Theoretical Computer Science 900 (2022) 53–78

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Upper and lower degree-constrained graph orientation with minimum penalty ☆

Yuichi Asahiro [a,*], Jesper Jansson [b], Eiji Miyano [c], Hirotaka Ono [d]

[a] Department of Information Science, Kyushu Sangyo University, Fukuoka 813-8503, Japan
[b] Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
[c] Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, Fukuoka 820-8502, Japan
[d] Graduate School of Informatics, Nagoya University, Nagoya 464-8601, Japan

ARTICLE INFO

Article history:
Received 28 January 2021
Received in revised form 15 November 2021
Accepted 27 November 2021
Available online 2 December 2021
Communicated by A. Casteigts

Keywords:
Graph orientation
Degree constraint
Penalty function
Inapproximability
Tree
Treewidth

ABSTRACT

In the *degree-constrained graph orientation problem*, the input is an unweighted, undirected graph $G = (V, E)$ and nonnegative integers a_v and b_v (with $a_v \leq b_v$) for each $v \in V$, and the objective is to assign a direction to every edge in E in such a way that for each $v \in V$, the number of outgoing edges in the resulting directed graph lies in the interval $[a_v, b_v]$. When such an orientation does not exist, it is desirable to find an orientation that best fits the condition instead. In this paper, we consider the problem of finding an orientation that minimizes the total penalty $\sum_{v \in V} c_v$, where c_v is a penalty incurred whenever a vertex v violates its degree constraints. As penalty functions, convex, concave, and step functions are considered in this paper. We show that the problem with any convex penalty function can be solved in $O(|E|^{1.5} \min\{\log(|E| \cdot C), |E|^{0.5} \log \Delta \log |E|\})$ time, where Δ and C are the maximum degree and the largest magnitude of a penalty, respectively. In contrast, we show APX-hardness of the problem with step or concave functions. For trees and graphs with treewidth τ, the problem with any penalty function can be solved exactly in $O(|V| \log \Delta)$ time and $O(\tau^2 \Delta^{2\tau+2} |V|)$ time, respectively. Finally, we consider the generalization of the problem to edge-weighted graphs and establish strong bounds on its inapproximability that hold even in the special case of stars. On the positive side, we can extend our algorithms for unweighted version of the problem to obtain pseudo-polynomial-time algorithms for the edge-weighted problem variant when restricted to trees and graphs with bounded treewidth. Also, we design a PTAS and a linear-time algorithm for stars with further restrictions on the degree constraints and edge weights.

1. Introduction

1.1. Notation and problem definition

Let $G = (V, E)$ be an unweighted undirected graph, where V and E denote the sets of vertices and edges, respectively. The vertex set and the edge set of G is also denoted by $V(G)$ and $E(G)$. We allow G to have parallel edges; G is possibly a

☆ A part of the results in this paper appeared in Proceedings of Computing: The Australasian Theory Symposium (CATS 2012), Conferences in Research and Practice in Information Technology, Vol. 128, pp. 139–146, 2012.

* Corresponding author.
E-mail addresses: asahiro@is.kyusan-u.ac.jp (Y. Asahiro), jj@i.kyoto-u.ac.jp (J. Jansson), miyano@ai.kyutech.ac.jp (E. Miyano), ono@i.nagoya-u.ac.jp (H. Ono).

https://doi.org/10.1016/j.tcs.2021.11.019
0304-3975/

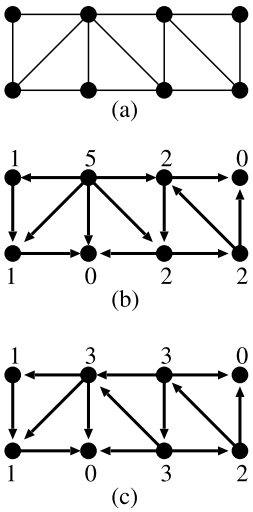

Fig. 1. Example of MPDCO$_p$: (a) An undirected graph $G = (V, E)$; (b) an orientation Λ of G with outdegree sequence $(1, 1, 5, 0, 2, 2, 0, 2)$ from the left top to the right bottom; (c) an orientation Λ' of G with outdegree sequence $(1, 1, 3, 0, 3, 3, 0, 2)$.

multi-graph. Throughout the paper, let $|V| = n$ and $|E| = m$ for the graph. Two vertices u and v are called *adjacent* to each other if $\{u, v\} \in E$. Let $N_G(u)$ be the set of adjacent vertices of u in G, i.e., $N_G(u) = \{v \mid \{u, v\} \in E\}$, and $d_G(u) = |N_G(u)|$ is called *degree* of u in G. We omit the subscripts G's of $N_G(u)$ and $d_G(u)$ if it is clear from the context; sometimes we use $N(u)$ and $d(u)$ instead of $N_G(u)$ and $d_G(u)$. We denote $\max\{d(v) \mid v \in V\}$ by Δ. An *orientation* Λ *of* G is a set of an assignment of a direction to each edge $\{u, v\} \in E$, i.e., $\Lambda(\{u, v\}) = (u, v)$ or (v, u). We simply use Λ to represent $\Lambda(E) = \bigcup_{e \in E}\{\Lambda(e)\}$ if no confusion arises. We denote the directed graph obtained from G and Λ by $\Lambda(G) = (V, \Lambda(E))$. The *outdegree* of a vertex u in $\Lambda(G)$ is $|\{(u, v) \mid (u, v) \in \Lambda\}|$, which is denoted by $d^+_G(\Lambda, u)$.

Assume that $V = \{v_1, v_2, \ldots, v_n\}$ and that a sequence of $2n$ nonnegative integers a_{v_i} and b_{v_i} satisfying $a_{v_i} \le b_{v_i}$ for $1 \le i \le n$ is given. A *degree-constrained orientation* is an orientation Λ of G such that $a_{v_i} \le d^+_G(\Lambda, v_i) \le b_{v_i}$ holds for every $v_i \in V$. Obviously, G does not always have a degree-constrained orientation. In such a case, we would like to find an orientation that "best" fits the degree constraint.

For a graph $G = (V, E)$, an orientation Λ of G, and a vertex $v_i \in V$, we define a cost function c_G such that $c_G(\Lambda, v_i) = d^+_G(\Lambda, v_i) - b_{v_i}$ if $d^+_G(\Lambda, v_i) > b_{v_i}$, $c_G(\Lambda, v_i) = a_{v_i} - d^+_G(\Lambda, v_i)$ if $d^+_G(\Lambda, v_i) < a_{v_i}$, and $c_G(\Lambda, v_i) = 0$ otherwise. Then the *violation vector* $c_G(\Lambda)$ for a graph G and its orientation Λ is defined as $(c_G(\Lambda, v_1), c_G(\Lambda, v_2), \ldots, c_G(\Lambda, v_n))$. For a vertex v_i with outdegree k under some orientation, we sometimes denote its *cost* by $\theta_{v_i}(k)$, that is, $\theta_{v_i}(k) = k - b_{v_i}$ if $k > b_{v_i}$, $\theta_v(k) = a_{v_i} - k$ if $k < a_{v_i}$, $\theta_{v_i}(k) = 0$ otherwise. The purpose of defining $\theta_{v_i}(k)$ is to estimate the cost on a vertex v_i only depending on its outdegree regardless of orientations. For an orientation Λ of G, it holds that $\theta_{v_i}(d^+_G(\Lambda, v_i)) = c_G(\Lambda, v_i)$. A *penalty function* p is a finite, nonnegative, and nondecreasing function with n variables. By using these, we define the *best-fit* orientation to the degree constraint by an orientation Λ of G that minimizes *the total penalty* $p(c_G(\Lambda))$ *(of* Λ*)*

Given an input graph $G = (V, E)$ and two nonnegative integers a_v and b_v for each $v \in V$ satisfying $a_v \le b_v$, the Minimum Penalty Degree-Constrained Orientation problem with a fixed penalty function p (MPDCO$_p$ for short) asks for the best-fit orientation of G.

As an example of an instance of MPDCO$_p$, see the undirected graph $G = (V, E)$ in Fig. 1-(a). Let $a_v = 1$ and $b_v = 2$ for any $v \in V$. Figs. 1-(b) and (c) are two directed graphs obtained by orientations Λ and Λ', respectively. We first consider the case when p is a summation of a convex function $g_1(x) = x^2$ for each vertex, i.e., $p(c_G(\Lambda)) = \sum_{v \in V} g_1(c_G(\Lambda, v))$. By the orientation Λ of Fig. 1-(b), the outdegree sequence of the vertices is $(1, 1, 5, 0, 2, 2, 0, 2)$ in the column-major order from the left to the right. Thus its violation vector $c_G(\Lambda)$ is $(0, 0, 3, 1, 0, 0, 1, 0)$ and so the total penalty $p(c_G(\Lambda))$ is $\sum_{v \in V} g_1(c_G(\Lambda, v)) = 3^2 + 1^2 + 1^2 = 11$. On the other hand, by the orientation Λ' in Fig. 1-(c), $c_G(\Lambda') = (0, 0, 1, 1, 1, 1, 1, 0)$ and $p(c_G(\Lambda')) = \sum_{v \in V} g_1(c_G(\Lambda', v)) = 1^2 + 1^2 + 1^2 + 1^2 + 1^2 = 5$.

As another example, consider the case that p is a summation of the following concave function $g_2(x)$ instead of $g_1(x)$:

$$g_2(x) = \begin{cases} x & \text{if } 0 \le x \le 1, \\ 1 & \text{if } x > 1. \end{cases}$$

Then, the total penalty $p(c_G(\Lambda))$ of Λ is $\sum_{v \in V} g_2(c_G(\Lambda, v)) = 1 + 1 + 1 = 3$, and the total penalty $p(c_G(\Lambda'))$ of Λ' is $\sum_{v \in V} g_2(c_G(\Lambda', v)) = 1 + 1 + 1 + 1 + 1 = 5$. The "balanced" orientation Λ' in Fig. 1-(c) is worse than the "unbalanced" orientation Λ in Fig. 1-(b) for the pair of G and g_2. Also, note that the total penalty highly depends on the values of a_v and b_v.

In this paper, we assume that all penalty functions are *linearly separable*, i.e., any penalty function can be written as $p(c_G(\Lambda)) = \sum_{v \in V} g(c_G(\Lambda, v))$ for some nonnegative nondecreasing one-variable function g with $g(0) = 0$. The function g is called the *separated penalty function* (of the penalty function p). Namely, the *penalty* on each vertex v under an orientation Λ is represented by $g(c_G(\Lambda, v))$. Throughout the paper, we assume that g is a fixed function, and can be evaluated in constant time; for a nonnegative real x, the value of $g(x)$ is obtained in $O(1)$ time. In this setting, we focus on the following types of functions as typical examples of g: convex, concave, and step functions. A function g is *convex* or a *convex function* if $g(\alpha x_1 + (1-\alpha)x_2) \leq \alpha g(x_1) + (1-\alpha)g(x_2)$ for any x_1, x_2, and α satisfying $0 \leq \alpha \leq 1$. Similarly, a function g is *concave* or a *concave function* if $g(\alpha x_1 + (1-\alpha)x_2) \geq \alpha g(x_1) + (1-\alpha)g(x_2)$ for any x_1, x_2, and α satisfying $0 \leq \alpha \leq 1$. Also, a function g is a *step function* if it can be written as $g(x) = \sum_{i=0}^{k} \alpha_i \chi_{A_i}(x)$, where $k \geq 0$, α_i's are real numbers, A_i's are intervals, and χ_A is the indicator function of an interval A such that $\chi_A(x) = 1$ if $x \in A$, and $\chi_A(x) = 0$ otherwise. We say that a penalty function p is *linearly separable with a convex (concave, or step) function (g)* if p is linearly separable and the separated penalty function of p is a convex (concave, or step) function (g). If the penalty function p of MPDCO$_p$ is linearly separable with a (convex, concave, or step) function (g), we simply indicate it by writing "MPDCO$_p$ with a (convex, concave, or step) function (g)."

An algorithm for MPDCO$_p$ is an *α-approximation algorithm*, or its *approximation ratio* is α, if it holds that $p(c_G(\Lambda))/p(c_G(\Lambda^*)) \leq \alpha$ for any input graph G, where Λ and Λ^* are orientations obtained by the algorithm and an optimal algorithm, respectively. Then, an algorithm for MPDCO$_p$ is a *polynomial-time approximation scheme (PTAS)* if it takes a parameter ϵ and a graph G as input, then outputs an orientation Λ such that $p(c_G(\Lambda))/p(c_G(\Lambda^*)) \leq 1 + \epsilon$ with running time which is polynomial in n and m for a fixed ϵ. Also, an algorithm for MPDCO$_p$ is a *fixed parameter tractable (FPT) algorithm* parameterized by a parameter if it runs in $O(f(k) \cdot p(n, m))$ time, where $f(k)$ is a function only depending on the size k of the parameter and $p(n, m)$ is polynomial in n and m.

1.2. Summary of results

In this paper, we study the relationship between the computational complexity of MPDCO$_p$ and linearly separable penalty functions. Clearly, the nature of MPDCO$_p$ depends on the penalty function p.

The results in this paper are summarized as follows:

- MPDCO$_p$ with a convex function g can be solved in $O(m^{1.5} \min\{\log m g(\Delta), m^{0.5} \log \Delta \log m\})$ time (Theorem 2 in Section 2). Hence, if g is a polynomial function, such as $g(x) = x^k$ with a positive constant k, the running time is $O(m^{1.5} \log m)$.
- MPDCO$_p$ with a step (or concave) function has no polynomial-time approximation algorithm whose approximation ratio is better than $\sqrt{2} - o(1)$ unless P=NP; it is APX-hard (Theorem 1 in Section 1.3).
- For trees, MPDCO$_p$ with any function g can be solved in $O(n \log \Delta)$ time (Theorem 3 in Section 3.1.1). Also, for some restricted functions, MPDCO$_p$ on trees can be solved in $O(n)$ time (Theorem 4 in Section 3.1.2). This running time is attainable for any function g as long as basic arithmetic operations can be done in $O(1)$ time; g does not have to be convex or concave. In addition, for graphs with treewidth τ, MPDCO$_p$ with any function can be solved in $O(\tau^2 \Delta^{2\tau+2} n)$ time (Theorem 9 in Section 3.2). This algorithm is an FPT algorithm parameterized by treewidth plus the maximum degree. On the other hand, we show W[1]-hardness of MPDCO$_p$ with a step (or concave) function parameterized by treewidth only (Theorem 10 in Section 3.3).
- For the edge-weighted version of MPDCO$_p$ with a convex, concave, or step function, there is no polynomial-time $\rho(n)$-approximation algorithm, where $\rho(n) \geq 1$ is any polynomial-time computable function (Corollary 14 in Section 4.1 and Corollary 18) in Section 4.3. The inapproximability results are shown based on the strong NP-hardness of the problem for edge-weighted planar bipartite graphs and the weak NP-harness for edge-weighted stars (Theorem 13 in Section 4.1 and Theorem 17 in Section 4.3).
- For the edge-weighted version of MPDCO$_p$, we extend the polynomial-time algorithm in Sections 3.1 and the FPT algorithm in 3.2 (Theorems 15 and 16 in Section 4.2). In addition, we design polynomial-time (approximation) algorithms for edge-weighted stars (Theorems 21 and 22 in Section 4.3).

1.3. Related work

Graph orientation itself is a fundamental problem in the area of graph theory and combinatorial optimization. In general, graph orientation is a problem of finding an orientation to a given undirected graph to meet some given requirement. Various kinds of requirements such as connectivity, reachability, acyclicity, etc. have been considered [1–3], and in particular, there is a large literature devoted to graph orientation with degree constraints; see, e.g., Sections 61.1 in [4], 7.4.3 in [5], and 2.3 in [6].

Many fundamental graph problems such as graph routing, matching, and covering can be formalized as degree-constrained graph orientations. One of the earliest related results was published in 1953 by Landau [7], who proved a

necessary and sufficient condition on the sum of the outdegrees of any subset of vertices in a directed complete graph.[1] Hakimi [8], Frank and Gyárfás [9], Chrobak and Eppstein [10], and Gabow [11] studied a variant of the degree-constrained orientation problem in which the goal is to orient as many edges as possible in an undirected graph, subject to the upper and lower bounds on the outdegree of each vertex (or equivalently, the upper bounds on the indegree and outdegree of each vertex). In [8], Hakimi gave the necessary and sufficient conditions of graphs that can be oriented so that every outdegree is at most a given upper bound. These were generalized by Frank and Gyárfás in [9] to a characterization of graphs that can be oriented so that every outdegree is between given upper and lower bounds.

In [10], Chrobak and Eppstein studied orientations of planar graphs and showed that an orientation with maximum outdegree 3 as well as an acyclic orientation with maximum outdegree 5 can be obtained in linear time for any planar graph. In another line of research, Gabow considered the *partial orientation* problem in [11], which formulates the degree-constrained orientation problem as an optimization problem. A partial orientation assigns a unique direction to a subset of the edges, leaving the remaining edges unoriented. Then, the goal is to orient as many edges as possible in the input undirected graph without breaking the degree constraints. He proved that the partial orientation problem is MAXSNP-hard and provided an LP-rounding algorithm which achieves approximation ratio 3/4.

In situations where it is impossible to orient the edges of a given graph so that all of the specified degree constraints are satisfied at the same time, one may still need a reasonably good orientation. MPDCO$_p$ resolves this issue by interpreting the degree constraints as *soft constraints* that may be violated, and returning an orientation that minimizes the total penalty charged for the violated constraints. It should be noted that MPDCO$_p$ is a natural generalization of several optimization problems to control the outdegrees of an undirected graph. For example, the Minimum Maximum Outdegree Orientation problem (MinMaxO for short) is to find an orientation of an input graph G minimizing $\max_{u\in V}\{d^+_G(\Lambda,u)\}$ (e.g., [12–15]). MinMaxO can be used in efficient dynamic data structures for graphs that support fast vertex adjacency queries under a series of edge operations [16]. Furthermore, MinMaxO is a special case of the Minimum Makespan problem (e.g., [17]). MinMaxO can be expressed as MPDCO$_p$ with a convex function $g(x)=\ell^x$ for an $\ell > n$, e.g., $\ell = 2n$ $(< m)$ by setting $a_v = b_v = 0$ for every $v \in V(G)$ in the input graph G, which defines the total penalty of an orientation Λ as $p(c_G(\Lambda)) = \sum_{i=1}^{n} \ell^{c_G(\Lambda,v)}$, where $g(\Delta) = O(m^{\Delta})$. As mentioned in the previous section, MPDCO$_p$ with a convex function $g(x)$ can be solved in $O(m^{1.5}\min\{\log(mg(\Delta)), m^{0.5}\log\Delta\log m\})$ time, i.e., MinMaxO can be solved in $O(m^{1.5}\min\{\Delta, m^{0.5}\log\Delta\}\log m)$ time. This result can be compared to the result that MinMaxO on unweighted graphs can be solved in $O(m^{1.5}\log\Delta)$ time [13].

