λ be its orientation. Construct another graph G'' as follows: First, for any vertex

u with $d^+(u) > k$ in $\Lambda(G')$, remove $d^+(u) - k$ arbitrary outgoing edges from u; then, replace every directed edge (u, v) by an undirected edge $\{u, v\}$. The maximum flow of $N_G(\alpha; \beta; \gamma)$ is constructed based on $\Lambda(G'')$ as in the proof of Lemma 1. Finally, its size is $(n\gamma - \beta) + \alpha + \beta = n\gamma + \alpha$, where $n\gamma - \beta$, α, and β are respectively the (total) sizes of the flows in the edges of the form (v, e), the edge (s, r), and the edge (y, t). □

2.3. *Algorithms*

2.3.1. *Algorithm descriptions*

We first present an algorithm for solving MIN-DEL-MAX based on Lemma 1 and a binary search. Then, we explain how to modify the algorithm to MAX-INS-MAX, p-DEL-MAX, and p-INS-MAX, and how to use Lemma 2 to solve MIN-INS-MIN, MAX-DEL-MIN, p-INS-MIN, and p-DEL-MIN in the same way. All of the resulting algorithms run in polynomial time. Recall that some of the problems only need the subgraph N_H of N_G. This will lead to more efficient algorithms for them than those that use the whole N_G.

To solve MIN-DEL-MAX, we set $\beta = 0$, and $\gamma = k$. Setting $\beta = 0$ means that we do not insert any edge into G, and hence we only need the part N_H of N_G here. Let $N_H(\alpha; k)$ denote the subgraph of $N_G(\alpha; 0; k)$ induced by $\{s, r, x, y, t\} \cup V \cup E$. Recall that the value of k is already fixed in MIN-DEL-MAX. By Lemma 1, the minimum number of edges that need to be deleted to obtain a graph G' with $\min_{\Lambda \in \Gamma(G')} \max_{u \in V} d^+(u) \leq k$ $(= \gamma)$ is the same as the smallest value of α such that there exists a flow in $N_H(\alpha; k)$ (or $N_G(\alpha; 0; k)$) of size $|E|$. A binary search on α will give this minimum value. The algorithm's pseudocode is listed in Fig. 2. The binary search on α is conducted by the lines 1–3 and 6–11 of the algorithm. Then the construction of $N_H(\alpha; k)$ and solving the maximum flow problem on $N_H(\alpha; k)$ are done in lines 4 and 5.

The algorithms for the other seven problems are analogous to the above. For this reason, we omit the pseudocodes and only give a brief explanation of how to adapt the algorithm to each of the problems.

For p-DEL-MAX, we set $\alpha = p$ and $\beta = 0$, and solve the maximum flow problem for $N_H(p; \gamma)$ (or $N_G(p; 0; \gamma)$). Since the capacity γ of edges of the form (s, v) corresponds to the outdegree of v, we do a binary search on γ to find the smallest value of γ for which there is a flow of size $|E|$ in $N_H(p; \gamma)$. Similarly, to solve MAX-INS-MAX, we set $\alpha = 0$ and $\gamma = k$. By Lemma 1, the maximum number of edges that may be inserted to obtain a graph G' with $\min_{\Lambda \in \Gamma(G')} \max_{u \in V} d^+(u) \leq k$ is the same as the largest value of β such that there exists a flow in $N_G(0; \beta; k)$ of size $|E| + \beta$. We use a binary search on β to find the maximum value of β for which the condition holds. For p-INS-MAX, set $\alpha = 0$ and $\beta = p$. We use a binary search on γ to find the smallest γ such that there is a flow in $N_G(0; p; \gamma)$ of size $|E| + p$.

For the remaining four problems MIN-INS-MIN, MAX-DEL-MIN, p-INS-MIN, and p-DEL-MIN, we use the same strategy but apply Lemma 2 instead of Lemma 1: For MIN-INS-MIN, we set $\alpha = 0$ and $\gamma = k$. Then a binary search

```
1: α_min ← 0, α_max ← |E|;
2: repeat
3:     α ← ⌊(α_min + α_max)/2⌋;
4:     Construct N_H(α; k);
5:     Solve the maximum flow problem on N_H(α; k). Let f be the
       value of the maximum flow of N_H(α; k);
6:     if f = |E| then
7:         α_max ← α;
8:     else
9:         α_min ← α;
10:    end if
11: until α_min ≥ α_max
12: Output α and halt;
```

Fig. 2. The algorithm for MIN-DEL-MAX on unweighted graphs.

on β is conducted to find the minimum value of β such that there exists a flow in $N_G(0;\beta;k)$ of size nk. For p-INS-MIN, we set $\alpha = 0$ and $\beta = p$. Then we use a binary search on γ to find the maximum value of γ such that there exists a flow in $N_G(0;p;\gamma)$ of size $n\gamma$. For MAX-DEL-MIN, we set $\beta = 0$ and $\gamma = k$. Again we use a binary search on α to find the maximum value of α such that there exists a flow in $N_H(\alpha;k)$ of size $nk+\alpha$. For p-DEL-MIN, we set $\alpha = p$ and $\beta = 0$, and then use a binary search on γ to find the maximum value of γ such that there exists a flow in $N_H(p;\gamma)$ of size $n\gamma + p$.

