International Journal of Foundations of Computer Science
Vol. 22, No. 3 (2011) 583–601
© World Scientific Publishing Company
DOI: 10.1142/S0129054111008246

GRAPH ORIENTATION TO MAXIMIZE THE MINIMUM WEIGHTED OUTDEGREE

YUICHI ASAHIRO

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

JESPER JANSSON

Ochanomizu University
Bunkyo-ku, Tokyo 112-8610, Japan
Jesper.Jansson@ocha.ac.jp

EIJI MIYANO

Department of Systems Design and Informatics, Kyushu Institute of Technology
Iizuka, Fukuoka 820-8502, Japan
miyano@ces.kyutech.ac.jp

HIROTAKA ONO

Department of Informatics, Kyushu University
Nishi-ku, Fukuoka 819-0395, Japan
ono@inf.kyushu-u.ac.jp

Received 20 July 2009
Accepted 2 March 2010
Communicated by Jacir L. Bordir

We study a new variant of the graph orientation problem called MaxMinO where the input is an undirected, edge-weighted graph and the objective is to assign a direction to each edge so that the minimum weighted outdegree (taken over all vertices in the resulting directed graph) is maximized. All edge weights are assumed to be positive integers. This problem is closely related to the job scheduling on parallel machines, called the machine covering problem, where its goal is to assign jobs to parallel machines such that each machine is covered as much as possible. First, we prove that MaxMinO is strongly NP-hard and cannot be approximated within a ratio of $2-\epsilon$ for any constant $\epsilon > 0$ in polynomial time unless P=NP, even if all edge weights belong to $\{1, 2\}$, every vertex has degree at most three, and the input graph is bipartite or planar. Next, we show how to solve MaxMinO exactly in polynomial time for the special case in which all edge weights are equal to 1. This technique gives us a simple polynomial-time $\frac{w_{max}}{w_{min}}$-approximation algorithm for MaxMinO where w_{max} and w_{min} denote the maximum and minimum weights among all the input edges. Furthermore, we also observe that this approach yields an exact algorithm for the general case of MaxMinO whose running time is polynomial whenever the number of edges having weight larger than w_{min} is at most logarithmic in the number of vertices. Finally, we show that MaxMinO is solvable in polynomial time if the input is a cactus graph.

Keywords: Graph orientation; Approximation algorithm; Hardness of approximation; Scheduling; Maximum flow.

Mathematics Subject Classification 2010: 05C85, 68Q17, 68R10

1. Introduction

An orientation of an undirected graph is an assignment of a direction to each of its edges. Graph orientation is a well-studied area of graph theory and combinatorial optimization and thus a large variety of objective functions have been considered so far. The objective of the present paper is the maximization of the minimum outdegree. It is closely related to the classic job scheduling on parallel machines. In the parallel machine scheduling scenario, our problem can be regarded as the restricted assignment variant of the machine covering problem [23]: We are given a set of jobs J, a set of machines M to process the jobs and a processing time p_j for each job $j \in J$. For a given assignment of jobs to machines, the load of a machine $m_i \in M$ is defined as the sum of the processing of jobs assigned to m_i. The goal of the machine covering problem is to assign jobs to parallel machines such that each machine is occupied (covered) as long as possible (without introducing machine idle times). This problem was originally motivated by the sequencing of maintenance actions for modular gas turbine aircraft engines [10]. In the restricted assignment variant studied in the paper, the job can only be processed on a subset of the machines. In the following, we first define several terminologies and our objective function, then describe related work, and summarize our results.

Problem definition. Let $G = (V, E, w)$ be a given undirected, edge-weighted graph with vertex set V and edge set E whose weights are numbers specified by a function w. An *orientation* Λ of G is defined to be any function on E of the form $\Lambda : \{u, v\} \mapsto \{(u, v), (v, u)\}$, i.e., an assignment of a direction to each undirected edge $\{u, v\}$ in E. Given an orientation Λ of G, the *weighted outdegree* $d_\Lambda(v)$ of a vertex $v \in V$ is defined as the total weight of all edges leaving v, i.e., $d_\Lambda(v) = \sum_{\substack{\{u,v\} \in E: \\ \Lambda(\{u,v\})=(v,u)}} w(\{u, v\})$, and the *minimum weighted outdegree* $\delta_\Lambda(G)$ is defined by $\delta_\Lambda(G) = \min_{v \in V}\{d_\Lambda(v)\}$.

This paper deals with the problem of finding an orientation of the input graph such that the minimum weighted outdegree is maximized. We call this problem Maximum Minimum Weighted Outdegree Graph Orientation Problem (MaxMinO for short): The input is an undirected, edge-weighted graph $G = (V, E, w)$ with $w : E \to \mathbb{Z}^+$, where $\mathbb{Z}^+$ denotes the set of positive integers, and the objective is to find an orientation Λ^* of G which maximizes $\delta_\Lambda(G)$ over all possible orientations Λ of G. Such an orientation is called a *max-min orientation of* G, and the corresponding value $\delta_{\Lambda^*}(G)$ is denoted by $OPT(G)$. The special case of MaxMinO where all edge weights of the input graph are equal to 1 is referred to as *unweighted* MaxMinO.