Finally, we will discuss a related problem called Min W-Heavy and show how the known results for it immediately imply the APX-hardness of MPDCO$_p$ with a step function. For an input undirected graph and a positive number W, the problem Min W-Heavy is to find an orientation such that the number of vertices of outdegree at least W is minimized in the resulted directed graph. Min W-Heavy is APX-hard for any fixed W [18]. By a reduction from Min 1-Heavy, the APX-hardness of MPDCO$_p$ with a step function is straightforwardly obtained by letting

$$g(x) = \begin{cases} 0 & \text{if } x = 0 \text{ and} \\ 1 & \text{if } x > 0. \end{cases}$$

Note that the above function g is also a concave function. We observe that the minimum total penalty for MPDCO$_p$ with the above g is obtained by minimizing the number of vertices of outdegree at least one under an orientation, which is equivalent to solving Min 1-Heavy. Then the APX-hardness of MPDCO$_p$ with a step function (or a concave function) follows directly from the APX-hardness of Min 1-Heavy. It is shown that Min-1-Heavy cannot be approximated within a ratio of 1.3606 unless $P = NP$ [18] based on the lower bound 1.3606 of polynomial-time approximation ratio of Minimum Vertex Cover [19]. Recently, this lower bound has been increased to $\sqrt{2} - o(1)$ [20,21] and hence we have the following theorem.

Theorem 1. *For MPDCO$_p$ with a step function or a concave function, there is no polynomial-time algorithm whose approximation ratio is better than $\sqrt{2} - o(1)$, unless $P = NP$.*

Also note that the problem Min W-Heavy for $W \geq 2$ can be viewed as MPDCO$_p$ with a step function g such that $g(x) = 1$ if $x \geq W$ and $g(x) = 0$ otherwise.

1.4. Organization of the paper

The remainder of the paper is organized as follows. In Section 2, we present a polynomial-time algorithm for MPDCO$_p$ with convex functions. Section 3 presents a polynomial-time algorithm for trees and an FPT algorithm parameterized by treewidth plus the maximum degree for MPDCO$_p$ with general functions. In Section 4, we consider the edge-weighted version of the problem. Section 5 discusses further research on the problems and concludes the paper.

[1] For an undirected unweighted complete graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, let $\mathcal{V}_k = \{V' \subseteq V \mid |V'| = k\}$ for $1 \leq k \leq n$. Also let $I(V') = \{i \mid v_i \in V'\}$ for $V' \in \mathcal{V}_k$. Then, for $V' \in \mathcal{V}_k$ and any sequence of integers $(d_1, d_2, \ldots, d_n)$ such that $\sum_{1\leq i\leq n} d_i = n(n-1)/2$, it holds that $\sum_{i\in I(V')} d_i \geq k(k-1)/2$ if and only if there is an orientation Λ of G such that $d_i = d^+_G(\Lambda, v_i)$ for $1 \leq i \leq n$.

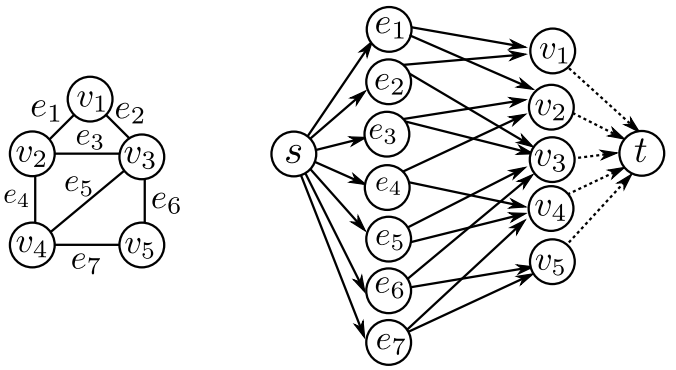

Fig. 2. Network construction in the proof of Theorem 2: (Left) An undirected graph G; (Right) the network $\mathcal{N}$ constructed from G.

2. Polynomial-time algorithms for convex functions

In this section, we present a simple approach to solve MPDCO$_p$ with a convex function g. For simplicity, MPDCO$_{convex}$ represents this variant of the problem. The approach is based on the reduction to the Convex Cost Flow problem (CCF for short). Using a strongly polynomial-time algorithm for CCF, the approach for MPDCO$_{convex}$ runs in $O(m^{1.5}\min\{\log(mg(\Delta)), m^{0.5}\log\Delta\log m\})$ time.

In general, a network flow problem is the problem of finding some optimal flow on a given network that satisfies capacity constraints on arcs (directed edges) and supply/demand conditions on vertices, and many types of network flow problems are intensively and extensively studied. See [22]. Among them, CCF is an optimization problem on a network in which each arc is associated with a convex function whose variable is flow through the arc. The cost of flow is the total sum of flow costs on the arcs, and the convex cost flow problem is the problem of finding a flow whose cost is minimum. If the convex functions are just linear functions, the problem is simply called the Minimum Cost Flow problem, which is a well studied problem. It is known that CCF can be solved in $O(NM\log(N^2/M)\log(NU))$ time, where N, M and U are the number of vertices, the number of edges, and the largest magnitude of the lower and upper bounds on capacities in the network, respectively [23]. Fortunately, our problem belongs to a relatively smaller class of the flow problems; the capacity of every arc is integral. For this class, faster algorithms by [22] and [24] can be applied, which yields the next theorem.

Theorem 2. *MPDCO$_{convex}$ can be solved in $O(m^{1.5}\min\{\log(mg(\Delta)),\ m^{0.5}\log\Delta\log m\})$ time.*

Proof. From graph $G=(V,E)$, sequence of $2n$ integers $(a_1,a_2,\ldots,a_n)$ and $(b_1,b_2,\ldots,b_n)$, and convex function g, we construct the following network $\mathcal{N}=(V_{\mathcal{N}},E_{\mathcal{N}})$:

$$V_{\mathcal{N}} = V \cup E \cup \{s,t\} \text{ and}$$
$$E_{\mathcal{N}} = \bigcup_{e=\{u,v\}\in E} \{(s,e),(e,u),(e,v)\} \cup E_t,$$

where $E_t=\bigcup_{v\in V}\{(v,t)\}$. The capacity of an arc (v,t) in E_t is defined by $\mathrm{cap}((v,t))=d_G(v)$, and those of the other arcs are 1. The supply of source s and the demand of sink t are set to be m. See Fig. 2 as an example of this construction. In Fig. 2, the dotted arcs represent E_t.

For this network, a flow of $\mathcal{N}$ is a function $f: E_{\mathcal{N}}\to R^+$, where R^+ is the set of nonnegative real numbers, which satisfies the following three conditions.

- $f((i,j))\le \mathrm{cap}((i,j))$ for $(i,j)\in E_{\mathcal{N}}$,
- $\sum_{(u,i)\in E_{\mathcal{N}}} f((u,i)) = \sum_{(j,u)\in E_{\mathcal{N}}} f((j,u))$ for $u\in V_{\mathcal{N}}\setminus\{s,t\}$, and
- $\sum_{(s,i)\in E_{\mathcal{N}}} f((s,i)) = m$ and $\sum_{(i,t)\in E_{\mathcal{N}}} f((i,t)) = m$.

We define the cost function of an arc $(v,t)\in E_t$ as

$$\mathrm{cost}_{(v,t)}(x) = \begin{cases} g(a_v - x) & \text{if } 0\le x < a_v, \\ 0 & \text{if } a_v \le x \le b_v, \text{ and} \\ g(x-b_v) & \text{if } x > b_v, \end{cases}$$

where x is the amount of flow on this arc. For any other arc (i,j), $\mathrm{cost}_{(i,j)}(x)=0$ for any x. Note that the cost functions are all convex under the assumption g is a convex function with $g(0)=0$. See Fig. 3 for an example of $g(x)$ and $\mathrm{cost}_{(v,t)}(x)$.

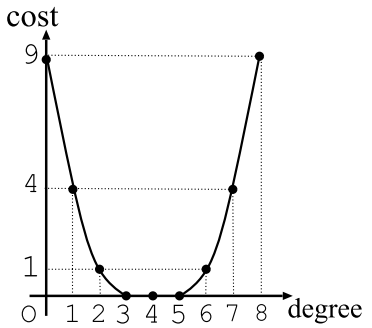

Fig. 3. Example of $\mathrm{cost}_{(v,t)}(x)$: $a_v = 3$, $b_v = 5$, and $g(x) = x^2$.

Here, we can see that if f is an integral flow, it can be regarded as an orientation of G. In fact, in any feasible integral flow of $\mathcal{N}$, for any $e = \{u, v\} \in E$, exactly one of the following cases holds: $f((e,u)) = 1$ and $f((e,v)) = 0$, or $f((e,u)) = 0$ and $f((e,v)) = 1$, which can be interpreted as an orientation (u,v) or (v,u) of e, respectively. Then, $f((v,t))$ corresponds to the outdegree of v, which implies that $cost_{(v,t)}(f((v,t)))$ is the penalty on v under the orientation, and hence $\sum_{v\in V} cost_{(v,t)}(f((v,t))) = \sum_{(i,j)\in E_{\mathcal{N}}} cost_{(i,j)}(f((i,j)))$ is the total penalty of the orientation. Therefore, our problem is to find an integral flow that minimizes $\sum_{(i,j)\in E_{\mathcal{N}}} cost_{(i,j)}(f((i,j)))$, which is CCF with integral constraints.

Since it is known that CCF with integral capacities has an integral optimal flow [24], the integral constraints can be removed. Then, we can utilize algorithms in [24] and [22]: Let N and M be the number of vertices and the number of edges in the network, respectively. Also, U and C respectively denote the largest magnitude of the lower and upper bounds on capacities of an arc and the largest magnitude of a cost on an arc in the network. The running time of the first algorithm in [24] is $O(\min\{\sqrt{\tilde{M}}, N^{2/3}U^{1/3}\}\tilde{M}\log(NC))$ time, where $\tilde{M}$ is the total capacity of edges. In our reduction, $N = O(m)$, $M = O(m)$, $U = \Delta$, $\tilde{M} = O(m)$, and C is the largest penalty on a vertex, i.e., $g(\Delta)$ since $g(0) = 0$ and g is convex. Hence this algorithm runs in $O(m^{1.5}\log(mg(\Delta)))$ time. The running time of the second algorithm in [22] is $O(M\log U(M + N\log N))$ corresponding to $O(m^2\log\Delta\log m)$ for our setting, which is smaller than the first one when $g(\Delta)$ is very large, e.g., $g(\Delta) \geq 2^m$. Consequently, we can solve MPDCO_{convex} in $O(m^{1.5}\min\{\log(mg(\Delta)), m^{0.5}\log\Delta\log m\})$ time. □

According to this theorem, the problem can be solved in $O(m^{1.5}\log m)$ time for natural polynomial penalty functions, such as $g(x) = x^k$ with a constant k, since $g(\Delta) \leq \Delta^k = O(m^k)$ for this function g.

Remark 1. This reduction is also available for hypergraph orientation. A *hypergraph* $\mathcal{H} = (V, H)$ is an extension of ordinary graphs, in which each hyperedge $e \in H$ can have more than two vertices. An orientation Λ of a hypergraph is an assignment of a hyperedge e to a vertex in e, which is a generalization of graph orientation [6]. The reduction in the proof of Theorem 2 also works for hypergraphs, and achieves essentially the same time complexity, i.e., $O(m^{1.5}\min\{\log(mg(\tilde{\Delta})), m^{0.5}\log\tilde{\Delta}\log m\})$, where $\tilde{\Delta} = \max_{v\in V}\{|\{e \mid v \in e\}|\}$.

3. General penalty setting for graphs with bounded treewidth

As seen in Sec. 1.3, MPDCO_p with a function g is APX-hard for general graphs if g is a step function or a concave function, although if g is a convex function, the problem can be solved in polynomial time as in Section 2. In this section, we restrict our attention to graphs with bounded treewidth. In Section 3.1, we first propose an $O(n\log\Delta)$-time algorithm that solves MPDCO_p with any functions on trees for general functions (Section 3.1.1), and then propose an $O(n)$-time algorithm for restricted functions (Section 3.1.2). Then, in Section 3.2, we design an FPT algorithm parameterized by treewidth plus the maximum degree, which also solves MPDCO_p with any functions. Finally, in Section 3.3, we show that MPDCO_p parameterized by treewidth is W[1]-hard for a step (or concave) function, i.e., it seems difficult to design any FPT algorithm parameterized by treewidth only.

3.1. Polynomial-time algorithm for trees

3.1.1. $O(n\log\Delta)$-time algorithm

Our algorithm is based on dynamic programming. To explain the idea, we first introduce some notation. We denote the optimal value of MPDCO_p of a graph G by $OPT(G)$ in the following. For a tree T rooted at a vertex r, we consider a tree $T + s$ in which a vertex s is attached with r, that is, $T + s = (V(T) \cup \{s\}, E(T) \cup \{\{r, s\}\})$ with rooted at s. In the context of MPDCO_p, s is a virtual vertex for which no penalty is charged in any orientation (or, we assume $a_s = 0$ and $b_s = \infty$).

In this setting, we consider two "optimal" orientations of $T + s$; one is an optimal orientation under the constraint that $\{s, r\}$ is oriented as (s, r), and the other is the one under the constraint that $\{r, s\}$ is oriented as (r, s). We denote the values of such orientations by $q^-(T)$ and $q^+(T)$, respectively. That is, these can be represented by

$$q^-(T) = \min_{\Lambda\in\mathcal{L}(T)} \left\{ \sum_{v\in V(T)} g(c_T(\Lambda, v)) \right\} \text{ and}$$

$$q^+(T) = \min_{\Lambda\in\mathcal{L}(T)} \left\{ g(\theta_r(d^+_T(\Lambda, r)+1)) + \sum_{v\in V(T)\setminus\{r\}} g(c_T(\Lambda, v)) \right\},$$

where $\mathcal{L}(T)$ is the set of orientations of T. Note that $q^-(T) = OPT(T)$. Clearly, $OPT(T+s) = \min\{q^-(T), q^+(T)\}$.

Now we show a "principle of optimality" equation. Let

$$\mathcal{L}(v,k) = \{\Lambda \mid \Lambda \in \mathcal{L}(T), d^+_T(\Lambda, v) = k\},$$

and T_s be the subtree of T rooted at a vertex s. Then, we have

$$q^-(T) = \min_{k\in\{0,\ldots,d_T(r)\}} \left\{ \min_{\Lambda\in\mathcal{L}(r,k)} \{g(\theta_r(k)) + q(T,k)\} \right\} \text{ and}$$

$$q^+(T) = \min_{k\in\{0,\ldots,d_T(r)\}} \left\{ \min_{\Lambda\in\mathcal{L}(r,k)} \{g(\theta_r(k+1)) + q(T,k)\} \right\},$$

where

$$q(T,k) = \min_{\substack{N'\subseteq N_T(r)\\ |N'|=k}} \left\{ \sum_{s\in N'} q^-(T_s) + \sum_{s\in N_T(r)\setminus N'} q^+(T_s) \right\}.$$

Here $q(T,k)$ is the minimum total penalty among all orientations of T, ignoring the cost on r under the constraint that $d^+_T(\Lambda, r) = k$.

The values $g(\theta_r(k))$ and $g(\theta_r(k+1))$ depend only on k (they do not depend how T is oriented). Hence, $q^-(T)$ and $q^+(T)$ are transformed to

$$q^-(T) = \min_{k\in\{0,\ldots,d_T(r)\}} \left\{ g(\theta_r(k)) + \min_{\Lambda\in\mathcal{L}(r,k)} \{q(T,k)\} \right\} \text{ and}$$

$$q^+(T) = \min_{k\in\{0,\ldots,d_T(r)\}} \left\{ g(\theta_r(k+1)) + \min_{\Lambda\in\mathcal{L}(r,k)} \{q(T,k)\} \right\}.$$

Also, the condition $\Lambda \in \mathcal{L}(r,k)$ is implied by the definition of $q(T,k)$, in which the two conditions $N' \subseteq N_T(r)$ and $|N'| = k$ are satisfied, edges $\{r, v\}$'s for $v \in N'$ are oriented towards v, and all other edges incident to r are oriented towards r. Therefore, $\min_{\Lambda\in\mathcal{L}(r,k)}\{q(T,k)\}$ is transformed to just $q(T,k)$ without "$\min_{\Lambda\in\mathcal{L}(r,k)}$." Hence $q^-(T)$ and $q^+(T)$ are simplified to

$$q^-(T) = \min_{k\in\{0,\ldots,d_T(r)\}} \{g(\theta_r(k)) + q(T,k)\} \text{ and} \tag{1}$$

$$q^+(T) = \min_{k\in\{0,\ldots,d_T(r)\}} \{g(\theta_r(k+1)) + q(T,k)\}. \tag{2}$$

In addition, $q(T,k)$ can be rewritten as

$$q(T,k) = \sum_{s\in N_T(r)} q^+(T_s) + \min_{\substack{N'\subseteq N_T(r)\\ |N'|=k}} \left\{ \sum_{s\in N'} h(T_s) \right\}, \tag{3}$$

where $h(T_s) = q^-(T_s) - q^+(T_s)$. Namely, $q^-(T)$ and $q^+(T)$ can be essentially computed by $(d_T(r)+1)$-times evaluation of

$$\tilde{h}(T,k) \stackrel{\text{def}}{=} \min_{\substack{N'\subseteq N_T(r)\\ |N'|=k}} \left\{ \sum_{s\in N'} h(T_s) \right\}. \tag{4}$$

Without loss of generality, let $N_T(r) = \{v_1, \ldots, v_d\}$, $I = \{1, \ldots, d\}$, and $h'_i = h(T_{v_i})$ for $i \in I$, where $d = d_T(r)$ for simplicity. By sorting h'_i's, we can obtain a list $h'_{i_1}, h'_{i_2}, \ldots, h'_{i_d}$ such that $h'_{i_1} \le h'_{i_2} \le \cdots \le h'_{i_d}$ and $i_j \in I$ for $j \in \{1, \ldots, d\}$. Then, it holds that

$$\tilde{h}(T,k) = \sum_{j=1}^{k} h'_{i_j}. \tag{5}$$

Based on these ideas, we can compute $q^-(T)$ and $q^+(T)$ of T rooted at r from the values of $q^-(T_s)$ and $q^+(T_s)$ for $s \in N_T(r)$ as follows: Let $Q = \sum_{s \in N_T(r)} q^+(T_s)$ which is the first term of the right hand side of (3). The same value Q is used to compute $q(T,k)$ for every k. Thus, we compute Q in $O(d_T(r))$ time only once. Then, we compute $h(T_s)$ for each $s \in N_T(r)$ in $O(d_T(r))$ time, and construct a list L of $h(T_s)$'s, whose length is $d_T(r)$. Spending $O(d_T(r)\log d_T(r))$ time, L is sorted, say, by the merge sort. After that, based on the value $\sum_{s \in N_T(r)} q^+(T_s)$ and the sorted list L, we obtain $q^-(T)$ and $q^+(T)$ for each $k \in \{0, \ldots, d_T(r)\}$ by the following procedure.