2.3.2. *Time complexity of the algorithms*

First we analyze the time complexity of the algorithms utilizing N_H. Let $n = |V|$ and $m = |E|$ for a given graph $G = (V, E)$, and let $N = |V(N_H)|$ and $M = |E(N_H)|$. We note that for the directed graph N_H, $N = n+m+4$ and $M = n+4m+2$. For MIN-DEL-MAX, the search for $\alpha = O(m)$ can be carried out using the binary search technique, which takes $O(\log m) = O(\log n)$ time since $m = O(n^2)$. The maximum flow problem on $N_H(\alpha;k)$ can be solved in $O(MN)$ time [15], so MIN-DEL-MAX can also be solved in $O(m^2 \log n)$ time. Since $\gamma = O(n)$, in a similar way, we can analyze the time complexity of our algorithms for p-DEL-MAX, MAX-DEL-MIN, and p-DEL-MIN. Thus we have:

Theorem 3. *For unweighted graphs*, MIN-DEL-MAX, p-DEL-MAX, MAX-DEL-MIN, *and* p-DEL-MIN *can be solved in* $O(m^2 \log n)$ *time.*

For MAX-DEL-MIN, the time complexity can in fact be further improved by using Theorem 8 in [2] instead, which states that MaxMinO is solvable in $O(m^{3/2} \log m \log^2 \Delta)$ time for unweighted graphs. More precisely, we have:

Theorem 4. MAX-DEL-MIN *can be solved in* $O(m^{3/2} \log m \log^2 \Delta)$ *time for unweighted graphs.*

Proof. First compute an orientation by which the minimum outdegree has value $\max_{\Lambda \in \Gamma(G)} \min_{u \in V} d^+(u)$ ($\geq k$) using the algorithm for MaxMinO for unweighted graphs from [2], and obtain a directed graph. Then delete $d^+(v) - k$ arbitrary outgoing edges from each vertex v in the directed graph to get a directed graph G' with $d^+(v) = k$ for every $v \in V$. This deletion of edges only needs linear time since we can delete an arbitrary set of outgoing edges for each vertex. The number of deleted edges is the maximum possible: Since every vertex has outdegree k, the number of directed edges in the graph is nk, and so deleting any more edges would result in some vertex having outdegree strictly less than k by the pigeonhole principle. Thus MAX-DEL-MIN can be solved in $O(m^{3/2} \log m \log^2 \Delta)$ time. □

As for the other four algorithms utilizing N_G, let $N = |V_G| = \frac{n(n+1)}{2} + 5$ and $M = |E_G| = \frac{n(3n-1)}{2} + m + 3$ in the directed graph N_G. For MIN-INS-MIN, the search for $\beta = O(m)$ can be carried out using the binary search technique and therefore takes $O(\log m) = O(\log n)$ time. The maximum flow problem on $N_G(0; \beta; k)$ can be solved in $O(MN)$ time [15], so MIN-INS-MIN can also be solved in $O(n^4 \log n)$ time. Using $\gamma = O(m)$, we can bound the time complexity of our algorithms for p-INS-MIN, MAX-INS-MAX, and p-INS-MAX in the same way. We obtain:

Theorem 5. *For unweighted graphs*, MIN-INS-MIN, p-INS-MIN, MAX-INS-MAX, *and* p-INS-MAX *can be solved in* $O(n^4 \log n)$ *time.*

We remark that at the end of each of the eight algorithms, one may also output the edges to be deleted/inserted and an orientation of the resulting graph that corresponds to the computed optimal value without increasing the time complexity simply by taking the solution to the maximum flow problem that was obtained in the final iteration and applying the constructions in the proofs of Lemmas 1 and 2.

3. Edge-Weighted Graphs

In this section, we consider edge-weighted graphs. First, in Sec. 3.1, we restrict our attention to edge-weighted trees, and develop polynomial-time algorithms for six of the eight problems. Then, Sec. 3.2 shows the inapproximability for all of the eight problems on edge-weighted planar bipartite graphs. Unless stated otherwise, all graphs considered in this section are edge-weighted. Throughout this section, although the input graphs belong to some restricted graph class, the edge-modified graphs have no such restrictions.

3.1. *Polynomial-time algorithms for edge-weighted trees*

Assuming P≠ NP, inapproximability of the problems on planar bipartite graphs will be shown in Sec. 3.2. In this section, we design exact polynomial-time algorithms for some of the problems on trees which is a representative subclass of planar bipartite graphs.

The important thing here is that we know the optimal costs, i.e., the maximum outdegree of a vertex for MinMaxO and the minimum outdegree of a vertex for MaxMinO on a tree: For MinMaxO, the maximum outdegree of a vertex under any orientation is at least the maximum weight of the edges. Indeed, choosing an arbitrary root and directing all the edges towards the root gives this optimal cost. Then, for MaxMinO, the minimum outdegree of a vertex under any orientation is 0, since there exist only $n-1$ edges so that at least one of n vertices cannot have outgoing edges under any orientation.

Observation 6. For edge-weighted trees, the optimal costs for MinMaxO and MaxMinO are $w_{\max}$ and 0, respectively, where $w_{\max}$ is the maximum weight of edges.

3.1.1. MAX-DEL-MIN *and* p-DEL-MIN

Based on Observation 6, we immediately have the following theorem:

Theorem 7. *For edge-weighted trees,* MAX-DEL-MIN *and* p-DEL-MIN *can be solved in* $O(n)$ *time and* $O(1)$ *time, respectively.*

Proof. Since the optimal cost k' for MaxMinO on edge-weighted trees is 0, the input k of MAX-DEL-MIN must be zero since $k \leq k'$. Hence an algorithm deleting all the edges solves MAX-DEL-MIN, i.e., the maximum number of deleted edges equals the number of edges in the input which can be obtained in $O(n)$ time. Similarly, for p-DEL-MIN, deleting p arbitrary edges obtains the optimal (the largest minimum) outdegree zero. Indeed, we do not delete any edge and can just answer the optimal solution zero, which can be done in $O(1)$ time. □

3.1.2. MIN-DEL-MAX *and* p-DEL-MAX

The solutions to MIN-DEL-MAX and p-DEL-MAX are relatively straightforward, as shown in the next theorem.