1. Compute Q and construct the sorted list L. This step takes $O(d_T(r)) + O(d_T(r)\log d_T(r)) = O(d_T(r)\log d_T(r))$ time as described above.
2. Let $k = 0$ and $h_k = 0$ which corresponds to $\tilde{h}(T,0)$. This step takes constant time.
3. If $k \geq 1$, add the kth value in L to h_k, which obtains $\tilde{h}(T,k)$ based on (5). This step takes constant time.
4. Compute $q(T,k) = Q + h_k$ in constant time by the equation (3), based on Q and h_k.
5. Compute $g(\theta_r(k)) + q(T,k)$ and $g(\theta_r(k+1)) + q(T,k)$.
6. If $k < d_T(r)$, then let $k = k+1$ and go back to Step 3. Otherwise, i.e., if $k = d_T(r)$, then halt. This step also takes constant time.

In the above, each of Steps 2 through 6 spends constant time only for each $k \in \{0, \ldots, d_T(r)\}$. Hence, in total, we need $O(d_T(r)\log d_T(r))$ time to obtain the values $g(\theta_r(k)) + q(T,k)$ and $g(\theta_r(k+1)) + q(T,k)$ for every k. After that, we need to take the minimum from those $O(d_T(r))$ values in (1) and (2), which takes $O(d_T(r))$ time.

In summary, $q^-(T)$ and $q^+(T)$ can be computed in $O(d_T(r)\log d_T(r))$ time, provided $q^-(T_s)$ and $q^+(T_s)$ for every $s \in N_T(r)$. Thus, for an input tree G, $q^+(G)$ and $q^-(G)(= OPT(G))$ are obtained by the bottom up manner and the total running time is

$$\sum_{v \in V(G)} O(d_G(v)\log d_G(v)) = O(n \log \Delta). \tag{6}$$

Finally we have the following theorem.

Theorem 3. *For any linearly separable penalty function, MPDCO$_p$ can be solved in $O(n \log \Delta)$ time when the input graph is a tree.*

Remark 2. The above $O(n \log \Delta)$-time algorithm works for any rational-valued penalty function as long as basic arithmetic operations can be done in $O(1)$ time; it is not necessary for g to be monotone or to satisfy $g(0) = 0$.

3.1.2. $O(n)$-time algorithm for restricted functions

In this section, we show that the running time of the algorithm in the previous section can be reduced if the separated penalty function has some special property. An important observation is that we do not need to check every $k \in \{0, \ldots, d_T(r)\}$ in (1) and (2) for such functions. We continue to use the symbols used in the previous section.

Consider a step function g such that $g(0) = 0$ and $g(x) = c$ for $x > 0$ with some positive constant c. The shape of this function g is drawn by only one segment $g(x) = c$ with a point $g(0) = 0$. Namely, it can be written as $g(x) = \sum_{i=0}^{1} \alpha_i \chi_{A_i}(x)$ such that $\alpha_0 = 0$, $\alpha_1 = c$, $A_0 = [0,0]$, and $A_1 = (0, \Delta]$, where χ_A is the indicator function of an interval A such that $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ otherwise. Note that the domain of the separated penalty functions is $[0, \Delta]$. In general, a step function can be written as $g(x) = \sum_{i=0}^{k} \alpha_i \chi_{A_i}(x)$ for some α_i's and A_i's. As for convex and concave functions, some of them are piecewise linear functions. Namely, a convex or a concave function can be written as $g(x) = \sum_{i=0}^{k} (\alpha_i x + \beta_i)\chi_{A_i}(x)$, where $\alpha_i x + \beta_i$ is a linear function with constants α_i and β_i, defined on an interval A_i. In this section, we consider functions such that the number k of intervals is a constant. We call such function *a function with constant intervals*.

Let us consider the case that separated penalty function is a step function g such that $g(0) = 0$ and $g(x) = c$ for $x > 0$ with some positive constant c. Let L and R be the largest indices such that $h'_{i_L} < 0$ and $h'_{i_R} \leq 0$, respectively, i.e., $h'_{i_{L+1}}, \ldots, h'_{i_R}$ are all 0. Thus, it holds that $\min_{k \in \{0,\ldots,d_T(r)\}}\{\tilde{h}(T,k)\} = \tilde{h}(T,L) = \tilde{h}(T,L+1) = \cdots = \tilde{h}(T,R)$. Moreover, $\tilde{h}(T,k)$ is decreasing when $k \leq L$ and increasing when $k \geq R$. If $h'_{i_1} > 0$, we let $L = R = 0$. In below, we show that the equations (1) and (2) in the previous section can be simplified for such a function g, which leads to the linear-time algorithm.

We consider two cases (Case 1) $[a,b] \cap [L,R] \neq \emptyset$, and (Case 2) $[a,b] \cap [L,R] = \emptyset$, where $[x,y]$ represents a closed interval between x and y. Note that a, b, L, and R are all integers between 0 and Δ.

Case 1 $[a,b] \cap [L,R] \neq \emptyset$:

See Fig. 4-(Left) for an example. Let κ be the smallest integer in $[a,b] \cap [L,R]$. Then, since $\theta_r(\kappa) = 0$ implying that $g(\theta_r(\kappa)) = 0$ and $\tilde{h}(T,\kappa)$ is the minimum, it holds that

$$q^-(T) = \min_{k \in \{0,\ldots,d_T(r)\}} \{g(\theta_r(k)) + q(T,k)\} = Q + \tilde{h}(T,\kappa).$$

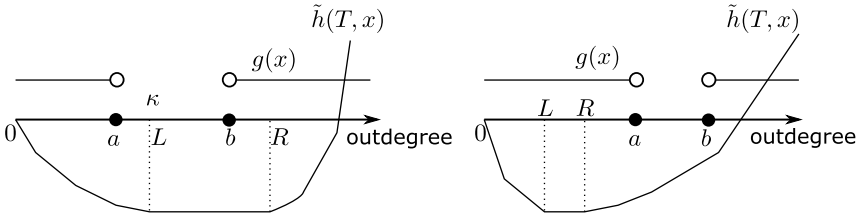

Fig. 4. Computation of $q^-(T)$ and $q^+(T)$: (Left) $[a,b]\cap[L,R]\neq\emptyset$. According to the definition of κ, we have $\kappa = L$ in this example. (Right) $[a,b]\cap[L,R]=\emptyset$.

For $q^+(T)$, we combine $g(\theta_r(k+1))$ and $q(T,k)$ in (2). Even if $q(T,\kappa)$ takes the minimum, it may happen $g(\theta_r(\kappa+1))=c$ when $\kappa=b$. However, in this case, $k=\kappa-1$ is the only other possibility which gives the minimum based on the property of $\tilde{h}(T,k)$ mentioned in the above. Therefore we can obtain $q^+(T)$ as follows.

$$\begin{aligned} q^+(T) &= \min_{k\in\{0,\ldots,d_T(r)\}}\{g(\theta_r(k+1))+q(T,k)\} \\ &= \min\{g(\theta_r(\kappa+1))+Q+\tilde{h}(T,\kappa),\ g(\theta_r(\kappa))+Q+\tilde{h}(T,\kappa-1)\}. \end{aligned}$$

Case 2 $[a,b]\cap[L,R]=\emptyset$:
See Fig. 4-(Right) for an example. Since $[a,b]\cap[L,R]=\emptyset$, either of $R\le a-1$ or $b+1\le L$ holds. Assume that $R\le a-1$. In this case, $g(\theta_r(k))+q(T,k)$ takes the minimum when $k=a$ or $k=R$. (k can be either of $L+1,\ldots,R$, when we only consider $q^-(T)$. However we prefer setting $k=R$ for $q^+(T)$ in the discussion below.) Hence, we observe that

$$\begin{aligned} q^-(T) &= \min_{k\in\{0,\ldots,d_T(r)\}}\{g(\theta_r(k))+q(T,k)\} \\ &= \min\{c+Q+\tilde{h}(T,R),\ Q+\tilde{h}(T,a)\}. \end{aligned}$$

As for $q^+(T)$, $g(\theta_r(k+1))$ becomes 0 if $k=a-1$. Thus, setting $k=R$ may yield $g(\theta_r(k+1))=0$. This is the reason that we chose $k=R$ in the above. Hence, we take the minimum among two settings $k=R$ and $k=a-1$:

$$\begin{aligned} q^+(T) &= \min_{k\in\{0,\ldots,d_T(r)\}}\{g(\theta_r(k+1))+q(T,k)\} \\ &= \min\{g(\theta_r(R+1))+Q+\tilde{h}(T,R),\ Q+\tilde{h}(T,a-1)\}. \end{aligned}$$

Similarly, for the case $L\ge b+1$, we can obtain

$$\begin{aligned} q^-(T) &= \min_{k\in\{0,\ldots,d_T(r)\}}\{g(\theta_r(k))+q(T,k)\} \\ &= \min\{c+Q+\tilde{h}(T,L),\ Q+\tilde{h}(T,b)\},\ \text{and} \\ q^+(T) &= \min_{k\in\{0,\ldots,d_T(r)\}}\{g(\theta_r(k+1))+q(T,k)\} \\ &= \min\{g(\theta_r(L+1))+Q+\tilde{h}(T,L),\ Q+\tilde{h}(T,b-1)\}. \end{aligned}$$

We estimate the running time to compute the above equations for $q^-(T)$ and $q^+(T)$. By scanning $h(T_{v_i})$'s once, we can know L and R as the numbers of $h(T_{v_i})$'s which are negative or zero. There is an algorithm which can finds the k-th smallest number in a given list L in $O(|L|)$ time [25] where $|L|$ is the number of elements in the list L. We can utilize this algorithm to find h'_{i_L} and h'_{i_R} in $O(d_T(r))$ time. After we find h'_{i_L} (or h'_{i_R}), we can pick h'_{i_1} through h'_{i_L} (or h'_{i_R}) by scanning h'_i's again, breaking ties arbitrarily, in $O(d_T(r))$ time. According to this, for Case 1, we can compute κ in constant time, $\tilde{h}(T,\kappa)$ and $\tilde{h}(T,\kappa-1)$ in $O(d_T(r))$ time. Similarly, for Case 2, $\tilde{h}(T,R)$, $\tilde{h}(T,a-1)$, $\tilde{h}(T,L)$, and $\tilde{h}(T,b)$ are all computed in $O(d_T(r))$ time. Finally, $q^-(T)$ and $q^+(T)$ are computed in constant time based on these values, where Q is computed only once in advance spending $O(d_T(r))$ time similarly to the estimation for Theorem 3. In total, we need a constant number of $O(d_T(r))$-time computations, i.e. $q^-(T)$ and $q^+(T)$ can be computed in $O(d_T(r))$ time. This implies that the total running time of the algorithm becomes $O(n)$.

In the above, we only considered a step function having only two segments. When the separated penalty function is a function with constant intervals, a similar algorithm works. In order to find the minimum for $q^-(T)$ and $q^+(T)$, we need to check the points corresponding to endpoints of an interval, and its neighbors based on the observation that

- the sum of a constant (by a step function) and a linear function (by $\tilde{h}$) on an interval is also a linear function,
- the sum of two linear functions (by a convex/concave function and by $\tilde{h}$) on an interval is also a linear function,

and hence a minimum value appears at either end of the interval. Then, since the number of intervals is a constant, the total number of candidates is also a constant for each of the above equations. This increases the total running time by a constant number of $O(d_T(r))$-computations, which is still $O(d_T(r))$. Therefore, an optimal solution is obtained in $\sum_{v_i \in V} d_t(r) = O(n)$ time for such functions with constant intervals.

Theorem 4. *If the separated penalty function is a function with constant intervals, MPDCO$_p$ can be solved in $O(n)$ time when the input graph is a tree.*

3.2. FPT algorithm parameterized by treewidth and the maximum degree

In this section, we propose an FPT algorithm parameterized by treewidth plus the maximum degree for any linearly separable penalty functions.

3.2.1. Overview and notation

Intuitively, the treewidth is a measure of the tree-likeness of a graph. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where T is a tree with node set I and $\mathcal{X} = \{X_i \subseteq V \mid i \in I\}$, with the following three properties [26,27].

- **(i)** $\bigcup_{i \in I} X_i = V$.
- **(ii)** For every $\{u, w\} \in E$, there exists an $i \in I$ satisfying $\{u, w\} \subseteq X_i$.
- **(iii)** For all $i_1, i_2, i_3 \in I$, if i_2 is on the path between i_1 and i_3 in T, then $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$.

The width of a tree decomposition $(T, \{X_i \mid i \in I\})$ is defined by $\max_{i \in I}\{|X_i| - 1\}$. The treewidth $\tau(G)$ of G is the minimum width over all tree decompositions of G. A tree decomposition of G with minimum width t can be obtained in $O(t^{O(t^3)}n)$ time [28]. A tree decomposition $(T, \{X_i \mid i \in I\})$ is *a nice tree decomposition with introduce-edge nodes* if it satisfies four more conditions in the following [29–31].

- **(iv)** The tree decomposition is rooted at a node $r \in I$ such that $X_r = \emptyset$.
- **(v)** Every node in I has at most two children.
- **(vi)** For every edge in E, there is exactly one introduce-edge node (defined in the next condition (vii)) in T.
- **(vii)** Each node $i \in I$ belongs to one of the following five types:
  - *Leaf node*: it has no children and $X_i = \emptyset$;
  - *Introduce-vertex node*: it has exactly one child $j \in I$ with $X_i = X_j \cup \{v\}$ for a vertex $v \in V$;
  - *Forget node*: it has exactly one child $j \in I$ with $X_i = X_j \setminus \{v\}$ for a vertex $v \in V$;
  - *Introduce-edge node*: it has exactly one child $j \in I$ and is labeled with an edge $\{u, v\} \in E$ such that $\{u, v\} \subseteq X_i$ and $X_i = X_j$; and
  - *Join node*: it has exactly two children j and j' such that $X_i = X_j = X_{j'}$.

In this paper, we simply say that a tree decomposition is *nice* if it is a nice tree decomposition with introduce-edge nodes. A tree decomposition of width τ for a graph having n vertices can be transformed to a nice tree decomposition with $O(\tau n)$ nodes and width τ in $O(\tau^2 \max\{|I|, n\})$ time [31]. Thus we assume that a tree decomposition is nice and has $O(\tau n)$ nodes in the following. Also, we can assume that if $X_i = \emptyset$ for a node i, then i is either a leaf or the root r, because otherwise the input graph G is disconnected, and the algorithm in this section can be applied to each connected component.

We assume that for an input graph G, a nice tree decomposition $(T, \mathcal{X})$ of width τ is given, where the node set of T is I, T is rooted at a node $r \in I$, and $\mathcal{X} = \{X_i \mid i \in I\}$. For a node i of T, let $T[i]$ denote the subtree of T induced by i and all descendants of i. Also, we define a partition $\{E_i \mid i \in I\}$ of E according to X_i's, which is always possible due to property (vi): E_i is the set of all edges introduced in node i.

Now we design a dynamic programming algorithm (DP for short) that runs from leaves of T to the root r. Since each subtree of the form $T[i]$ corresponds to a subgraph of G, we let $G[i]$ be the subgraph whose vertex set is $\bigcup_{j \in V(T[i])} X_i$ and edge set is $\bigcup_{j \in V(T[i])} E_j$. The DP computes the optimal penalty of $G[i]$ from the optimal penalty of subgraph(s) defined by its child(ren).

For simplicity, we assume that $V(G) = \{1, 2, \ldots, n\}$ in the following. Consider a subset X_i of $V(G)$ such that $X_i = \{i_1, i_2, \ldots, i_{|X_i|}\}$ and $i_1 < i_2 < \cdots < i_{|X_i|}$. Let $D(G, X_i)$ be a set of all vectors $(\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_{|X_i|})$ such that each $\boldsymbol{d}_h$ is a non-negative integer satisfying $0 \leq \boldsymbol{d}_h \leq d_G(i_h)$ for $1 \leq h \leq |X_i|$ (recall that $d_G(i_h)$ is the degree of the vertex i_h in G). That is, $D(G, X_i)$ is a set of $|X_i|$-dimensional vectors representing sequence of (possible) outdegrees of vertices in X_i. Such a vector $\boldsymbol{d} \in D(G, X_i)$ is called an *outdegree vector for* i, and $\boldsymbol{d}_k$ represents the k-th component of $\boldsymbol{d}$. As in the definition of the nice tree decomposition, X_i is allowed to be an empty set, and we define $D(G, \emptyset) = \{()\}$, where "()" represents a zero dimensional vector. Since $|X_i| \leq \tau + 1$ and each $\boldsymbol{d}_h$ is at most Δ, the size $|D(G, X_i)|$ of $D(G, X_i)$ is $O(\Delta^{\tau+1})$.

Assume $X_i = \{i_1, i_2, \ldots, i_{|X_i|}\} \subseteq V(G)$ such that $i_1 < i_2 < \cdots < i_{|X_i|}$. For the graph $G[i]$ and an outdegree vector $\boldsymbol{d}$ for i, we define a function $OPT(i, \boldsymbol{d})$ which represents the optimal (minimum) penalty of an orientation of $G[i]$ under the constraint that the sequence of the outdegrees of every vertex $i_h \in X_i$ in the resulting orientation is $\boldsymbol{d}_h$. Namely, $OPT(i, \boldsymbol{d})$ for a non-leaf node i is defined as

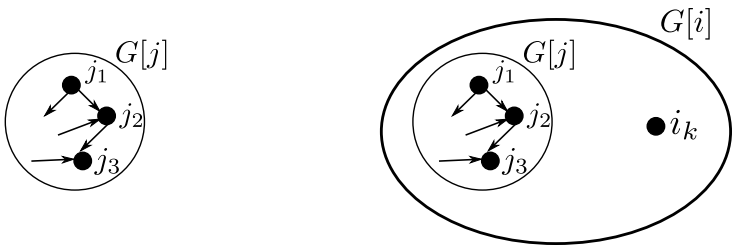

Fig. 5. Illustrating a node of type *introduce-vertex node*: (Left) A graph $G[j]$ and an orientation of $G[j]$; (Right) A graph $G[i]$ in which the vertex i_k is introduced, and an orientation of $G[i]$.

$$OPT(i, \boldsymbol{d}) = \begin{cases} \min_{\Lambda \in \mathcal{L}(i,\boldsymbol{d})}\{p(c_{G[i]}(\Lambda))\} & \text{if } \mathcal{L}(i,\boldsymbol{d}) \neq \emptyset, \text{ and} \\ +\infty & \text{otherwise,} \end{cases} \tag{7}$$

where $\mathcal{L}(i, \boldsymbol{d}) = \{\Lambda \mid \Lambda$ is an orientation of $G[i], d^+_{G[i]}(\Lambda, i_h) = \boldsymbol{d}_h$ for $1 \leq h \leq |X_i|\}$. We say that an orientation $\Lambda \in \mathcal{L}(i, \boldsymbol{d})$ *follows* $\boldsymbol{d}$. In case i is a leaf node, we define $OPT(i, ()) = 0$. Thus, $\min_{\boldsymbol{d} \in D(G,X_i)}\{OPT(i, \boldsymbol{d})\}$ is the optimal penalty of MPDCO$_p$ for $G[i]$, and then the optimal penalty for G is $\min_{\boldsymbol{d} \in D(G[r],X_r)}\{OPT(r, \boldsymbol{d})\} = OPT(r, ())$, where $G[r] = G$ and $X_r = \emptyset$. We show that $OPT(i, \boldsymbol{d})$ can be computed from the information on i's child(ren) according to the type of i as follows. In the following subsections, we consider four cases, each of which corresponds to a type of a node in (vii) (except for leaf nodes) of the definition of the nice tree decomposition.

3.2.2. Introduce-vertex node

Let i be an introduce-vertex node. Without loss of generality, assume that

- j is the child of i,
- $X_j = \{j_1, j_2, \ldots, j_{|X_j|}\}$ such that $j_1 < j_2 < \cdots < j_{|X_j|}$,
- A vertex i_k is introduced in i for some $i_k \in V$, i.e., $X_i = X_j \cup \{i_k\}$, and
- $X_i = \{i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1}, \ldots, i_{|X_i|}\}$ such that $i_1 < i_2 < \cdots < i_{|X_i|}$, $i_h = j_h$ for $1 \leq h \leq k-1$, and $i_{h+1} = j_h$ for $k \leq h \leq |X_j|$.

If j is a leaf node, i.e., i_k is the only vertex in $G[i]$ having no edges, then we set

$$OPT(i, (0)) = 0,$$

which takes constant time. For $1 \leq h \leq d_G(i_k)$, we set

$$OPT(i, (h)) = +\infty,$$

which also takes constant time for each h, since we can not orient any edge in $G[i]$ and so the outdegree of i_k must be zero.

Suppose that j is not a leaf node. Since the vertex i_k is introduced as an isolated vertex, an optimal orientation of $G[j]$ can be also used as an optimal orientation for the part $G[j]$ in $G[i]$. Namely, an optimal orientation of $G[i]$ is the same as an optimal orientation of $G[j]$. See Fig. 5. Thus, for outdegree vectors $\boldsymbol{i}$ for i and $\boldsymbol{j}$ for j, if $\boldsymbol{i}_h = \boldsymbol{j}_h$ for $1 \leq h \leq k-1$, $\boldsymbol{i}_k = 0$, and $\boldsymbol{i}_h = \boldsymbol{j}_{h-1}$ for $k+1 \leq h \leq |X_i|$ hold, then we set

$$OPT(i, \boldsymbol{i}) = OPT(j, \boldsymbol{j}),$$

considering that an optimal orientation for $G[j]$ is also applied to the part $G[j]$ in $G[i]$. For each $\boldsymbol{i} \in D(G, X_i)$, it takes $O(\tau)$ time to obtain an outdegree vector $\boldsymbol{j}$ which satisfies the condition if it exists, since $|X_i| \leq \tau + 1$ and so $\boldsymbol{i}$ is an $O(\tau)$-dimensional vector. If such an outdegree vector $\boldsymbol{j}$ for j does not exist, then we set

$$OPT(i, \boldsymbol{i}) = +\infty,$$

meaning that such an orientation following the outdegree vector $\boldsymbol{i}$ does not exist. Thus, it takes $O(\tau)$ time for the case j is not a leaf node.

Since $|D(G, X_i)| = O(\Delta^{\tau+1})$ and it takes $O(\tau)$ time for each case, the recursive formula for an introduce-vertex node i can be computed in $O(\tau\Delta^{\tau+1})$ time in total:

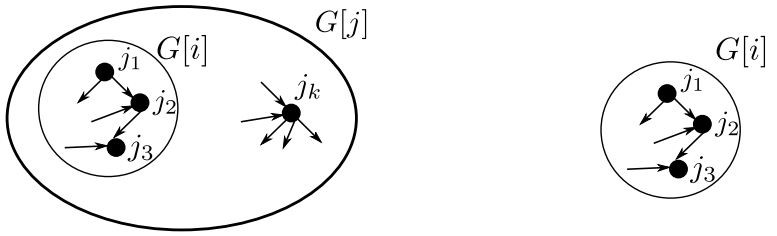

Fig. 6. Illustrating a node of type *forget node*: (Left) A graph $G[j]$ and an orientation of $G[j]$; (Right) A graph $G[i]$ constructed by removing the vertex j_k, and an orientation of $G[i]$.

Claim 5. *For an introduce-vertex node i whose child is j and $\boldsymbol{i} \in D(G, X_i)$,*

$$OPT(i, \boldsymbol{i}) = \begin{cases} 0 \\ \quad \text{if } j \text{ is a leaf node and } \boldsymbol{i} = (0), \\ OPT(j, \boldsymbol{j}) \\ \quad \text{if } j \text{ is not a leaf node, } \boldsymbol{i}_h = \boldsymbol{j}_h \text{ for } 1 \le h \le k-1, \boldsymbol{i}_k = 0, \\ \quad \text{and } \boldsymbol{i}_h = \boldsymbol{j}_{h-1} \text{ for } k+1 \le h \le |X_i|, \text{ and} \\ +\infty \\ \quad \text{otherwise,} \end{cases} \tag{8}$$

which can be computed in $O(\tau)$ time. Hence we spend $O(\tau \Delta^{\tau+1})$ time for each introduce-vertex node i. □

3.2.3. Forget node

Let i be a forget node. Without loss of generality, assume that

- j is the child of i,
- $X_j = \{j_1, j_2, \ldots, j_{|X_j|}\}$ such that $j_1 < j_2 < \cdots < j_{|X_j|}$, and
- A vertex j_k is removed from j for some k such that $1 \le k \le |X_j|$, i.e., $X_i = X_j \setminus \{j_k\}$.

Note that j must be a non-leaf node with $X_j \neq \emptyset$, since i removes a vertex j_k from X_j. The set X_i may be empty for the case i is the root node r. If $X_i = \emptyset$, then it must hold that $\boldsymbol{i} = ()$ and j is a node including only one vertex j_1. Then the outdegree of the vertex j_1 in $G[i]$ can be any value between 0 and $d_G(j_1)$. Hence we take the minimum among penalties of orientations of $G[j]$ following an outdegree vector (h) for $0 \le h \le d_G(j_1)$ in this case:

$$OPT(i, ()) = \min_{0 \le h \le d_G(j_1)} \{OPT(j, (h))\},$$

which takes $O(d_G(j_1)) = O(\Delta)$ time with checking whether $X_i = \emptyset$ and $\boldsymbol{i} = ()$ in constant time. If $X = \emptyset$ but $\boldsymbol{i} \neq ()$, then we set

$$OPT(i, \boldsymbol{i}) = +\infty,$$

which takes constant time.

Assume that $X_i \neq \emptyset$. Consider outdegree vectors $\boldsymbol{i}$ for i and $\boldsymbol{j}$ for j, and an optimal orientation of $G[i]$ which follows $\boldsymbol{i}$. Suppose that $\boldsymbol{i}_h = \boldsymbol{j}_h$ for $1 \le h \le k-1$ and $\boldsymbol{i}_h = \boldsymbol{j}_{h+1}$ for $k \le h \le |X_i|$ hold. Then if we use an orientation of $G[j]$ following $\boldsymbol{j}$ as an orientation of $G[i]$, it follows $\boldsymbol{i}$. Under such an orientation of $G[j]$, the outdegree of the vertex j_k in $G[i]$ can be any value between 0 and $d_G(j_k)$. See Fig. 6. Let $J_k(\boldsymbol{i})$ be a set of $|X_j|$-dimensional vector $\boldsymbol{j}$'s such that $\boldsymbol{j}_h = \boldsymbol{i}_h$ for $1 \le h \le k-1$, $\boldsymbol{j}_h = \boldsymbol{i}_{h-1}$ for $k+1 \le h \le |X_j|$, and $0 \le \boldsymbol{j}_k \le d_G(j_k)$. We take the minimum among penalties of such orientations following $\boldsymbol{i}$ if $J_k(\boldsymbol{i}) \neq \emptyset$:

$$OPT(i, \boldsymbol{i}) = \min_{\boldsymbol{j} \in J_k(\boldsymbol{i})} \{OPT(j, \boldsymbol{j})\}.$$

Since $\boldsymbol{j}_k$ is at most Δ and $\boldsymbol{i}$ is an $O(\tau)$-dimensional vector, $|J_k(\boldsymbol{i})| = O(\Delta)$ and $J_k(\boldsymbol{i})$ can be obtained in $O(\tau\Delta)$ time. Taking the minimum spends $O(\Delta)$ time, as a result, this can be computed in $O(\tau\Delta)$ time. If $X_i \neq \emptyset$ but $J_k(\boldsymbol{i}) = \emptyset$, we set

$$OPT(i, \boldsymbol{i}) = +\infty,$$

which takes constant time.

Since $|D(G, X_i)| = O(\Delta^{\tau+1})$ and it takes $O(\tau\Delta)$ time for each case, the recursive formula for a forget node i can be computed in $O(\tau\Delta^{\tau+2})$ time in total:

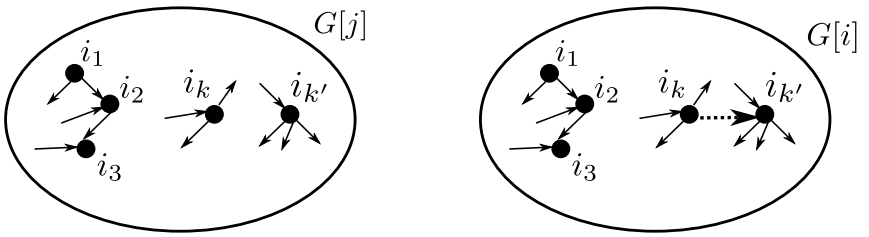

Fig. 7. Illustrating a node of type *introduce-edge node*: (Left) A graph $G[j]$ and an orientation of $G[j]$; (Right) A graph $G[i]$ constructed by inserting an edge $\{i_k, i_{k'}\}$, and an orientation of $G[i]$ in which the edge $\{i_k, i_{k'}\}$ is oriented as $(i_k, i_{k'})$.

Claim 6. *For a forget node i whose child is j and $\boldsymbol{i} \in D(G, X_i)$,*

$$OPT(i, \boldsymbol{i}) = \begin{cases} \min_{0 \le h \le d_G(j_1)}\{OPT(j, (h))\} & \text{if } X_i = \emptyset \text{ and } \boldsymbol{i} = (), \\ \min_{\boldsymbol{j} \in J_k(\boldsymbol{i})}\{OPT(j, \boldsymbol{j})\} & \text{if } X_i \neq \emptyset \text{ and } J_k(\boldsymbol{i}) \neq \emptyset, \text{ and} \\ +\infty & \text{otherwise}, \end{cases} \tag{9}$$

where $J_k(\boldsymbol{i})$ is a set of $|X_j|$-dimensional vector $\boldsymbol{j}$'s such that $\boldsymbol{j}_h = \boldsymbol{i}_h$ for $1 \le h \le k-1$, $\boldsymbol{j}_h = \boldsymbol{i}_{h-1}$ for $k+1 \le h \le |X_j|$, and $0 \le \boldsymbol{j}_k \le d_G(j_k)$. The above recursive formula can be computed in $O(\tau\Delta)$ time for each $\boldsymbol{i}$, and hence the total time required for a forget node i is $O(\tau\Delta^{\tau+2})$. □

3.2.4. Introduce-edge node

Let i be an introduce-edge node. Without loss of generality, assume that

- j is the child of i,
- $X_i = X_j = \{i_1, i_2, \ldots, i_{|X_i|}\}$, and
- An edge $\{i_k, i_{k'}\}$ is introduced in i for some k and k' such that $k \neq k'$ and $\{k, k'\} \subseteq \{1, 2, \ldots, |X_i|\}$.

See Fig. 7. The edge $\{i_k, i_{k'}\}$ is oriented in either way $(i_k, i_{k'})$ or $(i_{k'}, i_k)$ in an orientation. In the former case, the outdegree of i_k increases by one. Similarly, in the latter case, the outdegree of $i_{k'}$ is increased by one. In both cases, outdegrees of other vertices do not change. Suppose that we construct an orientation Λ_i of $G[i]$ by adding $(i_k, i_{k'})$ to an orientation Λ_j of $G[j]$. Then if Λ_i and Λ_j respectively follow outdegree vectors $\boldsymbol{i}$ and $\boldsymbol{j}$, then it holds that $\boldsymbol{i}_k = \boldsymbol{j}_k + 1$ and $\boldsymbol{i}_h = \boldsymbol{j}_h$ for $h \neq k$. If we add $(i_{k'}, i_k)$ to Λ_j instead of $(i_k, i_{k'})$, then it holds that $\boldsymbol{i}_{k'} = \boldsymbol{j}_{k'} + 1$ and $\boldsymbol{i}_h = \boldsymbol{j}_h$ for $h \neq k'$. An important thing here is that the increment of the outdegree of the vertex i_k (or $i_{k'}$) increases the penalty on i_k (or $i_{k'}$) from $g(\theta_{i_k}(\boldsymbol{i}_k - 1))$ to $g(\theta_{i_k}(\boldsymbol{i}_k))$ (or from $g(\theta_{i_{k'}}(\boldsymbol{i}_{k'} - 1))$ to $g(\theta_{i_{k'}}(\boldsymbol{i}_{k'}))$). Penalties on other vertices do not change.

Let $\boldsymbol{i}^{(\boldsymbol{k})}$ be an $|X_i|$-dimensional vector such that $\boldsymbol{i}^{(\boldsymbol{k})}_k = \boldsymbol{i}_k - 1$ and $\boldsymbol{i}^{(\boldsymbol{k})}_h = \boldsymbol{i}_h$ for $h \neq k$. Similarly let $\boldsymbol{i}^{(\boldsymbol{k'})}$ be an $|X_i|$-dimensional vector such that $\boldsymbol{i}^{(\boldsymbol{k'})}_{k'} = \boldsymbol{i}_{k'} - 1$ and $\boldsymbol{i}^{(\boldsymbol{k'})}_h = \boldsymbol{i}_h$ for $h \neq k'$. Since $|X_i| \le \tau + 1$, obtaining $\boldsymbol{i}^{(\boldsymbol{k})}$ and $\boldsymbol{i}^{(\boldsymbol{k'})}$ takes $O(\tau)$ time. Here, $\boldsymbol{i}^{(\boldsymbol{k})}$ and/or $\boldsymbol{i}^{(\boldsymbol{k'})}$ may not belong to $D(G, X_j)$, e.g., if $\boldsymbol{i}_k = 0$, then $\boldsymbol{i}^{(\boldsymbol{k})}_k = -1$ and hence $\boldsymbol{i}^{(\boldsymbol{k})} \notin D(G, X_j)$. Thus, by checking whether $\boldsymbol{i}_k$ and $\boldsymbol{i}_{k'}$ are positive in constant time, we can know whether $\boldsymbol{i}^{(\boldsymbol{k})}$ and $\boldsymbol{i}^{(\boldsymbol{k'})}$ belong to $D(G, X_j)$. If $\boldsymbol{i}^{(\boldsymbol{k})} \in D(G, X_j)$ but $\boldsymbol{i}^{(\boldsymbol{k'})} \notin D(G, X_j)$, then the edge $\{i_k, i_{k'}\}$ can be oriented only as $(i_k, i_{k'})$. Thus, we set

$$OPT(i, \boldsymbol{i}) = OPT(j, \boldsymbol{i}^{(\boldsymbol{k})}) - g(\theta_{i_k}(\boldsymbol{i}_k - 1)) + g(\theta_{i_k}(\boldsymbol{i}_k)), \tag{10}$$

which takes constant time assuming $\boldsymbol{i}^{(\boldsymbol{k})}$ is given. Similarly, if $\boldsymbol{i}^{(\boldsymbol{k})} \notin D(G, X_j)$ but $\boldsymbol{i}^{(\boldsymbol{k'})} \in D(G, X_j)$, then we set

$$OPT(i, \boldsymbol{i}) = OPT(j, \boldsymbol{i}^{(\boldsymbol{k'})}) - g(\theta_{i_{k'}}(\boldsymbol{i}'_{\boldsymbol{k}} - 1)) + g(\theta_{i_{k'}}(\boldsymbol{i}'_{\boldsymbol{k}})), \tag{11}$$

which again takes constant time assuming $\boldsymbol{i}^{(\boldsymbol{k'})}$ is given. In the case that both of $\boldsymbol{i}^{(\boldsymbol{k})}$ and $\boldsymbol{i}^{(\boldsymbol{k'})}$ belong to $D(G, X_j)$, we need to take the minimum of penalties of the two orientations in which the edge $\{i_k, i_{k'}\}$ is oriented either way $(i_k, i_{k'})$ or $(i_{k'}, i_k)$. Namely, we set

$$OPT(i, \boldsymbol{i}) = \min\{OPT(j, \boldsymbol{i}^{(\boldsymbol{k})}) - g(\theta_{i_k}(\boldsymbol{i}_k - 1)) + g(\theta_{i_k}(\boldsymbol{i}_k)), OPT(j, \boldsymbol{i}^{(\boldsymbol{k'})}) - g(\theta_{i_{k'}}(\boldsymbol{i}_{k'} - 1)) + g(\theta_{i_{k'}}(\boldsymbol{i}_{k'}))\}, \tag{12}$$

which also takes constant time assuming $\boldsymbol{i}^{(\boldsymbol{k})}$ and $\boldsymbol{i}^{(\boldsymbol{k'})}$ are given. Note that the above equations (10), (11), and (12) can be computed depending on $\boldsymbol{i}_k$ and $\boldsymbol{i}_{k'}$, since the penalty function p is linearly separable with a function g.