Theorem 8. *For edge-weighted trees,* MIN-DEL-MAX *and* p-DEL-MAX *can be solved in* $O(n)$ *time.*

Proof. Let $w_{\max}$ be the maximum weight of edges in the input tree. For MIN-DEL-MAX, the algorithm just outputs the number of edges having weight larger than k, whose running time is $O(n)$. First for the case $k \geq w_{\max}$, we do not need to delete any edge from the input tree, since the optimal orientation of MinMaxO for the input tree gives the maximum outdegree $w_{\max}$ from Observation 6. Hence the number of deleted edges is zero which is clearly optimal. For the case $k < w_{\max}$, if there remains an edge of weight larger than k, the maximum outdegree under any orientation is also larger than k from Observation 6. Therefore the algorithm needs to delete all such edges.

As for p-DEL-MAX, the algorithm first finds an edge having the p-th largest weight (denoted by w_p) breaking ties arbitrarily, and then removes the edges of the largest weight through the p-th largest weight, so that the answer to the problem is the $(p+1)$-st largest weight of the edges. The running time of this algorithm is $O(n)$ by solving the selection problem [11]. If there remains an edge whose weight is larger than w_p in the resulting graph, the maximum outdegree of a vertex must be greater than w_p from Observation 6. Thus, the removed set of the edges is optimal. □

3.1.3. MIN-INS-MIN *and* p-INS-MIN

To develop fast algorithms for MIN-INS-MIN and p-INS-MIN, a more detailed investigation is needed.

MIN-INS-MIN on stars. Let us consider the case that the input tree is a star, with root r and leaves $v_0, v_1, \ldots, v_{n-2}$. The algorithm for MIN-INS-MIN is listed in Fig. 3. Note that we maintain Λ in the algorithm just to clarify the orientation of edges; to answer the problem we only need to obtain S.

Lemma 9. *For edge-weighted stars,* MIN-INS-MIN *can be solved in $O(n)$ time.*

Proof. If $\sum_{v \in V} w(r, v) < k$, there is no feasible solution, since we cannot insert any edge for the root r, so that the outdegree of r is less than k in any orientation. Lines 1–3 in the algorithm does this evaluation.

Suppose that $\sum_{v \in V} w(r, v) \geq k$. Line 4 of the algorithm partitions the vertex set into two subsets V_s and V_b, depending on whether the weight of the edge incident to a vertex is less than k or not. Consider a vertex $v_i \in V_s$. Even if we orient the

1: **if** $\sum_{v \in V} w(r, v) < k$ **then**
2: Output "No" and halt;
3: **end if**
4: Divide the vertex set V into two sets $V_s = \{v_i \mid w(\{r, v_i\}) < k\}$ and $V_b = \{v_i \mid w(\{r, v_i\}) \geq k\}$;
5: $\Lambda \leftarrow \bigcup_{v_i \in V_s} \{(r, v_i)\}$ and $S \leftarrow \bigcup_{v_i \in V_s} \{\{v_i, v_{(i+1) \bmod (n-1)}\}\}$, where $w(\{v_i, v_{(i+1) \bmod (n-1)}\}) = k$;
6: **if** $\sum_{v_i \in V_s} w(\{r, v_i\}) \geq k$ **then**
7: $\Lambda \leftarrow \Lambda \cup \bigcup_{v_i \in V_b} \{(v_i, r)\}$;
8: **else**
9: Pick one arbitrary vertex v_j from V_b;
10: $\Lambda \leftarrow \Lambda \cup \{(r, v_j)\} \cup \bigcup_{v_i \in V_b \setminus \{v_j\}} \{(v_i, r)\}$;
11: $S \leftarrow S \cup \{\{v_j, v_{(j+1) \bmod (n-1)}\}\}$, where $w(\{v_j, v_{(j+1) \bmod (n-1)}\}) = k$;
12: **end if**
13: Output $|S|$ and halt;

Fig. 3. The algorithm for MIN-INS-MIN for edge-weighted stars.

edge $\{r, v_i\}$ as (v_i, r), we need to add at least one edge for v_i in order to increase its outdegree to k. Hence edges of weight k in V_s are inserted in line 5 of the algorithm. Here, the weight of the inserted edge $\{r, v_i\}$ may be decreased to $k - w(r, v_i)$ by carefully selecting from either of (r, v_i) and (v_i, r) as orientation of the edge $\{r, v_i\}$. However this does not contribute to reduce the number of inserted edges, and so we just orient the edge as (r, v_i) and then set the weight of the inserted edge $\{v_i, v_{(i+1) \mod (n-1)}\}$ as k for every $v_i \in V_s$.

If $\sum_{v_i \in V_s} w(\{r, v_i\}) \geq k$, the outdegree of r under Λ is at least k after line 5 is done, by which we can orient $\{r, v_i\}$ as (v_i, r) for $v_i \in V_b$ (this is done in line 7) and no more edges need to be inserted. Otherwise, i.e. if $\sum_{v_i \in V_s} w(\{r, v_i\}) < k$, we must choose one vertex $v_j \in V_b$ and orient $\{r, v_j\}$ as (r, v_j) to make the outdegree of r at least k, since we cannot insert any edge for the root r (this is done in lines 9 and 10). Here v_j can be arbitrarily selected from V_b, since the weight of the edge incident to such a vertex is at least k. Then we need to insert one more edge of weight k for v_j, which is done in line 11.