Since $|D(G, X_i)| = O(\Delta^{\tau+1})$ and it takes $O(\tau)$ time for each case, the recursive formula for an introduce-edge node i can be computed in $O(\tau\Delta^{\tau+1})$ time in total:

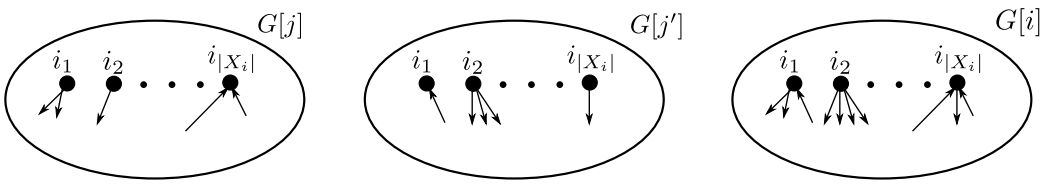

Fig. 8. Illustrating a node of type *join node*: (Left) A graph $G[j]$ and an orientation Λ_j of $G[j]$; (Center) A graph $G[j']$ and an orientation $\Lambda_{j'}$ of $G[j']$; (Right) A graph $G[i]$ which merges $G[j]$ and $G[j']$, and an orientation of $G[i]$ constructed from Λ_j and $\Lambda_{j'}$.

Claim 7. *For an introduce-edge node i whose child is j and $\boldsymbol{i} \in D(G, X_i)$,*

$$OPT(i,\boldsymbol{i}) = \begin{cases} OPT(j,\boldsymbol{i}^{(\boldsymbol{k})}) - g(\theta_{i_k}(\boldsymbol{i}_k - 1)) + g(\theta_{i_k}(\boldsymbol{i}_k)) \\ \quad \text{if } \boldsymbol{i}_k \geq 1 \text{ and } \boldsymbol{i}_{k'} = 0, \\ OPT(j,\boldsymbol{i}^{(\boldsymbol{k'})}) - g(\theta_{i_{k'}}(\boldsymbol{i}_{k'} - 1)) + g(\theta_{i_{k'}}(\boldsymbol{i}_{k'})) \\ \quad \text{if } \boldsymbol{i}_k = 0 \text{ and } \boldsymbol{i}_{k'} \geq 1, \\ \min\{OPT(j,\boldsymbol{i}^{(\boldsymbol{k})}) - g(\theta_{i_k}(\boldsymbol{i}_k - 1)) + g(\theta_{i_k}(\boldsymbol{i}_k)), \\ \quad OPT(j,\boldsymbol{i}^{(\boldsymbol{k'})}) - g(\theta_{i_{k'}}(\boldsymbol{i}_{k'} - 1)) + g(\theta_{i_{k'}}(\boldsymbol{i}_{k'}))\} \\ \quad \text{if } \boldsymbol{i}_k \geq 1 \text{ and } \boldsymbol{i}_{k'} \geq 1, \text{ and} \\ +\infty \\ \quad \text{otherwise,} \end{cases} \tag{13}$$

where $\boldsymbol{i}^{(\boldsymbol{k})}$ ($\boldsymbol{i}^{(\boldsymbol{k'})}$, resp.) is an $|X_i|$-dimensional vector such that $\boldsymbol{i}^{(\boldsymbol{k})}_k = \boldsymbol{i}_k - 1$ and $\boldsymbol{i}^{(\boldsymbol{k})}_h = \boldsymbol{i}_h$ for $h \neq k$ ($\boldsymbol{i}^{(\boldsymbol{k'})}_{k'} = \boldsymbol{i}_{k'} - 1$ and $\boldsymbol{i}^{(\boldsymbol{k'})}_h = \boldsymbol{i}_h$ for $h \neq k'$, resp.). The above recursive formula can be computed in $O(\tau)$ time for each $\boldsymbol{i}$, and hence the total time required for an introduce-edge node i is $O(\tau\Delta^{\tau+1})$. □

3.2.5. Join node

Let i be a join node. Without loss of generality, assume that

- j and j' are the children of i and
- $X_i = X_j = X_{j'} = \{i_1, i_2, \ldots, i_{|X_i|}\}$.

Since $E(G[j])$ and $E(G[j'])$ are disjoint, we can construct an orientation Λ_i of $G[i]$ by combining an orientation Λ_j of $G[j]$ with an orientation $\Lambda_{j'}$ of $G[j']$. See Fig. 8. Under the resulting orientation Λ_i of $G[i]$, it holds that $d^+_{G[i]}(\Lambda_i, i_h) = d^+_{G[j]}(\Lambda_j, i_h) + d^+_{G[j']}(\Lambda_{j'}, i_h)$. Thus, if Λ_i, Λ_j, and $\Lambda_{j'}$ respectively follow outdegree vectors $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{j'}$, then it holds that $\boldsymbol{i}_h = \boldsymbol{j}_h + \boldsymbol{j'}_h$ for $1 \leq h \leq |X_i|$. At this moment, the penalty on the vertex i_h under Λ_i is $g(c_{G[i]}(\Lambda_i, i_h)) = g(\theta_{i_h}(\boldsymbol{i}_h))$, while under Λ_j and $\Lambda_{j'}$, the penalties on i_h are $g(c_{G[j]}(\Lambda_j, i_h)) = g(\theta_{i_h}(\boldsymbol{j}_h))$ and $g(c_{G[j']}(\Lambda_{j'}, i_h)) = g(\theta_{i_h}(\boldsymbol{j'}_h))$, respectively.

Let $\gamma_h(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{j'})$ denote the amount of difference $g(\theta_{i_h}(\boldsymbol{i}_h)) - g(\theta_{i_h}(\boldsymbol{j}_h)) - g(\theta_{i_h}(\boldsymbol{j'}_h))$, which can be computed in constant time for a given triple $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{j'}$. Since there may exist more than one combination of $\boldsymbol{j}$ and $\boldsymbol{j'}$ satisfying the above condition, we take the minimum among such pairs of $\boldsymbol{j}$ and $\boldsymbol{j'}$. Let $P(\boldsymbol{i})$ be the set of pairs $(\boldsymbol{j}, \boldsymbol{j'})$ of two $|X_i|$-dimensional vectors $\boldsymbol{j}$ and $\boldsymbol{j'}$ such that $\boldsymbol{i}_h = \boldsymbol{j}_h + \boldsymbol{j'}_h$ for $1 \leq h \leq |X_i|$. If $P(\boldsymbol{i}) \neq \emptyset$, we set

$$OPT(i,\boldsymbol{i}) = \min_{(\boldsymbol{j},\boldsymbol{j'}) \in P(\boldsymbol{i})} \{OPT(j,\boldsymbol{j}) + OPT(j',\boldsymbol{j'}) + \sum_{1 \leq h \leq |X_i|} \gamma_h(\boldsymbol{i},\boldsymbol{j},\boldsymbol{j'})\}.$$

Since the penalty function p is linearly separable, $\gamma_h(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{j'})$ can be computed in constant time for each triple $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{j'}$. The size $|P(\boldsymbol{i})|$ of $P(\boldsymbol{i})$ is at most $O(\Delta^{\tau+1})$, since each $\boldsymbol{i}_h \leq \Delta$ for $1 \leq h \leq |X_i| \leq \tau + 1$. Then for each pair of $(\boldsymbol{j}, \boldsymbol{j'}) \in P(\boldsymbol{i})$, the formula (inside "min") $OPT(j,\boldsymbol{j}) + OPT(j',\boldsymbol{j'}) + \sum_{1 \leq h \leq |X_i|} \gamma_h(\boldsymbol{i},\boldsymbol{j},\boldsymbol{j'})$ can be computed in $O(|X_i|) = O(\tau)$ time, where the most time consuming part is to compute $\sum_{1 \leq h \leq |X_i|} \gamma_h(\boldsymbol{i},\boldsymbol{j},\boldsymbol{j'})$. Hence for an outdegree vector $\boldsymbol{i}$, it takes $O(\tau\Delta^{\tau+1})$ time to set the above recursive formula. As the remainder of the cases, if $P(\boldsymbol{i}) = \emptyset$, then we set

$$OPT(i,\boldsymbol{i}) = +\infty,$$

which takes constant time.

Since $|D(G, X_i)| = O(\Delta^{\tau+1})$ and it takes $O(\tau\Delta^{\tau+1})$ time for each case, the recursive formula for a join node i can be computed in $O(\tau\Delta^{2\tau+2})$ time in total:

Claim 8. *Let i be a join node whose two children are j and j'. Then, for $\boldsymbol{i} \in D(G, X_i)$,*

$$OPT(i, \boldsymbol{i}) = \begin{cases} \min_{(\boldsymbol{j}, \boldsymbol{j'}) \in P(\boldsymbol{i})} \{OPT(j, \boldsymbol{j}) + OPT(j', \boldsymbol{j'}) + \sum_{1 \le h \le |X_i|} \gamma_h(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{j'})\} & \text{if } P(\boldsymbol{i}) \neq \emptyset \text{ and} \\ +\infty \quad \text{otherwise}, \end{cases} \tag{14}$$

where $P(\boldsymbol{i})$ is the set of pairs $(\boldsymbol{j}, \boldsymbol{j'})$ of two $|X_i|$-dimensional vectors $\boldsymbol{j}$ and $\boldsymbol{j'}$ such that $\boldsymbol{i}_h = \boldsymbol{j}_h + \boldsymbol{j}'_h$ for $1 \le h \le |X_i|$, and $\gamma_h(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{j'}) = g(\theta_{i_h}(\boldsymbol{i}_h)) - g(\theta_{i_h}(\boldsymbol{j}_h)) - g(\theta_{i_h}(\boldsymbol{j}'_h))$. The above recursive formula can be computed in $O(\tau \Delta^{\tau+1})$ time for each $\boldsymbol{i}$, and hence the total time required for a join node i is $O(\tau \Delta^{2\tau+2})$. □

3.2.6. Running time

The optimal penalty can be obtained by computing $OPT(r, ())$ in the bottom up manner from leaves to the root r. For each node i, the recursive formula can be computed in $O(\tau \Delta^{2\tau+2})$ time, where the case that i is a join node spends the longest time. Since the number of nodes in a nice tree decomposition of a graph G with treewidth τ is $O(\tau n)$, we have the next theorem.

Theorem 9. *For an input graph, suppose that its nice tree decomposition with treewidth τ and $O(\tau n)$ nodes is given. Then, for any linearly separable penalty function, MPDCO$_p$ can be solved in $O(\tau^2 \Delta^{2\tau+2} n)$ time.*

3.3. W[1]-hardness with respect to treewidth

The algorithm presented in the previous section is an FPT algorithm with respect to $\tau + \Delta$, where τ and Δ are treewidth and the maximum degree. Then, a new question arises: is it possible to design an FPT algorithm parameterized only by τ? As we see in this section, the answer is likely "no"; we show that the next theorem, i.e., MPDCO$_p$ is $W[1]$-hard with respect to τ.

Theorem 10. *MPDCO$_p$ with a step (or concave) function parameterized by treewidth is $W[1]$-hard.*

We show this theorem by a reduction from k-Multicolor Clique:

Problem: k-Multicolor Clique
Input: A connected undirected graph $G = (V[1] \cup V[2] \cup \cdots \cup V[k], E)$, where the vertices of $V[i]$ induce an independent set for every i, $1 \le i \le k$.
Question: Is there a clique of size k (k-clique) in G?

The problem k-Multicolor Clique is known to be $W[1]$-hard with respect to the solution (clique) size k [32]. Actually, the proof utilizes the same graph of the reduction from k-Multicolor Clique to Capacitated Vertex Cover shown in [33], though we give the detail of the construction of the graph below for completeness.

From an instance G of k-Multicolor Clique, we construct a new graph H, a cost function c_H, and a penalty function p satisfying the followings: (i) G has a clique of size k if and only if H has an orientation Λ whose penalty $p(c_H(\Lambda))$ is at most $k(k+1)/2$, (ii) the treewidth of H is $O(k^3)$. As mentioned above, our H is identical to that in Section 3.2 of [33], where it is claimed that the treewidth of H is $O(k^3)$. This leads to the W[1]-hardness of MPDCO$_p$ with respect to the treewidth, based on the W[1]-hardness of k-Multicolor Clique with respect to k. Thus we prove only (i) below. We first give the reduction, and then show the equivalence by Lemmas 11 and 12.

The graph H

We construct H for $G = (V[1] \cup V[2] \cup \cdots \cup V[k], E)$, where E is partitioned into $E[i, j]$'s of $E[i, j] = \{\{u, u'\} \in E \mid u \in V[i], u' \in V[j]\}$ for $1 \le i < j \le k$. First we assume $k \ge 3$, since 2-Multicolor Clique is trivial. Also, for some integers ℓ and ℓ', we can assume that $|V[i]| = \ell$ for all i and $|E[i, j]| = \ell'$ for all i and j, since adding an isolated vertex and/or adding an edge with two vertices which are not incident to any other edges does not change the existence of k-clique in G when $k \ge 3$.

We give a vertex in $V[i]$ for each i a unique number between 1 and ℓ as an identifier in $V[i]$. In the proof below, a symbol to represent a vertex in $V[i]$ (e.g., u) also represents its unique identification number between 1 and ℓ. We prepare a vertex gadget for each $V[i]$, an edge gadget for each $E[i, j]$, and incidence gadgets that connect the vertex and edge gadgets according to their incidence relations, and H consists of these three types of gadgets. For two vertices u (left) and u' (right) in some gadget, we may add an identifier component $I(\alpha, \beta)$, which consists of α paths with length 2 and β pendant vertices attached with u' (see Fig. 9). The vertices between the left and right vertices are called *intermediate vertices*. Note that the construction of an identifier component is additive. For example, if u' is a right part of two identifier components $I(\alpha, \beta)$ and $I(\alpha', \beta')$, u' has $\beta + \beta'$ pendant vertices in total.

We now define the vertex gadget for $V[i]$. A vertex gadget has three types of components, one representative vertex, original vertices, and connector vertices. The representative vertex of $V[i]$ is $\hat{v}_i$. The original vertices correspond to the

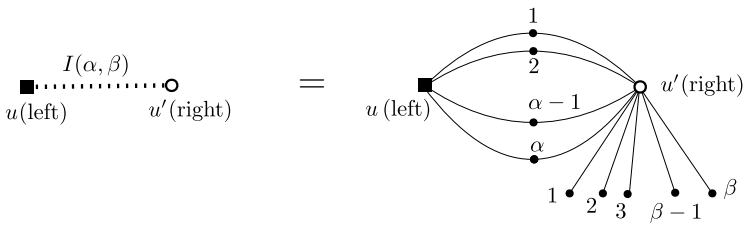

Fig. 9. (Left) Identifier component $I(\alpha, \beta)$ and (Right) its actual structure.

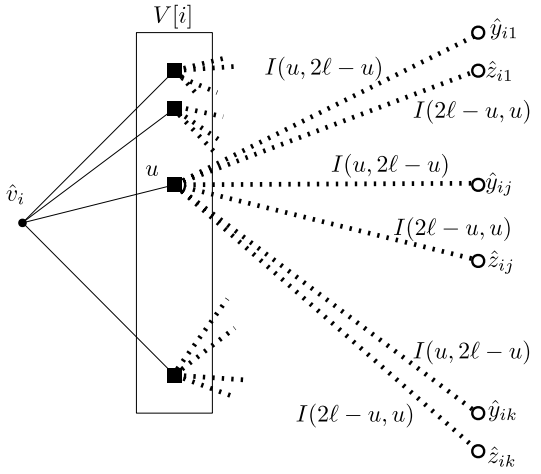

Fig. 10. Vertex gadget for $V[i]$.

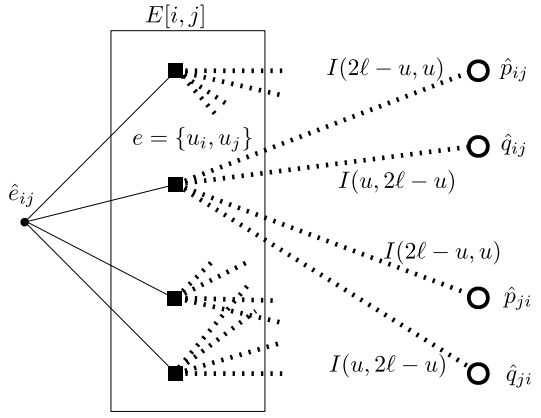

Fig. 11. Edge gadget for $E[i, j]$.

vertices in $V[i]$ of G. For original vertices, we use the same symbols of G. The connector vertices are $\hat{y}_{ij}$'s and $\hat{z}_{ij}$'s for every $j(\neq i)$. The representative vertex $\hat{v}_i$ is adjacent to all the original vertices. An original vertex u is connected with a connector vertex via an identifier component; for every $j(\neq i)$, u and $\hat{y}_{ij}$ (resp., $\hat{z}_{ij}$) are connected by $I(u, 2\ell - u)$ (resp., $I(2\ell - u, u)$), where u is left and $\hat{y}_{ij}$ (resp., $\hat{z}_{ij}$) is right in the identifier component. Fig. 10 shows an example of a vertex gadget.

We next define the edge gadget for $E[i, j]$. An edge gadget also has three types of components, one representative vertex, edge vertices, and connector vertices. The representative vertex of $E[i, j]$ is $\hat{e}_{i,j}$. The edge vertices correspond to the edges in $E[i, j]$ of G. The connector vertices are four vertices $\hat{p}_{ij}$, $\hat{q}_{ij}$, $\hat{p}_{ji}$, and $\hat{q}_{ji}$. The representative vertex $\hat{e}_{i,j}$ is adjacent to all the edge vertices. For edge $e = \{u, u'\}$ with $u \in V[i]$ and $u' \in V[j]$ in $E[i, j]$ of G, the corresponding edge vertex e is connected with connector vertices $\hat{p}_{ij}$, $\hat{q}_{ij}$, $\hat{p}_{ji}$, and $\hat{q}_{ji}$ via identifier component $I(2\ell - u, u)$, $I(u, 2\ell - u)$, $I(2\ell - u', u')$, and $I(u', 2\ell - u')$, respectively, where e is left in the identifier components. Fig. 11 shows an example of an edge gadget.

Fig. 12. Incidence gadget.

We finally define an incidence gadget, which consists of vertices $\hat{r}_{ij}$'s and $\hat{s}_{ij}$'s for all pairs of i and $j(\neq i)$. A vertex $\hat{r}_{ij}$ is connected with $\hat{y}_{ij}$ and $\hat{p}_{ij}$ via two identifier components $I(2\ell,0)$'s, and similarly, a vertex $\hat{s}_{ij}$ is connected with $\hat{z}_{ij}$ and $\hat{q}_{ij}$ via two identifier components $I(2\ell,0)$'s. Fig. 12 shows how an incidence gadget is.

Degree constraints

We now specify the degree constraints. First we set the degree constraints for the vertices (except for left and right vertices) in identifier components as follows. Although identifier components are included in vertex, edge, and incidence gadgets, we use the same degree constraints for every intermediate (or pendant) vertex:

- intermediate vertex v: $a_v = b_v = 1$
- pendant vertex v: $a_v = b_v = 0$

Let us describe the degree constraints for other vertices than these intermediate and pendant vertices. We set the degree constraints of the vertices in the vertex gadget for $V[i]$ as follows:

- representative vertex $\hat{v}_i$: $a_{\hat{v}_i} = b_{\hat{v}_i} = \ell - 1$
- original vertex v: $a_v = b_v = 0$
- connector vertex $\hat{y}_{ij}$ for $j \neq i$: $a_{\hat{y}_{ij}} = b_{\hat{y}_{ij}} = 2\ell^2$
- connector vertex $\hat{z}_{ij}$ for $j \neq i$: $a_{\hat{z}_{ij}} = b_{\hat{z}_{ij}} = 2\ell^2$

Next we set the degree constraints of the vertices in the edge gadget for $E[i, j]$ as follows:

- representative vertex $\hat{e}_{i,j}$: $a_{\hat{e}_{i,j}} = b_{\hat{e}_{i,j}} = \ell' - 1$
- edge vertex e: $a_e = b_e = 0$
- connector vertex $\hat{p}_{ij}$: $a_{\hat{p}_{ij}} = b_{\hat{p}_{ij}} = 2\ell\ell'$
- connector vertex $\hat{q}_{ij}$: $a_{\hat{q}_{ij}} = b_{\hat{q}_{ij}} = 2\ell\ell'$
- connector vertex $\hat{p}_{ji}$: $a_{\hat{p}_{ji}} = b_{\hat{p}_{ji}} = 2\ell\ell'$
- connector vertex $\hat{q}_{ji}$: $a_{\hat{q}_{ji}} = b_{\hat{q}_{ji}} = 2\ell\ell'$

Lastly we set the degree constraints for the vertices in the incident gadgets for every pair of i and $j(\neq i)$:

- vertex $\hat{r}_{ij}$: $a_{\hat{r}_{ij}} = b_{\hat{r}_{ij}} = 2\ell$
- vertex $\hat{s}_{ij}$: $a_{\hat{s}_{ij}} = b_{\hat{s}_{ij}} = 2\ell$

Penalty function

The remainder of the reduction is to choose a penalty function. We let the penalty function p be the summation of a step function g such that $g(x) = 0$ for $x = 0$ and $g(x) = 1$ for $x > 0$. This function g is a step function and also a concave function. This completes the reduction.

Lemmas

In the following, we show that G has a clique of size k if and only if H has an orientation Λ such that $p(c_h(\Lambda)) \leq k(k+1)/2$ by Lemmas 11 and 12. Theorem 10 follows from these two lemmas and the fact that the treewidth of H is $O(k^3)$.

Lemma 11. *If G has a clique of size k, then H has an orientation whose penalty is at most $k(k+1)/2$.*

Proof. Let $C = \{v_1, v_2, \ldots, v_k\}$ be a clique of size k in G, where we assume that $v_i \in V[i]$ without loss of generality. We construct an orientation of H whose penalty is at most (more precisely, equal to) $k(k+1)/2$ from C. First we orient the edges which are incident to the original vertex v_i in the vertex gadget for $V[i]$ outward from v_i for $1 \leq i \leq k$. After that similarly for edges which are incident to the edge vertex $e = \{v_i, v_j\}$ for every pair of $v_i, v_j \in C$, we orient them outward from e. By these, the degree constraints of these vertices corresponding to C and its edges in G are violated, and the total penalty on these vertices is $k + k(k-1)/2 = k(k+1)/2$. Since the penalty is already $k(k+1)/2$ at this moment, we need to orient the other edges so as to satisfy the degree constraints of all the remaining vertices. In below, we describe such an orientation.

First of all, every edge incident to a pendant vertex in identifier components is oriented toward the pendant vertex, since every pendant vertex must have outdegree 0 by its degree constraints. We orient the edges in the vertex gadget for $V[i]$ as follows.

- For the representative vertex $\hat{v}_i$, we orient the edges incident to $\hat{v}_i$ outward from $\hat{v}_i$, except for the edge $\{\hat{v}_i, v_i\}$ (which is already oriented as $(v_i, \hat{v}_i)$ in the above). The outdegree of $\hat{v}_i$ becomes $\ell - 1$ since $|V[i]| = \ell$, and satisfies the degree constraint for $\hat{v}_i$.
- For any original vertex $v \in V[i] \setminus \{v_i\}$, we orient all the edges incident to v toward v. (Note that this does not contradict the orientation $(\hat{v}_i, v)$ of the edge $\{\hat{v}_i, v\}$ already determined in the above.) These original vertices have outdegree 0 which satisfies their degree constraints.
- For any intermediate vertex v between an original vertex u and $\hat{y}_{ij}$ (or $\hat{z}_{ij}$), the edge $\{u, v\}$ is already oriented as (u, v) for $u = v_i$ or (v, u) for $u \neq v_i$. In order to satisfy the degree constraint of v, i.e., $a_v = b_v = 1$, the orientation of the edge $\{v, \hat{y}_{ij}\}$ (or $\{v, \hat{z}_{ij}\}$) is uniquely determined.

The above guarantees that the penalties on representative vertex, original vertices, and intermediate vertices in a vertex gadget are 0. On the other hand, the outdegrees of $\hat{y}_{ij}$'s and $\hat{z}_{ij}$'s have not been determined only by the above, although the number of edges already oriented outward from $\hat{y}_{ij}$ (or $\hat{z}_{ij}$) is $2\ell(\ell - 1) + 2\ell - v_i = 2\ell^2 - v_i$ (or $2\ell^2 - 2\ell + v_i$); the outdegree of those vertices will be determined after we decide orientations for the edges in incidence gadgets.

Similar to the vertex gadgets, we orient the edges in the edge gadget for $E[i, j]$ as follows.

- For the representative vertex $\hat{e}_{i,j}$, we orient the edges incident to $\hat{e}_{i,j}$ outward from $\hat{e}_{i,j}$, except for the edge $\{\hat{e}_{i,j}, \{v_i, v_j\}\}$ (which is already oriented as $(\{v_i, v_j\}, \hat{e}_{i,j})$ in the above). The outdegree of $\hat{e}_{i,j}$ becomes $\ell' - 1$ since $|E[i, j]| = \ell'$, and satisfies the degree constraint for $\hat{e}_{i,j}$.
- For any edge vertex $e \in E[i, j] \setminus \{\{v_i, v_j\}\}$, we orient all the edges incident to e toward e. (Note that this does not contradict the orientation $(\hat{e}_{i,j}, e)$ of the edge $\{\hat{e}_{i,j}, e\}$ already determined in the above.) These edge vertices have outdegree 0 which satisfies their degree constraints.
- For any intermediate vertex v between an edge vertex e and $\hat{p}_{ij}$ ($\hat{q}_{ij}$, $\hat{p}_{ji}$, or $\hat{q}_{ji}$), the edge $\{e, v\}$ is already oriented as (e, v) for $e = \{v_i, v_j\}$ or (v, e) for $e \neq \{v_i, v_j\}$. In order to satisfy the degree constraint of v, i.e., $a_v = b_v = 1$, the orientation of the edge $\{v, \hat{p}_{ij}\}$ ($\hat{q}_{ij}$, $\hat{p}_{ji}$, or $\hat{q}_{ji}$) is uniquely determined.

By the above, the numbers of outgoing edges of $\hat{p}_{ij}$ (or $\hat{p}_{ji}$) and $\hat{q}_{ij}$ (or $\hat{q}_{ji}$) are $2\ell\ell' - 2\ell + v_i$ and $2\ell\ell' - v_i$, respectively.

Let us look at an incidence gadget between $\hat{y}_{ij}$ and $\hat{p}_{ij}$. By the above, $\hat{y}_{ij}$ already has $2\ell^2 - v_i$ outgoing edges. Thus, to meet the degree constraint, v_i edges between $\hat{y}_{ij}$ and $\hat{r}_{ij}$ in the identifier component are oriented toward $\hat{r}_{ij}$, while the other $2\ell - v_i$ edges are oriented toward $\hat{y}_{ij}$. Similarly, since $\hat{p}_{ij}$ already has $2\ell\ell' - 2\ell + v_i$ outgoing edges, $2\ell - v_i$ edges between $\hat{p}_{ij}$ and $\hat{r}_{ij}$ are oriented toward $\hat{r}_{ij}$, while other v_i edges are oriented toward $\hat{p}_{ij}$. By these orientations of the edges in the incidence gadget, $\hat{r}_{ij}$ has outdegree 2ℓ, which satisfies the degree constraint for $\hat{r}_{ij}$.

As for an incidence gadget between $\hat{z}_{ij}$ and $\hat{q}_{ij}$, similar arguments to the above indicate that $2\ell - v_i$ edges and v_i edges between $\hat{z}_{ij}$ and $\hat{s}_{ij}$ are oriented toward $\hat{s}_{ij}$ and toward $\hat{z}_{ij}$, respectively, and then $2\ell - v_i$ edges and v_i edges between $\hat{q}_{ij}$ and $\hat{s}_{ij}$ are oriented toward $\hat{q}_{ij}$ and toward $\hat{s}_{ij}$, respectively. Hence, all of $\hat{z}_{ij}$, $\hat{q}_{ij}$, and $\hat{s}_{ij}$ satisfy their degree constraints. The discussion for incidence gadgets between $\hat{y}_{ji}$'s and $\hat{p}_{ji}$'s or between $\hat{z}_{ji}$'s and $\hat{q}_{ji}$'s are similar. This completes the proof. □

Lemma 12. *If H has an orientation whose penalty is at most $k(k+1)/2$, G has a clique of size k.*

Proof. Let Λ be the orientation of H, whose penalty is at most $k(k+1)/2$, that is, at most $k(k+1)$ vertices violate the degree constraints under Λ. We can assume that all the representative vertices satisfy their degree constraints under Λ by the following reason. If a representative vertex $\hat{v}_i$ violates the degree constraint under Λ, its outdegree is either ℓ or at most $\ell - 2$ since $a_{\hat{v}_i} = b_{\hat{v}_i} = \ell - 1$ and $d_H(\hat{v}_i) = \ell$. In the former case, even if all the original vertices in $V[i]$ satisfy the degree constraints (i.e., their outdegrees are all 0), we choose one $u \in V[i]$ and flip $(\hat{v}_i, u)$ to $(u, \hat{v}_i)$. Then $\hat{v}_i$ gets to satisfy the degree constraint instead of u and u violates its degree constraint, which does not increase the total penalty. In the latter case, we suppose that the outdegree of $\hat{v}_i$ is $\ell - \gamma$, where $2 \leq \gamma \leq \ell$, which implies that at least γ vertices in $V[i]$ violate their degree constraints. We arbitrarily choose $\gamma - 1$ vertices from them and flip the edges between $\hat{v}_i$ and them. As a result $\hat{v}_i$ gets to satisfy the degree constraint, and each of those $\gamma - 1$ vertices still violates its degree constraint or also gets to satisfy its degree constraint. Thus we could make the representative vertex $\hat{v}_i$ satisfy its degree constraint without increasing the number of vertices violating the degree constraints, i.e., without increasing the total penalty. Since a similar argument holds for a representative vertex $\hat{e}_{i,j}$ in each edge gadget $E[i, j]$, we can assume that every representative vertices in the vertex gadgets and the edge gadgets satisfy their degree constraints under Λ.

For each vertex gadget for $V[i]$, the representative vertex $\hat{v}_i$ is assumed to satisfy its degree constraint $a_{\hat{v}_i} = b_{\hat{v}_i} = \ell - 1$ under Λ as mentioned above. Since $d_H(\hat{v}_i) = \ell$, exactly one edge between $\hat{v}_i$ and original vertices is oriented outward from $\hat{v}_i$. Thus at least one original vertex u_i in each vertex gadget for $V[i]$ violates its degree constraint $a_{u_i} = b_{u_i} = 0$, where other original vertices than u_i may also have outdegree at least one. Similarly, at least one edge vertex in each edge gadget violates its degree constraint. Since the number of the vertex and the edge gadgets is $k + k(k-1)/2 = k(k+1)/2$, at least

$k(k+1)/2$ vertices violate their degree constraints under Λ. However, we have assumed that the total penalty of Λ is at most $k(k+1)/2$. This implies that exactly one original vertex (resp., edge vertex) in a vertex gadget (resp., an edge gadget) violates the degree constraint and no other vertices in a vertex gadget (resp., an edge gadget) violate the degree constraints. In particular, every edge incident to a pendant vertex v in an identifier component is oriented toward v, since the degree constraint of v is $a_v = b_v = 0$. We also assume that such edges are oriented toward pendant vertices in Λ.

Let u_i (resp., $e_{i,j} = \{u'_i, u'_j\}$) be the vertex violating the degree constraints in the vertex gadget for $V[i]$ (resp., the edge gadget for $E[i, j]$). Since the vertices $V[i] \setminus \{u_i\}$ (resp., $E[i, j] \setminus \{e_{i,j}\}$) in a vertex gadget (resp. an edge gadget) satisfy the degree constraints, all the edges incident to them are oriented toward them. By the degree constraints of intermediate vertices (they must have outdegree one) and pendant vertices (they must have outdegree zero) in an identifier component, the above orientation for representative vertices, original vertices, and edge vertices determines orientation of edges in identifier components between $V[i] \setminus \{u_i\}$ and $\hat{y}_{ij}$ (or $\hat{z}_{ij}$), and also edges in between $E[i, j] \setminus \{e_{i,j}\}$ and $\hat{p}_{ij}$ (or $\hat{q}_{ij}$, $\hat{p}_{ji}$, $\hat{q}_{ji}$). The numbers of these (already determined) outgoing edges from $\hat{y}_{ij}$ and $\hat{z}_{ij}$ are $2\ell^2 - u_i$ and $2\ell^2 - (2\ell - u_i)$, respectively, where edges incident to pendant vertices are already oriented. Similarly, the number of (already determined) outgoing edges from $\hat{p}_{ij}$, $\hat{q}_{ij}$, $\hat{p}_{ji}$, and $\hat{q_{ji}}$ are $2\ell(\ell' - 1) + u'_i$, $2\ell\ell' - u'_i$, $2\ell(\ell' - 1) + u'_i$, and $2\ell\ell' - u'_i$ respectively. In the following, we discuss about the remaining edges, i.e., paths between u_i and $e_{i,j}$ via connector vertices and identifier components.

For an identifier component $I' = I(\alpha, \beta)$ between a left vertex v and a right vertex v', if we decide to orient γ edges ($0 \le \gamma \le \alpha$) between v and intermediate vertices in I' outward from v, it determines orientation of the other unoriented edges in I' to meet the degree constraints: $\alpha - \gamma$ edges between v and intermediate vertices in I' are oriented toward v, and then γ (or $\alpha - \gamma$) edges between v' and intermediate vertices in I' are oriented toward v' (or outward from v'), where all edges incident to pendant vertices are already oriented outward from v' in the above. We say this process *γ-propagation from v to v'* in the following. Note that we have seen 0-propagation in the above, say, from an original vertex u_i to $\hat{y}_{ij}$ in a vertex gadget.

We here focus on a path from u_i to $e_{i,j}$ via $\hat{y}_{ij}$, $\hat{r}_{ij}$, $\hat{p}_{ij}$, and identifier components. Let γ be the number of edges oriented from u_i to intermediate vertices in the identifier component I' between u_i and $\hat{y}_{ij}$, where $0 \le \gamma \le u_i$. Then γ-propagation from u_i to $\hat{y}_{ij}$ orients $u_i - \gamma$ (or γ) edges in I' outward from $\hat{y}_{ij}$ (or toward $\hat{y}_{ij}$). Since $\hat{y}_{ij}$ already has other $2\ell^2 - u_i$ outgoing edges mentioned above, total number of outgoing edges from $\hat{y}_{ij}$ so far is $(2\ell^2 - u_i) + (u_i - \gamma) = 2\ell^2 - \gamma$. In order to meet the degree constraint of $\hat{y}_{ij}$, $a_{\hat{y}_{ij}} = b_{\hat{y}_{ij}} = 2\ell^2$, γ more edges should be oriented outward from $\hat{y}_{ij}$. Thus γ-propagation from $\hat{y}_{ij}$ to $\hat{r}_{ij}$ occurs, and then it causes another γ-propagation from $\hat{r}_{ij}$ to $\hat{p}_{ij}$, which increases the outdegree of $\hat{p}_{ij}$ by $2\ell - \gamma$. Together with the already oriented $2\ell(\ell' - 1) + u'_i$ outgoing edges, $\hat{r}_{ij}$ has outdegree at least $(2\ell(\ell' - 1) + u'_i) + (2\ell - \gamma) = 2\ell\ell' + u'_i - \gamma$. Since $\hat{r}_{ij}$ satisfies the degree constraint, $u'_i - \gamma \le 0$ must hold. This implies that $u'_i \le \gamma \le u_i$.

Similarly, we focus on a path from u_i to $e_{i,j}$ via $\hat{z}_{ij}$, $\hat{s}_{ij}$, $\hat{q}_{ij}$, and identifier components. Recall that the numbers of the predetermined outgoing edges from $\hat{z}_{ij}$ and $\hat{q}_{ij}$ are $2\ell^2 - (2\ell - u_i)$ and $2\ell\ell' - u'_i$, respectively. Let γ' be the number of edges oriented from u_i to intermediate vertices in the identifier component between u_i and $\hat{z}_{ij}$, where $0 \le \gamma' \le 2\ell - u_i$. By a similar argument to the above, $2\ell - \gamma'$ edges are oriented from $\hat{q}_{ij}$ to intermediate vertices in an identifier component between $\hat{s}_{ij}$ and $\hat{q}_{ij}$. Hence $\hat{q}_{ij}$ has outdegree at least $(2\ell\ell' - u'_i) + (2\ell - \gamma')$. Since $\hat{q}_{ij}$ satisfies the degree constraint $a_{\hat{q}_{ij}} = b_{\hat{q}_{ij}} = 2\ell\ell'$, $2\ell - \gamma' - u'_i \le 0$ must hold. This implies that $u_i \le 2\ell - \gamma' \le u'_i$. Combining this with above $u'_i \le u_i$, we have $u_i = u'_i$.

Since $u_j = u'_j$ holds by a similar argument, an edge vertex $e_{i,j} = \{u'_i, u'_j\}$ violating its degree constraint under Λ corresponds to the edge between u_i and u_j in G, both of which violate the degree constraints in $\Lambda(H)$. These imply that vertices violating the degree constraints under Λ for H correspond to a clique of size k in G. This completes the proof. □

4. Edge-weighted graphs

In the previous sections, the problem MPDCO$_p$ is defined on unweighted graphs. As a generalization, we can define the weighted version of MPDCO$_p$ (MPDCO$_{weighted}$ for short), in which the input is a weighted graph $G = (V, E, w)$, where V and E are the set of vertices and edges, respectively, and w is a weight function that assigns a weight (a nonnegative integer) to each edge. In this problem variant, the outdegree of a vertex is defined as the sum of weights of outgoing arcs incident to it in MPDCO$_{weighted}$. Let W denote the maximum possible outdegree, i.e., $W = \max_{v \in V}\{\sum_{u \in N_G(v)} w(\{v, u\})\}$.