As for the running time, every line of the algorithm can be done in $O(n)$ time. Hence the total running time is also $O(n)$. □

MIN-INS-MIN on non-stars. We observe that there is at most one vertex of degree at least $n-2$ in a tree if $n \geq 5$: If there is one vertex of degree $n-1$, the tree forms a star, and there is no other vertex of degree at least $n-2$. Suppose for contradiction that there are two vertices r_1 and r_2 of degree $n-2$. In this case, there exists only one vertex v which is not adjacent to r_1. If $r_2 = v$, then the set of the neighbor vertices of r_1 is the same as the one of r_2, so that there is a cycle which contradicts the fact that the considered graph is a tree. Otherwise, i.e., if $r_2 \neq v$ which means that r_2 is adjacent to r_1, $n-3$ vertices chosen from the neighbor vertices of r_1 and v are adjacent to r_2 that also derives existence of a cycle since $n \geq 5$. This again contradicts the fact that the considered graph is a tree.

Since a brute force algorithm can be used for trees having at most 4 vertices, we assume that the input tree has at least 5 vertices and there is at most one vertex of degree $n-2$ (the input is not a star).

Before describing the algorithm, we introduce notation only used for the algorithm. For a vertex v, its *target* $t(v)$ is a vertex satisfying that $t(v)$ is not adjacent to v in the input graph and if $t(v) = u$ for a vertex u, then $t(u) \neq v$. For a vertex u and a fixed (partial) orientation Λ, $d^+(\Lambda, u) = |\{(u, v) \in \Lambda\}|$, where "partial" means that Λ may determine directions of proper subset of edges in the input graph. For a vertex v, $E(v)$ denotes the set of edges connected to v. Note here that any unoriented degree is not counted in $d^+(\Lambda, u)$. A vertex v is a *boundary vertex* (under Λ) if only one edge in $E(v)$ is not unoriented and the other $|E(v)| - 1$ edges are oriented in Λ. Then, the unoriented edge is named *boundary edge*. Figure 4 is a description of a simple algorithm in a bottom-up greedy manner.

We begin our discussion of the running time by showing that we can find a target for every vertex in linear time: In line 1 of the algorithm, we need to find a

```
 1: For every v ∈ V, find a target t(v);
 2: S ← ∅ and Λ ← ∅;
 3: repeat
 4:     Pick a boundary vertex u and let its connecting boundary edge be e={u, v};
 5:     if d⁺(Λ, u) ≥ k then
 6:         Λ ← Λ ∪ {(v, u)};
 7:     else
 8:         if d⁺(Λ, u) + w(e) ≥ k then
 9:             Λ ← Λ ∪ {(u, v)};
10:         else
11:             S ← S ∪ {{u, t(u)}} with w({u, t(u)}) = k
12:             Λ ← Λ ∪ {(v, u), (u, t(u))};
13:         end if
14:     end if
15: until there is no boundary vertex
16: Output |S| and halt;
```

Fig. 4. The algorithm for MIN-INS-MIN for edge-weighted trees, except for stars.

target $t(v)$ for every v. Let v_1 be the vertex of degree $n-2$ if such a vertex exists, otherwise v_1 is chosen arbitrarily. There is a vertex v_2 which is not adjacent to v_1, and so we set $t(v_1) = v_2$. Since the degree of v_2 is less than $n-2$, there must be a vertex $v_3 \neq v_1$, which is not adjacent to v_2. So, we set $t(v_2) = v_3$. We can repeat this procedure because the vertices except for v_1 have at least two non-adjacent vertices. Repeating this, if a vertex whose target is already chosen is again selected as a target of some vertex (it is OK itself), we pick a vertex from vertices whose targets have not been determined as the start point of this procedure, and then continue. The target $t(v)$ can be chosen arbitrarily from the vertices not adjacent to v except for a vertex whose target is v, which can be done by scanning edges connected to v and maintaining $t(u)$ and $t^{-1}(u)$ for every vertex u. Hence, the line 1 of the algorithm can be done in $O(n)$ time. Since each edge is picked only once in line 4, the total running time of the algorithm is $O(n)$ with maintenance of the set of boundary vertices, Λ, and $d^+(\Lambda, u)$ for every vertex u.

We observe that the output S of the algorithm is optimal. Consider a tree G, a partial orientation Λ of G, and a boundary edge $e = \{u, v\}$ in G.

- The case $d^+(\Lambda, u) \geq k$ (line 5): In this case, orienting $\{u, v\}$ as (v, u) is clearly better than orienting it in the reverse direction (u, v).
- The case $d^+(\Lambda, u) < k$ and $d^+(\Lambda, u) + w(e) \geq k$ (line 8): If we orient the edge e as (v, u), this increases the outdegree of v, however, we need to insert an edge of weight at least $k - d^+(\Lambda, u)$ for u. Moreover we may need one more edge for v to increase its outdegree. Namely, we need to insert one or two edges for this case. On the other hand, orienting the edge e as (u, v) increases the outdegree of

u to at least k, i.e., insertion of an edge is not required for u. By this saving of an insertion, we can insert an edge for v of weight k without increasing the number of inserted edges compared to the case e is oriented as (v, u).

- The case $d^+(\Lambda, u) + w(e) < k$ (line 10): Even if we orient e as (u, v), we need to insert one edge of weight at least $k - (d^+(\Lambda, u) + w(e))$ for u. So an optimal orientation includes (v, u) as the direction of $\{u, v\}$ with an inserted edge of weight at least $k - d^+(\Lambda, u)$, which is better since the number of the inserted edges is the same and v's (temporal) outdegree is larger.

In summary, we have the following theorem by Lemma 9 and the above discussion:

Theorem 10. *For edge-weighted trees,* MIN-INS-MIN *can be solved in* $O(n)$ *time.*

***p*-INS-MIN.** As seen in the algorithm for MIN-INS-MIN in Fig. 4, if $p \geq n$ and the input is not a star, the minimum outdegree can be enlarged as much as we want, since for every vertex, we can prepare a target and insert an edge of arbitrary weight, which can be oriented towards the target. Hence assume $p \leq n - 1$ or the input is a star in the following.

Let us first consider the case that the input is a star and $p \geq n-1$. Let r denote the root of the input. We cannot insert any edge for r. Hence the possible maximum outdegree of r is $\sum_{v \in V \setminus \{r\}} w(\{r, v\})$, where V is the set of vertices in the input. Let this value be B. For leaves $v_0, v_1, \ldots, v_{n-2}$, we insert an edge $\{v_i, v_{(i+1) \mod (n-1)}\}$ of weight B for $1 \leq i \leq n-1$. As a result, every vertex has outdegree B by orienting $\{r, v_i\}$ as (r, v_i) and $\{v_i, v_{(i+1) \mod (n-1)}\}$ as $(v_i, v_{(i+1) \mod (n-1)})$, i.e., the optimal solution (the largest minimum outdegree) is B, which can be obtained in $O(n)$ time.

There are two cases to consider: The input is a star and $p \leq n - 2$, and the input is not a star and $p \leq n - 1$. Both of these cases imply that there is at least one vertex that is not adjacent to an inserted edge. Thus, the largest minimum outdegree of the vertices is at most $w_{\max}\Delta$, where $w_{\max}$ is the maximum weight of edges and Δ is the maximum (unweighted) degree of vertices. Utilizing the algorithm for MIN-INS-MIN, for a fixed ℓ, $1 \leq \ell \leq w_{\max}\Delta$, we check whether there is a set of p edges such that the minimum outdegree is at least ℓ. The search for the largest ℓ (the optimal solution) can be carried out using a binary search and hence takes $O(\log(w_{\max}\Delta))$ time. Therefore the total running time of this algorithm is $O(n \log(w_{\max}\Delta))$, since the algorithm for MIN-INS-MIN spends $O(n)$ time. Then we have the following theorem:

Theorem 11. *For edge-weighted trees,* p-INS-MIN *can be solved in* $O(n \log w_{\max}\Delta)$ *time.*

3.2. *Inapproximability for edge-weighted planar bipartite graphs*

MinMaxO is known to be NP-hard for edge-weighted planar bipartite graphs [5]. This implies the following inapproximability:

Theorem 12. *There is no polynomial-time $\rho(n)$-approximation algorithm for* MIN-DEL-MAX *on edge-weighted planar bipartite graphs unless $P = NP$, where $\rho(n) \geq 1$ is any polynomial-time computable function.*

Proof. Suppose for the sake of obtaining a contradiction that there exists a polynomial-time $\rho(n)$-approximation algorithm `ALG` for some polynomial-time computable function $\rho(n) \geq 1$ for MIN-DEL-MAX on edge-weighted planar bipartite graphs. Then, `ALG` can find an orientation in a given graph G in polynomial time such that the objective value $ALG(G)$ satisfies $OPT(G) \leq ALG(G) \leq \rho(n) \cdot OPT(G)$, where $ALG(G)$ and $OPT(G)$ are the number of deleted edges from G by `ALG` and an optimal algorithm, respectively. Therefore, one can distinguish either $OPT(G) > 0$ or $OPT(G) = 0$ in polynomial time using `ALG` based on the observation that $ALG(G) > 0$ if and only if $OPT(G) > 0$. Checking whether $OPT(G) = 0$ is equivalent to checking whether there is an orientation of G such that the maximum outdegree is at most k, i.e., solving the decision version of MinMaxO with target value k. This contradicts the NP-hardness of MinMaxO. □

As shown in [4], MinMaxO for edge-weighted bipartite graphs has an inapproximability ratio of 1.5, which implies the next theorem.

Theorem 13. *There is no polynomial-time* 1.5*-approximation algorithm for* p-DEL-MAX *on edge-weighted planar bipartite graphs unless $P = NP$.*

Proof. Consider an input planar bipartite graph G of MinMaxO. Add one new vertex u and one edge $\{u, v\}$ of weight $n \cdot w_{\max}$ to G for arbitrary $v \in V(G)$, where $w_{\max}$ is the maximum weight of edges in G. Let this new graph be G', where G' is also a planar bipartite graph. Observe that for 1-DEL-MAX, this new edge should be deleted since otherwise the maximum outdegree is at least $n \cdot w_{\max}$, which is larger than the total weight of edges incident to a vertex in G, and then we need to orient the edges in G optimally. Namely, if there exists a 1.5-approximation algorithm for 1-DEL-MAX, it also approximates MinMaxO within ratio 1.5. This contradicts the inapproximability of MinMaxO. This discussion can be extended to the case $p \geq 2$ by adding p new vertices and p new edges of weight $n \cdot w_{\max}$ to G. The theorem follows. □

The NP-hardness and known inapproximability bound of 2 for MaxMinO on edge-weighted planar bipartite graphs from [4] can be applied in the same way as for Theorem 12 and Theorem 13 to obtain the following theorem.

Theorem 14. *There is no polynomial-time $\rho(n)$ (or* 2*)-approximation algorithm for* MIN-INS-MIN (*or* p*-INS-MIN*) *on edge-weighted planar bipartite graphs unless $P = NP$, where $\rho(n) \geq 1$ is any polynomial-time computable function.*

Proof. The proof for MIN-INS-MIN is almost the same as the proof of Theorem 12 for MIN-DEL-MAX, but using MaxMinO instead of MinMaxO. As

for p-INS-MIN, we add independent p vertices to the input graph of MaxMinO, where this modified graph is also planar bipartite. This modification requires us to add p edges adjacent to those added p vertices, whose weights can be any value at least the optimal minimum outdegree of MaxMinO, e.g., $n \cdot w_{\max}$. By this modification to the input graph, solving p-INS-MIN is equivalent to solving MaxMinO for the original input graph, and so the inapproximability bound of MaxMinO is transferred to p-INS-MIN. □

For the other two problems (MAX-INS-MAX and MAX-DEL-MIN), we need to look a little deeper into the details of the NP-hardness proofs in [2, 4]. The next theorem shows the inapproximability of MAX-INS-MAX.