First, we show an inapproximability result for MPDCO$_{weighted}$ for planar bipartite graphs in Section 4.1. Then Section 4.2 extends the algorithms in Sections 3.1 and 3.2 for MPDCO$_{weighted}$ on trees and graphs with bounded treewidth. Lastly, in Section 4.3, we restrict our attention to stars, a rather restricted subclass of trees and planar bipartite graphs, and then show some tractability and intractability results.

4.1. Planar bipartite graphs

In Theorem 1, we show that it is strongly NP-hard to approximate the unweighted version MPDCO$_p$ within a ratio of $2 - o(1)$ for concave and step functions. In this section, we show much stronger inapproximability of MPDCO$_{weighted}$; it is impossible to approximate MPDCO$_{weighted}$ within a ratio of any polynomial-time computable function unless $P = NP$, although we can solve the problem in polynomial time for unweighted graphs with convex functions as in Section 2.

First we show the next theorem.

Theorem 13. *It is strongly NP-hard to distinguish $OPT(G) = 0$ and $OPT(G) > 0$ for MPDCO$_{weighted}$ with a convex, concave, or step function on weighted planar bipartite graphs.*

Proof. We give a reduction from the decision version of MinMaxO with target value k, i.e., checking whether there is an orientation of G such that the maximum weighted outdegree is at most k. This problem is strongly NP-hard for weighted planar bipartite graphs [13].

Let (G, k) be an instance of the decision version of MinMaxO, i.e., G is a weighted planar bipartite graph and k is a nonnegative integer at least 2. We use the graph G itself as a part of the instance of MPDCO$_{weighted}$. Then we set $a_v = 0$ and $b_v = k$ for every vertex $v \in V(G)$, and choose a function g in the penalty function satisfying the condition that $g(x) > 0$ if and only if $x > 0$. Note that many convex, concave, and step functions satisfy this condition for g. This reduction is done in polynomial time.

Let $OPT(G)$ be the total penalty for G in the problem MPDCO$_{weighted}$. It is easy to see that there is an orientation for G in which the maximum outdegree of a vertex is at most k if and only if $OPT(G) = 0$. Namely, MPDCO$_{weighted}$ contains the decision version of MinMaxO as a subproblem. Thus, the strong NP-hardness of MPDCO$_{weighted}$ on weighted planar bipartite graphs is implied by the strong NP-hardness of the decision version of MinMaxO on planar bipartite graphs. □

Based on the above theorem, we show the next corollary.

Corollary 14. *Suppose that $\rho(n) \geq 1$ is any polynomial-time computable function. Then, there is no polynomial-time $\rho(n)$-approximation algorithm for MPDCO$_{weighted}$ on weighted planar bipartite graphs whose penalty function is linearly separable and convex (concave, or step) unless $P = NP$.*

Proof. For the purpose of obtaining a contradiction, assume that there exists a polynomial-time $\rho(n)$-approximation algorithm ALG for MPDCO$_{weighted}$. The algorithm ALG finds an orientation of G having total penalty $ALG(G)$ such that $OPT(G) \leq ALG(G) \leq \rho(n) \cdot OPT(G)$, where $OPT(G)$ is the minimum total penalty for G. Hence, one can determine whether $OPT(G) > 0$ or $OPT(G) = 0$ in polynomial time using ALG based on the observation that $ALG(G) > 0$ if and only if $OPT(G) > 0$. This contradicts Theorem 13. The corollary follows. □

4.2. Graphs with bounded treewidth

In the previous section, we showed the strong NP-hardness of MPDCO$_{weighted}$ on planar bipartite graphs. A typical subclass of planar bipartite graphs is trees. In Section 3, we designed a polynomial-time algorithm for MPDCO$_p$ on unweighted trees. In this section, we first extend it to a pseudo-polynomial-time algorithm for MPDCO$_{weighted}$ on weighted trees.

Let us consider the algorithm for trees in Section 3.1. The computation of (4) could be done using a sorting algorithm for the unweighted version MPDCO$_p$. For MPDCO$_{weighted}$, we need to find a subset $N' \subseteq N_T(r)$ such that $\sum_{v \in N'} w(\{r, v\}) = k$ and $\sum_{s \in N'} h(T_s)$ is minimum. Using an algorithm for the KNAPSACK problem [34], we can find such N' in time $O(|N_G(r)| \cdot W)$: Given a sequence of pairs (x_i, y_i) of nonnegative integers for $1 \leq i \leq z$ and a value B, KNAPSACK asks to find $X' \subseteq X$ such that $\sum_{x_i \in X'} x_i \leq B$ and $\sum_{x_i \in X'} y_i$ is the maximum among all such subsets. The algorithm in [34] runs in $O(zB)$ time, and finds a subset X' satisfying $\sum_{x_i \in X'} x_i = B'$ for every $1 \leq B' \leq B$, which maximizes $\sum_{x_i \in X'} y_i$. Thus, replacing $h(T_s)$ with $C - h(T_s)$ for a large value C such that $C \geq \max_{s \in N_T(r)} h(T_s)$, the algorithm in [34] can obtain a subset $N' \subseteq N_T(r)$ such that $\sum_{v \in N'} w(\{r, v\}) = k$ and $\sum_{s \in N'} h(T_s)$ is minimum, which takes $O(|N_G(r)| \cdot k) = O(|N_G(r)| \cdot W)$ time. In practice, we do not need to compute once for every k; only spending $O(|N_G(r)| \cdot W)$ time for the case $k = W$, we can construct a table which stores the information on the relation between k and the value $\min_{N' \subseteq N_T(r), |N'| = k}\{\sum_{s \in N'} h(T_s)\}$. Then we can refer entries in the table for each k in constant time.

If $\sum_{s \in N_T(r)} q^+(T_s)$ and $\min\{\sum_{s \in N'} h(T_s)\}$ are given, (3) can be computed in constant time. Here, since $\sum_{s \in N_T(r)} q^+(T_s)$ is common for every k, we can compute it once in advance spending $O(|N_G(r)|)$ time. Since it takes $O(|N_G(r)|)$ time to compute $g(\theta_r(k)) + q(T, k)$ and $g(\theta_r(k+1)) + q(T, k)$ for each $k \in \{0, \ldots, W\}$, (1) and (2) can be computed in $O(|N_G(r)| \cdot W)$ time. In summary, for each k, it takes $O(|N_G(r) \cdot W|) + O(|N_G(r)|) + O(|N_G(r)| \cdot W) = O(|N_G(r) \cdot W|)$ time, where the first and the second terms in the left hand side are respectively for solving KNAPSACK and computing $\sum_{s \in N_T(r)} q^+(T_s)$ described in the above. Thus the estimation of the total running time in the equation (6) is modified as

$$\sum_{v \in V(G)} O(|N_G(v)| \cdot W) = O(Wn),$$

and we have the following theorem.

Theorem 15. *For any linearly separable penalty function, MPDCO$_{weighted}$ can be solved in $O(Wn)$ time when the input graph is a tree.*

Next, we consider the FPT algorithm parameterized by treewidth τ and the maximum degree Δ in Section 3.2. Since the outdegree of each vertex is at most W in MPDCO$_{weighted}$, the set $D(G, X_i)$ of outdegree vectors has size at most $O(W^{\tau+1})$,

although this size was $O(\Delta^{\tau+1})$ for the unweighted version MPDCO$_p$. The time complexity for each case is modified as follows.

- *The node i is a leaf node.*
  The initial formula $OPT(i,()) = 0$ for the unweighted version can be used as it is, and it can be computed in constant time, too. □
- *The node i is an introduce-vertex node.*
  The recursive formula (8) can be used as it is. Since $|D(G, X_i)| = O(W^{\tau+1})$, the total time increases to $O(\tau W^{\tau+1})$ from $O(\tau\Delta^{\tau+1})$. □
- *The node i is a forget node.*
  In the recursive formula (9), we need to take the minimum. For the first case in (9), the range of h is $0 \le h \le d_G(j_k)$, however in MPDCO$_{weighted}$, it is replaced with $0 \le h \le W$. For the second case in (9), the size of $J_k(\boldsymbol{i})$ increases to $O(W)$ from $O(\Delta)$ since $0 \le \boldsymbol{j}_k \le W$, and hence $J_k(\boldsymbol{i})$ can be obtained in $O(\tau W)$ time. Thus, for each $\boldsymbol{i} \in D(G, X_i)$, it takes $O(\tau W)$ time, and hence the total time needed increases to $O(\tau W^{\tau+2})$ from $O(\tau\Delta^{\tau+2})$. □
- *The node i is an introduce-edge node.*
  The crucial difference from before is that when orienting an edge $\{i_k, i_{k'}\}$, the increase of outdegree is $w(\{i_k, i_{k'}\})$ instead of one for the unweighted version MPDCO$_p$. Thus, we define $\boldsymbol{i}^{(\boldsymbol{k})}$ as an $|X_i|$-dimensional vector such that $\boldsymbol{i}^{(\boldsymbol{k})}_k = \boldsymbol{i}_k - w(\{i_k, i_{k'}\})$ and $\boldsymbol{i}^{(\boldsymbol{k})}_h = \boldsymbol{i}_h$ for $h \neq k$. The vector $\boldsymbol{i}^{(\boldsymbol{k'})}$ is defined similarly. Here, by checking $\boldsymbol{i}_k \ge w(\{i_k, i_{k'}\})$ (or $\boldsymbol{i}_{k'} \ge w(\{i_k, i_{k'}\})$), we can know whether $\boldsymbol{i}^{(\boldsymbol{k})}$ (or $\boldsymbol{i}^{(\boldsymbol{k'})}$) belongs to $D(G, X_j)$. (This differs from checking whether $\boldsymbol{i}_k$ and $\boldsymbol{i}_{k'}$ are positive for the unweighted version of the problem.) Therefore, to handle this case, the recursive formula (13) in Claim 7 is modified as follows.

$$OPT(i,\boldsymbol{i}) = \begin{cases} OPT(j,\boldsymbol{i}^{(\boldsymbol{k})}) - g(\theta_{i_k}(\boldsymbol{i}_k - w(\{i_k,i_{k'}\}))) + g(\theta_{i_k}(\boldsymbol{i}_k)) \\ \quad \text{if } \boldsymbol{i}_k \ge w(\{i_k,i_{k'}\}) \text{ and } \boldsymbol{i}_{k'} < w(\{i_k,i_{k'}\}), \\ OPT(j,\boldsymbol{i}^{(\boldsymbol{k'})}) - g(\theta_{i_{k'}}(\boldsymbol{i}_{k'} - w(\{i_k,i_{k'}\}))) + g(\theta_{i_{k'}}(\boldsymbol{i}_{k'})) \\ \quad \text{if } \boldsymbol{i}_k < w(\{i_k,i_{k'}\}) \text{ and } \boldsymbol{i}_{k'} \ge w(\{i_k,i_{k'}\}), \\ \min\{OPT(j,\boldsymbol{i}^{(\boldsymbol{k})}) - g(\theta_{i_k}(\boldsymbol{i}_k - w(\{i_k,i_{k'}\}))) + g(\theta_{i_k}(\boldsymbol{i}_k)), \\ \quad OPT(j,\boldsymbol{i}^{(k')}) - g(\theta_{i_{k'}}(\boldsymbol{i}_{k'} - w(\{i_k,i_{k'}\}))) + g(\theta_{i_{k'}}(\boldsymbol{i}_{k'}))\} \\ \quad \text{if } \boldsymbol{i}_k \ge w(\{i_k,i_{k'}\}) \text{ and } \boldsymbol{i}_{k'} \ge w(\{i_k,i_{k'}\}), \text{ and} \\ +\infty \\ \quad \text{otherwise.} \end{cases}$$

  The above can still be computed in $O(\tau)$ time for each $\boldsymbol{i} \in D(G, X_i)$. Since $|D(G, X_i)| = O(W^{\tau+1})$, the total time increases to $O(\tau W^{\tau+1})$ from $O(\tau\Delta^{\tau+1})$. □
- *The node i is a join node.*
  The recursive formula (14) can be used as it is, however the size of $P(\boldsymbol{i})$ increases to $O(W^{\tau+1})$ since the maximum outdegree of a vertex is at most W in MPDCO$_{weighted}$. Thus for each $\boldsymbol{i} \in D(G, X_i)$, the recursive formula (14) can be computed in $O(\tau W^{\tau+1})$, and hence the total time needed is $O(\tau W^{2\tau+2})$. □

From the above observations, the algorithm spends $O(\tau^2 W^{2\tau+2} n)$ time, since the number of nodes in a nice tree decomposition is $O(\tau n)$.

Theorem 16. *For an input graph, suppose that its nice tree decomposition with treewidth τ and $O(\tau n)$ nodes is given. Then, for any linearly separable penalty function, MPDCO$_{weighted}$ can be solved in $O(\tau^2 W^{2\tau+2} n)$ time.*

4.3. Stars

In the previous sections, we saw that MPDCO$_{weighted}$ is intractable for planar bipartite graphs although we could design pseudo-polynomial-time algorithms for graphs with bounded treewidth. A natural question is that whether we can design any polynomial-time algorithm for weighted trees or not. In this section, we investigate stars, which is more restricted class of graphs than trees. First we show that MPDCO$_{weighted}$ is weakly NP-hard even for stars with some restrictions on degree constraints, and then propose an algorithm to cope with those instances.

We only consider stars having at least three vertices in this section (since a star of two vertices has only one edge and an optimal orientation can be obtained easily by orienting the edge in either way). For a star G, a vertex incident to only one edge is called a *leaf*, and the vertex incident to at least two vertices is called the *center*.

For a weighted graph $G = (V, E, w)$ and degree constraints on its vertices of an instance of MPDCO$_{weighted}$, let $R(G)$ be the set of pairs of a lower bound and an upper bound of degrees for each vertex, i.e., $R(G) = \{(a_v, b_v) \mid v \in V\}$. The size

$|R(G)|$ of $R(G)$ represents the number of variations of the degree constraints. $OPT(G)$ denotes the optimal penalty for G. We define restricted sets $\mathcal{G}$ and $\mathcal{H}$ of edge-weighted stars based on $|R(G)|$. The set $\mathcal{G}$ contains every edge-weighted star G satisfying the following two conditions.

- $|R(G)| \leq 2$, i.e., $R(G) = \{(a_1, b_1), (a_2, b_2)\}$ for some (a_1, b_1) and (a_2, b_2) (possibly $(a_1, b_1) = (a_2, b_2)$).
- The maximum weight of an edge of G is at most $\max\{b_1, b_2\}$.

Similarly, the set $\mathcal{H} \subseteq \mathcal{G}$ contains every edge-weighted star G satisfying the following two conditions.

- $|R(G)| = 1$, i.e., $R(G) = \{(a, b)\}$ for some a and b.
- The maximum weight of an edge of G is at most b.

Even for these rather restricted instances $\mathcal{G}$ and $\mathcal{H}$, $MPDCO_{weighted}$ is weakly NP-hard.

Theorem 17.

(I) *It is weakly NP-hard to distinguish $OPT(G) = 0$ and $OPT(G) > 0$ for $MPDCO_{weighted}$ on $G \in \mathcal{G}$ with a convex, concave, or step function.*

(II) *Suppose that k is a positive number. It is weakly NP-hard to distinguish $OPT(G) = k$ and $OPT(G) > k$ for $MPDCO_{weighted}$ on $G \in \mathcal{H}$ with a concave or step function.*

Proof. We give reductions from the problem Partition, which is weakly NP-hard [35]. Given a set X of non-negative integers $x_1, x_2, \ldots, x_y$, where y is the number of the integers, Partition asks whether there is a set $X' \subseteq X$ satisfying $\sum_{x_i \in X'} x_i = \sum_{x_i \in X} x_i/2$. Let $B = \sum_{x_i \in X} x_i/2$. Note that we can assume that $\max_{1 \leq i \leq y}\{x_i\} < B$, since if $x_j > B$ then clearly x_j is not included in the solution X', and if $x_j = B$ for some j, then it can be easily found. Construct an instance of $\text{MPDCO}_{weighted}$, i.e., a star G, a penalty function, and degree constraints, from an instance of Partition as follows. We prepare the center r and its children (leaves) $v_1, v_2, \ldots, v_y$. The weight of an edge $\{r, v_i\}$ is set to x_i for $1 \leq i \leq y$.

Let us describe penalty functions and degree constraints for (I) and (II).

(I) The degree constraint is set to (B, B) for r, and $(0, B)$ for every v_i, where $R(G) = \{(B, B), (0, B)\}$. As the penalty function, we choose a non-decreasing function g such that $g(0) = 0$ and $g(x) > 0$ if $x > 0$, where g can be a convex function, a concave function, or a step function.

(II) The degree constraint is set to (B, B) for every vertex, where $R(G) = \{(B, B)\}$. As the penalty function, we choose a non-decreasing function g such that $g(0) = 0$ and $g(x) = k/y$ if $x > 0$, where g can be regarded as a concave function and also as a step function.

Clearly, the above reductions can be done in polynomial time. One important property of G is that the penalty of a leaf v_i is always 0 for (I) and k/y for (II) regardless of the orientation of the edge $\{r, v_i\}$, since $0 \leq x_i < B$.

We show that there is a set X' such that $\sum_{x_i \in X'} x_i = B$ if and only if $OPT(G) = 0$ for (I) (or $OPT(G) = k$ for (II)).

($\Rightarrow$) Suppose that there is a set X' such that $\sum_{x_i \in X'} x_i = B$. For each edge $\{r, v_i\}$, orient it as (r, v_i) if $x_i \in X'$, and (v_i, r) otherwise. Under this orientation, r has outdegree B and hence its penalty is 0. As mentioned above, every leaf has penalty 0 for (I) (or k/y for (II)). Thus the total penalty of this orientation is 0, i.e., $OPT(G) = 0$ for (I) (or $OPT(G) = k$ for (II)).