Theorem 15. *There is no polynomial-time $\rho(n)$-approximation algorithm for* MAX-INS-MAX *on edge-weighted planar bipartite graphs unless $P = NP$, where $\rho(n) \geq 1$ is any polynomial-time computable function.*

Proof. The NP-hardness of MinMaxO is proved in [4] for planar bipartite graphs having edge-weights 1 and $w \geq 2$ based on the intractability of distinguishing between the optimal value being w and the optimal value being $w+1$, where "+1" comes from the weight 1 of edges. We can double the edge-weights 1 and w to 2 and $2w$ in the reduced graphs, respectively, and observe that distinguishing between an optimal value of $2w$ and $2w+2$ is still intractable for them.

Consider a planar bipartite graph G having edge-weights 2 and $2w$ constructed according to the above, and let $k = 2w+1$. Let $OPT_1(G)$ and $OPT_2(G)$ denote the optimal costs of MinMaxO and the maximum number of inserted edges of MAX-INS-MAX for G, respectively. If $OPT_1(G) = 2w$ for MinMaxO, then we can add one edge of weight one to G between an arbitrary pair of vertices of G, increasing the maximum outdegree of vertices to at most $2w+1$, i.e., $OPT_2(G) > 0$ for MAX-INS-MAX. On the other hand, if $OPT_1(G) = 2w+2$ for MinMaxO, no edge can be added under the condition that the maximum outdegree is at most k, i.e., $OPT_2(G) = 0$ for MAX-INS-MAX. Thus, $OPT_1(G) = 2w$ for MinMaxO if and only if $OPT_2(G) > 0$ for MAX-INS-MAX. Then, as in the proof of Theorem 12, the assumption that there exists a polynomial-time $\rho(n)$-approximation algorithm for some polynomial-time computable function $\rho(n) \geq 1$ for MAX-INS-MAX on edge-weighted graphs leads to a contradiction to the NP-hardness of MinMaxO. □

The inapproximability of MAX-DEL-MIN is established in the next theorem.

Theorem 16. *There is no polynomial-time $\rho(n)$-approximation algorithm for* MAX-DEL-MIN *on edge-weighted planar bipartite graphs unless $P = NP$, where $\rho(n) \geq 1$ is any polynomial-time computable function.*

Proof. The NP-hardness of MaxMinO for edge-weighted planar bipartite graphs is proved in [2] with edge-weights $w_{\min}$ and $w_{\max}$ such that $w_{\min} < w_{\max}$, considering to distinguish between $w_{\min}$ and $\min\{2w_{min}, w_{\max}\}$ of optimal values. Here we choose $w_{\min} = 2$ and $w_{\max} = 4$. Assume that the set of vertices in an input graph G of MaxMinO is partitioned into two partite sets U_1 and U_2. We add an extra vertex u and two extra edges $e_1 = \{u, v_1\}$ of weight 1 and $e_4 = \{u, v_4\}$ of weight 4 between u and two vertices v_1 and v_4 in U_1. Let this new planar bipartite graph (an instance of MAX-DEL-MIN) be G'. Then we set $k = 4$ for MAX-DEL-MIN. We observe that the outdegree of u is at least 4 under an orientation of G' if and only if e_4 is not deleted and oriented outward from u. Then, e_1 increases any outdegree of vertices by at most 1 and so this edge is a candidate for deletion.

Let $OPT_1(G)$ denote the optimal cost of MaxMinO for G. Also let $OPT_2(G')$ denote the maximum number of deleted edges of MAX-DEL-MIN for G'. If $OPT_1(G) \geq 4$ for MaxMinO, then we can delete e_1 from G' for MAX-DEL-MIN, by which the resulted graph has an orientation in which the minimum outdegree of vertices is at least 4 by orienting e_4 as (u, v_4) with the optimal orientation of G for MaxMinO. Thus $OPT_2(G') \geq 1$. As another case, assume that $OPT_1(G) \leq 3$ for MaxMinO. This assumption implies that there exists a vertex of outdegree at most 3 in the optimal orientation of G for MaxMinO. If we delete e_1 from G' for MAX-DEL-MIN, then the minimum outdegree of the vertices is the same as $OPT_3(G)$, i.e., at most 3 which does not satisfy the value of k, since e_4 must be oriented as (u, v_4) to increase u's outdegree to 4 (otherwise its outdegree becomes 0). Hence we consider to keep e_1 and delete one edge between U_1 and U_2. However, since $OPT_1(G) \leq 3$, there exists a vertex of outdegree at most 3 under any orientation. This implies that the outdegree of such a vertex is 0 or 2, since the weight of an edge of G is either 2 or 4. Even if v_1 is only such a vertex and e_1 is oriented as (v_1, u) (increasing v_1's outdegree by one), there still remains a vertex with outdegree at most 3. Namely, we cannot delete any edge from G' which means $OPT_2(G') = 0$. In summary, $OPT_1(G) = 4$ for MaxMinO if and only if $OPT_2(G') > 0$ for MAX-DEL-MIN with setting $k = 4$. Then, as in the proof of the previous theorems, the assumption that there exists a polynomial-time $\rho(n)$-approximation algorithm for some polynomial-time computable function $\rho(n) \geq 1$ for MAX-DEL-MIN on edge-weighted graphs leads to a contradiction to the NP-hardness of MaxMinO. □

Also for p-INS-MAX and p-DEL-MIN, we can show the same inapproximability as p-DEL-MAX and p-INS-MIN, respectively.

Theorem 17. *There is no polynomial-time* 1.5 (*or* 2)*-approximation algorithm for* p-INS-MAX (*or* p-DEL-MIN) *on edge-weighted planar bipartite graphs unless* $P = NP$.

Proof. Figure 5 illustrates examples of the two reductions in this proof for the case $p = 2$.

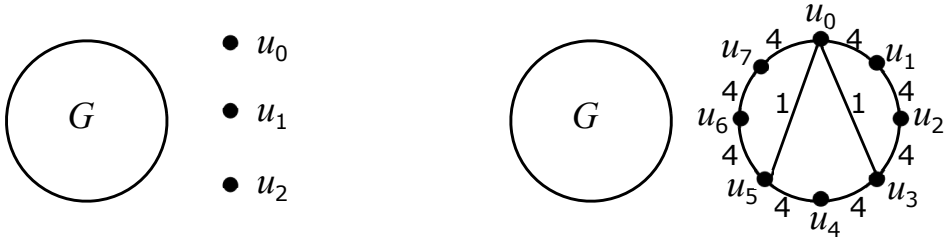

Fig. 5. Example graphs constructed by the reductions in the proof of Theorem 17. The left graph is for 2-INS-MAX and the right one is for 2-DEL-MIN.

The reduction for p-INS-MAX is done from MinMaxO. We add $p+1$ independent vertices $u_0, \ldots, u_p$ to the input planar bipartite graph G of MinMaxO. An optimal set of inserted edges for p-INS-MAX includes edges $\{u_i, u_{i+1}\}$ of weight 1 for $0 \leq i \leq p-1$, which constitutes a path of those $p+1$ vertices $u_0, \ldots, u_p$. Let the optimal orientation of G for MinMaxO be Λ. The above modification to G does not increase the maximum outdegree of the vertices of G under Λ. Hence, solving p-INS-MAX for this new graph is equivalent to solving MinMaxO for G, which gives the inapproximability of p-INS-MAX based on the inapproximability of MinMaxO.

For p-DEL-MIN, the reduction is from MaxMinO. Let us consider an input planar bipartite graph G to MaxMinO with edge-weights 2 and 4, considering to distinguish between 2 and 4 of optimal values, as in the proof of Theorem 16. We construct an instance of p-DEL-MIN by adding new $2p+4$ vertices $u_0, u_1, \ldots, u_{2p+3}$ to G. Among those vertices, we add edges such that

- for each i, $0 \leq i \leq 2p+3$, $\{u_i, u_{(i+1) \mod (2p+4)}\}$ of weight 4, i.e., the $2p+4$ vertices and these edges form a cycle of length $2p+4$, and
- for each j, $2 \leq j \leq p+1$, $\{u_0, u_{2j-1}\}$ of weight 1, where these edges form a star with p leaves rooted at u_0.

Clearly the constructed graph is still planar bipartite. An optimal set of deleted edges for p-DEL-MIN includes the set of added edges of weight 1, since the added cycle constituted by the edges of weight 4, can have minimum outdegree 4 by orienting the edges in, say, clockwise direction. Then, again, solving p-DEL-MIN for this modified graph is equivalent to solving MaxMinO for G, considering to distinguish between 2 and 4 of optimal values, which gives the inapproximability of p-DEL-MIN based on the inapproximability of MaxMinO. □

4. Concluding Remarks

We have introduced eight new graph orientation problems whose objective is to minimize/maximize the outdegree of the vertices after inserting/deleting edges, and presented polynomial-time algorithms for these problems on unweighted graphs.

Also we showed the polynomial-time inapproximability for those problems on edge-weighted graphs, and polynomial-time algorithms for six of the problems were designed on edge-weighted trees. One of the further research topics is to study the complexity of MAX-INS-MAX and p-INS-MAX on edge-weighted trees.

Very recently, Frank and Murota [12] proposed a strongly polynomial-time algorithm to find a maximum flow which is decreasingly minimal on a specified subset F of directed edges in a given network. Here, a flow is *decreasing minimal on* F if the largest flow on an edge in F is as small as possible, within this, the second largest flow on an edge in F is as small as possible, and so on. Also, in the given network, upper and lower bounds on the size of a flow passing through each edge are prescribed. By appropriately setting the upper and lower bounds for the edges, and choosing $F = \{(s, v) \mid v \in V\}$ from N_G, we can utilize their algorithm to obtain a decreasingly minimal maximum flow on F in the algorithms of Theorems 3 and 5. After modifying Lemmas 1 and 2 accordingly, a *decreasingly minimal orientation* (also called a *lexicographic orientation* in [7]) for our problems can thus be constructed, meaning that the maximum outdegree of a vertex is as small as possible, and among all such orientations, the second largest outdegree of a vertex is as small as possible, etc.

One generalization of the problems is to extend the input to hypergraphs. In Sec. 2, we constructed a network $N_G(\alpha; \beta; \gamma)$ from an input graph G and solved the maximum flow problem on it by using a polynomial-time algorithm. In $N_G(\alpha; \beta; \gamma)$, we prepare an edge (v, e) if a vertex v is incident to an edge e in G so that the vertex e in $N_G(\alpha; \beta; \gamma)$ has two incoming edges. If the input G is a hypergraph, a vertex in $N_G(\alpha; \beta; \gamma)$ corresponding to an hyperedge in G may have at most n incoming edges. Thus, $N_G(\alpha; \beta; \gamma)$ has at most nm edges between V and E, where n and m are the number of vertices and edges in G, respectively. The running times of the algorithms using $N_H(\alpha; \gamma)$ (a part of $N_G(\alpha; \beta; \gamma)$) in Theorem 3 for MIN-DEL-MAX, p-DEL-MAX, MAX-DEL-MIN, and p-DEL-MIN increase to $O(nm^2 \log m)$, which is still polynomial. On the other hand, the algorithms in Theorem 5 do not run in polynomial time in the worst case since they use the whole $N_G(\alpha; \beta; \gamma)$ and the size of $\overline{E}$ in $N_G(\alpha; \beta; \gamma)$ might not be polynomial in n and m.