($\Leftarrow$) Assume there is an orientation of G with total penalty 0 for (I) (or k for (II)). Since the total penalty of this orientation is 0 for (I) (or k for (II)), the center r has outdegree B. Let E' be the set of edges oriented outward from r. Since the outdegree of r is B, it satisfies $\sum_{(r, v_i) \in E'} x_i = B$. Thus, based on E', a set X' such that $\sum_{x_i \in X'} x_i = B$ can be constructed as $X' = \{x_i \mid (r, v_i) \in E'\}$. □

Theorem 17(I) gives an inapproximability for $\mathcal{G}$:

Corollary 18. *Let $\rho(n) \geq 1$ be any polynomial-time computable function. For any linearly separable penalty function with a non-decreasing function, $MPDCO_{weighted}$ on $\mathcal{G}$ has no polynomial-time $\rho(n)$-approximation algorithm unless $P = NP$.*

Proof. The proof is similar to the one for Corollary 14. If there is a polynomial-time $\rho(n)$-approximation algorithm ALG for $\text{MPDCO}_{weighted}$ on $\mathcal{G}$, it can determine whether $OPT(G) = 0$ or $OPT(G) > 0$ in polynomial time since $OPT(G) \leq ALG(G) \leq \rho(n) \cdot OPT(G)$. Thus, by the reduction in Theorem 17, we can solve Partition using ALG in polynomial time, which contradicts the weak NP-hardness of Partition. □

Compared to the above inapproximability for $\mathcal{G}$, checking whether $OPT(G) = 0$ or $OPT(G) > 0$ for $\mathcal{H}$ can be done in linear time.

Theorem 19. *For any linearly separable penalty function, there is a linear-time algorithm which can check whether $OPT(G) = 0$ or $OPT(G) > 0$ for $G \in \mathcal{H}$*

Proof. Let the common degree constraint be (a, b), and the maximum weight of an edge be w_{max}. Since G is a star, it holds that $|E(G)| = |V(G)| - 1$. This implies that there is a vertex having outdegree 0 under any orientation. Hence, if $a > 0$, then we can know that $OPT(G) > 0$. Assume $a = 0$ in below.

Since every edge is oriented and its weight is contained as a part of outdegree of a vertex, we observe that the maximum outdegree of a vertex is at least w_{max} under any orientation. By this observation, $b \geq w_{max}$ holds if $OPT(G) = 0$. Consider a linear-time algorithm in which every edge is oriented toward the center in turn. The maximum outdegree of a vertex under this orientation is w_{max}. Namely, if $b \geq w_{max}$, the total penalty of this orientation is 0, i.e., $OPT(G) = 0$. Hence, $b \geq w_{max}$ holds if and only if $OPT(G) = 0$.

By the above discussion, we can check whether $OPT(G) = 0$ or $OPT(G) > 0$ by checking whether both of $a = 0$ and $b \geq w_{max}$ hold or not. Note that, in the above, we construct an orientation by an algorithm, however we do not need to construct any orientation if we just want to know whether $OPT(G) = 0$ or not; we just need to scan the edges in linear time and know the maximum weight of an edge. The theorem follows. □

Remark 3. Also for an edge-weighted tree G with $|R(G)| = 1$, checking whether $OPT(G) = 0$ or $OPT(G) > 0$ can be done in linear time by checking $a = 0$ and $b \geq w_{max}$. Consider a linear-time algorithm in which an arbitrary vertex is chosen as root and every edge is oriented toward the root. Under the obtained orientation, the outdegree of an vertex is at least 0 and at most w_{max}. Then, exactly the same argument as the above proof also shows that there is a vertex of outdegree 0 and another vertex of outdegree at least w_{max} under any orientation for edge-weighted trees.

By Theorem 19, we can easily know whether $OPT(G) = 0$ or not for a star in $\mathcal{H}$. Hence there may be an approximation algorithm for $\mathcal{H}$, despite the inapproximability for $\mathcal{G}$. Indeed we propose a polynomial-time approximation scheme (PTAS) for $\mathcal{H}$ in below.

For a star, the *inward orientation* represents the orientation in which every edge is oriented toward the center. We say that an edge is oriented *inward* if it is oriented toward the center, *outward* otherwise. Clearly the inward orientation is obtained in linear time. The next lemma relates the penalty of inward orientation with that of an optimal orientation.

Lemma 20. *For a star $G \in \mathcal{H}$, suppose that $k \geq 1$ edges are oriented outward in an optimal orientation. Then, for any linearly separable penalty function, $A(G)/OPT(G) \leq 1 + 1/k$, where $A(G)$ is the total penalty of the inward orientation.*

Proof. Let $R(G) = \{(a, b)\}$ and the penalty function be linearly separable with a function g. The center and leaves of G are denoted by r and $v_1, v_2, \ldots, v_{n-1}$, respectively. The weight of an edge $\{r, v_i\}$ is denoted by w_i, where $w_i \leq b$ holds since $G \in \mathcal{H}$. Let Λ^* and Λ be an optimal orientation and the inward orientation, respectively. Also $OPT(G)$ and $A(G)$ denote the total penalties of Λ^* and Λ, respectively.

Without loss of generality, in the optimal orientation, suppose that edges $\{r, v_1\}, \ldots, \{r, v_k\}$ are oriented outward for some $1 \leq k \leq n-1$, and the rest of the edges are oriented inward, where no edge is oriented inward when $k = n-1$. Let $w' = \sum_{i=1}^{k} w_i$. We partition the vertices $v_1, \ldots, v_k$ into two sets S and M depending on edge weights such that $S = \{v_i \mid w_i < a,\ 1 \leq i \leq k\}$ and $M = \{v_i \mid a \leq w_i \leq b,\ 1 \leq i \leq k\}$. Let P be the sum of the penalties imposed on $v_{k+1}, \ldots, v_{n-1}$, i.e., $P = \sum_{i=k+1}^{n-1} g(c_G(\Lambda^*, v_i))(= \sum_{i=k+1}^{n-1} g(c_G(\Lambda, v_i)))$ if $k \leq n-2$, and 0 if $k = n-1$.

One can see that

$$A(G) = P + g(a) + \sum_{v_i \in S} g(a - w_i),$$

where the second term $g(a)$ is the penalty on the center r, the third term represents the penalties on the vertices in S, and then no penalty is imposed on the vertices in M.

On the other hand, depending on w', there are three cases for $OPT(G)$:

- **(i)** $w' < a$: $OPT(G) = P + g(a - w') + kg(a)$, where $g(a - w')$ is the penalty on the center, and $kg(a)$ is the total penalty imposed on $v_1, \ldots, v_k$.
- **(ii)** $a \leq w' \leq b$: $OPT(G) = P + kg(a)$.
- **(iii)** $b < w'$: $OPT(G) = P + g(w' - b) + kg(a)$.

In all these three cases, it holds that

$$OPT(G) \geq P + kg(a). \tag{15}$$

It satisfies that $g(a - w_i) \leq g(a)$ for any $v_i \in S$ since g is nondecreasing (recall that any separated function g is assumed to be nondecreasing in our problem setting described in Section 1.1). Thus, it holds that

$$A(G) = P + g(a) + \sum_{v_i \in S} g(a - w_i) \leq P + (k+1)g(a), \tag{16}$$

where the inequality comes from the facts that $g(a - w_i) \leq g(a)$ for any $v_i \in S$ and $|S| \leq k$. Since $OPT(G) \geq P + kg(a)$, we observe that

$$\frac{A(G)}{OPT(G)} \leq \frac{P + (k+1)g(a)}{P + kg(a)} = 1 + \frac{g(a)}{P + kg(a)} \leq 1 + \frac{1}{k}. \quad \square$$

Based on the above lemma, we design the following algorithm, which takes $G \in \mathcal{H}$ and a nonnegative constant k as input and then outputs an orientation of G.

Step 1: Construct all orientations in which at most k edges are oriented outward (including the inward orientation). Let $\mathcal{L}$ be the set of obtained orientations.
Step 2: For each orientation $\Lambda \in \mathcal{L}$, compute its total penalty $p(c_G(\Lambda))$.
Step 3: Output an orientation with $\min_{\Lambda \in \mathcal{L}}\{p(c_G(\Lambda))\}$.

We show that the above algorithm is a PTAS for MPDCO$_{weighted}$ on $\mathcal{H}$ with a concave or step function. (Note that the above algorithm is also a PTAS for convex functions, however we will see that the inward orientation is optimal for MPDCO$_{weighted}$ on $\mathcal{H}$ with convex functions later.)

Theorem 21. *For any ϵ satisfying $0 < \epsilon \leq 1$, there is an $O(n^{1/\epsilon})$-time $(1+\epsilon)$-approximation algorithm for MPDCO$_{weighted}$ on $\mathcal{H}$ with a concave or step function.*

Proof. If at most k edges are oriented outward in an optimal orientation, then Step 1 finds it. Otherwise, the inward orientation has the penalty at most $(1 + 1/(k+1)) \cdot OPT(G)$ from Lemma 20, since the optimal orientation has at least $k+1$ outward edges. Namely the approximation ratio of the algorithm is $1 + 1/(k+1)$.

Let us estimate the running time of the algorithm. Step 1 needs $O(n^{k+1})$ time since there are $O(n^k)$ orientations satisfying the condition, i.e., $|\mathcal{L}| = O(n^k)$ and each orientation is constructed in $O(n)$ time. Step 2 spends $O(n^{k+1})$ time since $O(n)$ time is needed to compute the total penalty $p(c_G(\Lambda))$ of an orientation $\Lambda \in \mathcal{L}$. Step 3 spends $O(|\mathcal{L}|) = O(n^k)$ time to find the minimum among $O(|\mathcal{L}|)$ values of penalties. Thus the total running time is $O(n^{k+1})$.

Setting $\epsilon = 1/(k+1)$, the above algorithm is an $O(n^{1/\epsilon})$-time $(1+\epsilon)$-approximation algorithm, where $0 < \epsilon \leq 1$ since $k \geq 0$. $\square$

In Theorem 17(II), the weak NP-hardness of MPDCO$_{weighted}$ on $\mathcal{H}$ is shown for a concave function or a step function. Hence the above PTAS is a possibly best algorithm for MPDCO$_{weighted}$ on $\mathcal{H}$ with a concave function or step function. As for convex functions, a more careful estimation in the proof of Lemma 20 shows that the inward orientation is optimal for $\mathcal{H}$. Namely, we can solve the problem MPDCO$_{weighted}$ on $\mathcal{H}$ with a concave function in linear time.

Theorem 22. *The inward orientation is optimal for MPDCO$_{weighted}$ on $\mathcal{H}$ with a convex function, i.e., there is a linear-time algorithm to solve MPDCO$_{weighted}$ on $\mathcal{H}$ with a convex function.*

Proof. We consider what happens in the proof of Lemma 20 when the separated penalty function is restricted to be a convex function. Let g be the separated function of the penalty function. We show that the inward orientation is optimal for MPDCO$_{weighted}$ on $\mathcal{H}$ with a convex function.

If $|S| \leq k-1$, then (16) can be replaced with

$$A(G) = P + g(a) + \sum_{v_i \in S} g(a - w_i) \leq P + kg(a),$$

since $g(a - w_i) \leq g(a)$ for any $v_i \in S$. In this case $A(G) \leq P + kg(a) \leq OPT(G)$ since $P + kg(a) \leq OPT(G)$ from (15), i.e., $A(G) = OPT(G)$; the inward orientation is optimal.

Assume that $|S| = k$, i.e., every edge oriented outward in Λ^* has weight less than a. Let $\alpha_i = \max\{a - \sum_{j=1}^{i} w_j, 0\}$, where $\alpha_0 = a$. Note that $\alpha_k = a - w'$ if $w' < a$ and 0 otherwise. We observe that for any $i \geq 1$,

$$g(a) - g(a - w_i) \geq g(\alpha_{i-1}) - g(\alpha_i)$$

holds, since $a - (a - w_i) = \alpha_{i-1} - \alpha_i = w_i$, $a \geq \alpha_{i-1}$, and g is convex. Hence,

$$\sum_{v_i \in S} (g(a) - g(a - w_i)) \geq \sum_{v_i \in S} (g(\alpha_{i-1}) - g(\alpha_i)) = g(\alpha_0) - g(\alpha_k).$$

Since $|S| = k$ and hence $\sum_{v_i \in S}(g(a) - g(a - w_i)) = kg(a) - \sum_{v_i \in S} g(a - w_i)$, we have

$$kg(a) - \sum_{v_i \in S} g(a - w_i) \geq g(\alpha_0) - g(\alpha_k) = g(a) - g(\alpha_k). \tag{17}$$

If $w' < a$, then $g(\alpha_k) = g(a - w')$. Thus, based on (17) and (i) in the proof of Lemma 20, it holds that

$$OPT(G) = P + g(a - w') + kg(a) \geq P + g(a) + \sum_{v_i \in S} g(a - w_i) = A(G),$$

i.e., the inward orientation is optimal. If $w' \geq a$, then $g(\alpha_k) = g(0) = 0$. Again based on (17), it holds that

$$OPT(G) \geq P + kg(a) \geq P + g(a) + \sum_{v_i \in S} g(a - w_i) = A(G),$$

where the first inequality comes from (15). Hence, the inward orientation is optimal also for the case $w' > a$.

For all the cases, the inward orientation is optimal. This shows that there is a linear-time algorithm for MPDCO$_{weighted}$ on $\mathcal{H}$ with a convex function. □

5. Concluding remarks

In this paper, we studied a type of degree-constrained orientation of undirected graphs. The problem is formulated as an optimization problem that minimizes penalties for violation. We showed inapproximability results of the problem with concave or step functions. Then we designed polynomial-time algorithms for the problem with convex functions and for the case where the input is restricted to trees or graphs with bounded treewidth. Finally, we presented several tractability and intractability results for the edge-weighted variant of the problem.

Designing approximation algorithms for the problem on unweighted general graphs with concave or step functions is a further research topic. Also, it may be interesting to investigate other restricted graph classes, e.g., interval graphs. To further consider the hypergraph setting and the edge-weighted variant of the problem, as seen in Sections 2 and 4, respectively is another topic. Finally, there may be many other possible extensions of the problem to be considered.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partially supported by JST CREST JPMJR1402 and JSPS KAKENHI Grant Numbers JP17H01698, JP17K00016, JP17K00024, JP17K19960, JP20H05967, JP21K11755, and JP21K19765.

References

[1] H. Robbins, A theorem on graphs, with an application to a problem of traffic control, Am. Math. Mon. 46 (1939) 281–283.
[2] C. Nash-Williams, On orientations, connectivity and odd vertex pairings in finite graphs, Can. J. Math. 12 (1960) 555–567.
[3] T. Ito, Y. Miyamoto, H. Ono, H. Tamaki, R. Uehara, Route-enabling graph orientation problems, in: ISAAC, 2009, pp. 403–412.
[4] A. Schrijver, Combinatorial Optimization, Springer, 2003.
[5] L. Lau, R. Ravi, M. Singh, Iterative Methods in Combinatorial Optimization, Cambridge Texts in Applied Mathematics, Cambridge Univ. Press, 2011.
[6] A. Frank, Connections in Combinatorial Optimization, Oxford University Press, USA, 2011.
[7] H. Landau, On dominance relations and the structure of animal societies: III. The condition for a score structure, Bull. Math. Biol. 15 (1953) 143–148.
[8] S.L. Hakimi, On the degree of the vertices of a directed graph, J. Franklin Inst. 279 (1965) 290–308.
[9] A. Frank, A. Gyárfás, How to orient the edges of a graph?, Combinatorica I (1978) 353–364.
[10] M. Chrobak, D. Eppstein, Planar orientations with low out-degree and compaction of adjacency matrices, Theor. Comput. Sci. 86 (1991) 243–266.
[11] H.N. Gabow, Upper degree-constrained partial orientations, in: SODA, 2006, pp. 554–563.
[12] Y. Asahiro, E. Miyano, H. Ono, K. Zenmyo, Graph orientation algorithms to minimize the maximum outdegree, Int. J. Found. Comput. Sci. 18 (2007) 197–215.
[13] Y. Asahiro, J. Jansson, E. Miyano, H. Ono, K. Zenmyo, Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree, J. Comb. Optim. 22 (2011) 78–96.
[14] D.R. Page, R. Solis-Oba, A 3/2-approximation algorithm for the graph balancing problem with two weights, Algorithms 9 (2016) 38.
[15] T. Ebenlendr, M. Krčál, J. Sgall, Graph balancing: a special case of scheduling unrelated parallel machines, Algorithmica 68 (2014) 62–80.
[16] G.S. Brodal, R. Fagerberg, Dynamic representation of sparse graphs, in: 6th International Workshop on Algorithms and Data Structures (WADS 1999), 1999, pp. 342–351.
[17] J.K. Lenstra, D.B. Shmoys, É. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Math. Program. 46 (1990) 259–271.
[18] Y. Asahiro, J. Jansson, E. Miyano, H. Ono, Degree-constrained graph orientation: maximum satisfaction and minimum violation, Theory Comput. Syst. 58 (2016) 60–93.
[19] I. Dinur, S. Safra, On the hardness of approximating minimum vertex cover, Ann. Math. (2005) 439–485.
[20] S. Khot, D. Minzer, M. Safra, On independent sets, 2-to-2 games and Grassmann graphs, in: STOC, 2017, pp. 576–589.
[21] S. Khot, D. Minzer, M. Safra, Pseudorandom sets in Grassmann graph have near-perfect expansion, in: 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS), 2018, pp. 592–601.

[22] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows - Theory, Algorithms and Applications, Prentice Hall, 1993.
[23] R.K. Ahuja, D.S. Hochbaum, J.B. Orlin, Solving the convex cost integer dual network flow problem, Manag. Sci. 49 (2003) 950–964.
[24] H.N. Gabow, R.E. Tarjan, Faster scaling algorithms for network problems, SIAM J. Comput. 18 (1989) 1013–1036.
[25] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, R.E. Tarjan, Time bounds for selection, J. Comput. Syst. Sci. 7 (1973) 448–461.
[26] R. Halin, S-functions for graphs, J. Geom. 8 (1976) 171–186.
[27] N. Robertson, P.D. Seymour, Graph minors. III. Planar tree-width, J. Comb. Theory, Ser. B 36 (1984) 49–64.
[28] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.
[29] T. Kloks, Treewidth, Computations and Approximations, Lecture Notes in Computer Science, vol. 842, Springer-Verlag, Berlin Heidelberg, 1994.
[30] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. ven Rooij, J. Wojtaszczyk, Solving connectivity problems parameterized by treewidth in single exponential time, in: 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011), 2011, pp. 150–159.
[31] M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer International Publishing, 2015.
[32] M.R. Fellows, D. Hermelin, F.R.S. Vialette, On the parameterized complexity of multiple-interval graph problems, Theor. Comput. Sci. 410 (2009) 53–61.
[33] M. Dom, D. Lokshtanov, S. Saurabh, Y. Villanger, Capacitated domination and covering: a parameterized perspective, in: M. Grohe, R. Niedermeier (Eds.), Parameterized and Exact Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 78–90.
[34] E. Lawler, Combinatorial Optimization, Dover Publications, 1976.
[35] M.R. Garey, D.S. Johonson, Computers and Intractability, W.H. Freeman, 1979.