Another natural generalization of MIN-DEL-MAX can be defined as follows:

Input: An unweighted graph $G = (V, E)$ and a mapping μ that assigns to each vertex $v \in V$ an integer from $\{0, 1, \ldots, \deg(v)\}$, where $deg(v)$ is the degree of v.
Goal: To find the minimum number of edges to delete to get a spanning subgraph H of G such that $d^+(v) \leq \mu(v)$ for every $v \in V$.

To solve this problem, we can first construct a directed graph N_H in the same way as in Sec. 2. Next, we augment the edge capacities in N_H to get a flow network as before, except for the capacities of the directed edges of the form $\{(s, v) \mid v \in V\}$. The capacity of the directed edge (s, v) is defined to be $\mu(v)$ instead of k, for every $v \in V$. Then we see that there exists a flow in N_H with value $|E|$ if and only if there are p edges in G whose deletion leaves a graph satisfying $d^+(v) \leq \mu(v)$, for

every vertex $v \in V(G)$. In other words, the modified problem can be solved in polynomial time. In fact, both the modified problem and the original MIN-DEL-MAX have the same time complexity since the directed graphs that we construct in both cases are the same. We can generalize MIN-INS-MIN, MAX-INS-MAX, MAX-DEL-MIN in the same way and solve them in polynomial time as well.

The above problem is a special case of General Factor introduced by Lovász [13, 14], which is defined as

Input: An unweighted graph $G = (V, E)$ and a mapping K that assigns to each vertex $v \in V$ a set $K(v) \subseteq \{0, 1, \ldots, \deg(v)\}$ of integers.
Goal: To check if there is a subgraph H of G s.t. $d_H(v) \in K(v)$ for every $v \in V$, where $d_H(v)$ is the degree of v in H.

General Factor is a generalization of the factor problem, and NP-hard even for unweighted graphs [14], and also for unweighted planar bipartite graphs [9]. Here the extension to the mapping from an integer to a set of integers makes the problem harder. We conjecture that the analogous generalizations to the problems in this paper are NP-hard.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Numbers JP17K00016 and JP17K00024, JST CREST JPMJR1402, and PolyU Fund 1-ZE8L.

References

[1] Y. Asahiro, J. Jansson, E. Miyano, H. Nikpey and H. Ono, Graph orientation with splits, In *Proceedings of ISCO 2018*, Volume 10856 of Lecture Notes in Computer Science (Springer, 2018), pp. 52–63.
[2] Y. Asahiro, J. Jansson, E. Miyano and H. Ono, Graph orientation to maximize the minimum weighted outdegree, *International Journal of Foundations of Computer Science* **22**(3) (2011) 583–601.
[3] Y. Asahiro, J. Jansson, E. Miyano and H. Ono, Graph orientations optimizing the number of light or heavy vertices, *Journal of Graph Algorithms and Applications* **19**(1) (2015) 441–465.
[4] Y. Asahiro, J. Jansson, E. Miyano, H. Ono and K. Zenmyo, Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree, *J. Combinatorial Optimization* **22**(1) (2011) 78–96.
[5] Y. Asahiro, E. Miyano and H. Ono, Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree, *Discrete Applied Mathematics* **159** (2011) 498–508.
[6] N. Bansal, A. Blum and S. Chawla, Correlation clustering, *Machine Learning* **56** (2004) 89–113.
[7] G. Borradaile, J. Iglesias, T. Migler, A. Ochoa, G. Wilfong and L. Zhang, Egalitarian graph orientations, *Journal of Graph Algorithms and Applications* **21**(4) (2017) 687–708.

[8] G. S. Brodal and R. Fagerberg, Dynamic representations of sparse graphs, In *Proceedings of WADS 1999*, Volume 1663 of Lecture Notes in Computer Science (Springer, 1999), pp. 342–351.

[9] G. Cornuéjols, General factors of graphs, *Journal of Combinatorial Theory, Series B* **45** (1988) 185–198.

[10] M. Chrobak and D. Eppstein, Planar orientations with low out-degree and compaction of adjacency matrices, *Theoretical Computer Science* **86**(2) (1991) 243–266.

[11] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithms*, 3rd edition (The MIT Press, 2009).

[12] A. Frank and K. Murota, Fair integral flows, 2020, arXiv:1907.02673, https://arxiv.org/abs/1907.02673.

[13] L. Lovász, The factorization of graphs, In *Combinatorial Structures and Their Applications* (1970) 243–246.

[14] L. Lovász, The factorization of graphs. II, *Acta Mathematica Academiae Scientiarum Hungaricae* **23** (1972) 223–246.

[15] J. B. Orlin, Max flows in $O(nm)$ time, or better, *Proceedings of the* 45^{th} *Annual ACM Symposium on Theory of Computing (STOC'13)*, Association for Computing Machinery (ACM) (Palo Alto, CA, USA, 2013), 765–774.

[16] R. Sharan, Graph modification problems and their applications to genomic research, PhD Thesis. School of Computer Science, Tel-Aviv University, 2002.

[17] V. Venkateswaran, Minimizing maximum indegree, *Discrete Applied Mathematics* **143** (2004) 374–378.