Throughout the paper, we use the following notation: $n = |V|$, $m = |E|$, and $W = \sum_{e \in E} w(e)$ for the input G. Furthermore, w_{max} and w_{min} denote the maximum and minimum weights, respectively, among all edges in E. For any $v \in V$, the (unoriented) *weighted degree of* v, denoted by $d(v)$, is the sum of all weights of edges incident to v, and $\Delta = \max_{v \in V}\{d(v)\}$ is the maximum (unoriented) weighted degree among all vertices in G. Also for a (fixed) $v \in V$, we call $|\{\{u, v\} \in E\}|$ (i.e., the number of edges incident to v) the (unoriented) *unweighted degree of* v, and denote it by $deg(v)$. We also call $\max_{v \in V} deg(v)$ the (unoriented) *unweighted degree of* G, both of which will be used to focus on the topological structure of the graph.

We say that an algorithm $\mathcal{A}$ is a *σ-approximation algorithm* for MaxMinO or that $\mathcal{A}$'s *approximation ratio* is at most σ, if $OPT(G) \leq \sigma \cdot \mathcal{A}(G)$ holds for any input graph G, where $\mathcal{A}(G)$ is the minimum weighted outdegree in the orientation returned by $\mathcal{A}$ on input G.

Related work. MaxMinO studied in the current work is closely related to the restricted assignment variant of the machine covering problem, which is often called the Santa Claus problem [4, 5, 9, 14]: Santa Claus has m gifts (corresponding to jobs, and to edges in MaxMinO) that he wants to distribute among n children (corresponding to machines, and to vertices in MaxMinO). Some gift may be worth \$100 but another may be less expensive, and some children do not want some of the gifts whatsoever (i.e., its value is 0 for the children). The goal of Santa Claus is to distribute the gifts in such a way that the least lucky child is as happy as possible. In addition, MaxMinO has the following restriction: Every gift is of great value only to exactly two children and thus it must be delivered to one of them. For the Santa Claus problem, Golovin [14] provided an $O(\sqrt{n})$-approximation algorithm for the restricted case where the value of each gift belongs to $\{1, k\}$ for some integer k. Bansal and Sviridenko [4] considered the general value case and showed that a certain linear programming relaxation can be used to design an $O(\log\log m / \log\log\log m)$-approximation algorithm, while Bezakova and Dani [5] showed that the general case is already NP-hard to approximate within ratios smaller than 2.

Another objective function studied for the graph orientation problem is that of *minimizing the maximum weighted outdegree* (MinMaxO), also known as *Graph Balancing* [1, 2, 3, 8, 16, 21]: Given an undirected graph with edge weights, we are asked to assign a direction to each edge so that the maximum outdegree is minimized. It can be shown that MinMaxO is generally NP-hard. Furthermore, Asahiro et al. [2] showed that it is still weakly NP-hard for outerplanar graphs, and strongly NP-hard for P_4-bipartite graphs. Fortunately, however, they also showed [2] that MinMaxO is tractable if the input is limited to trees or even to cactus graphs. Note that the class of cactus graphs is a maximal subset of the class of outerplanar graphs and the class of P_4-bipartite graphs, and a minimal superset of the class of trees. Very recently, Ebenlendr et al. [8] designed a polynomial-time 1.75-approximation algorithm for the general weighted case, and Asahiro et al. [1] showed that MinMaxO can be approximated within an approximation ratio of 1.5 in polynomial-time if all edge weights belong to $\{1, 2\}$. As for inapproximability, it is known that MinMaxO is NP-hard to approximate within approximation ratios smaller than 1.5 even for this restricted $\{1, 2\}$-case [1, 8].

Our new results. In this paper we study the computational complexity and (in)approximability of the machine covering problem from the viewpoint of graph orientation. In Section 2, we prove that MaxMinO is strongly NP-hard and cannot be approximated within a ratio of $\min\{2, \frac{w_{max}}{w_{min}}\} - \epsilon$ for any constant $\epsilon > 0$ in polynomial time unless P=NP, even if all edge weights belong to $\{w_{min}, w_{max}\}$, every vertex has unweighted degree at most three, and the input graph is bipartite and planar. As mentioned above, although MaxMinO imposes a strong restriction on the Santa Claus problem, unfortunately it remains hard.

Section 3 first considers the unweighted MaxMinO problem. We obtain an optimal orientation algorithm which runs in $O(m^{3/2} \cdot \log m \cdot \log^2 \Delta)$ time for the special case in which all edge weights are equal to 1. Here, it is important to note that Golovin [14] already claimed that the unweighted case of MaxMinO (more precisely, the Santa Claus problem described in **Related work**) can be solved in polynomial time, but no proof of this claim has ever been published as far as we know. Our contribution here is to provide a non-trivial, yet simple and efficient algorithm along with explicit proofs of its correctness and a running time analysis. In the same section, we also observe that our approach yields an exact algorithm for the general case of MaxMinO whose running time is polynomial whenever the number of edges having weight larger than w_{min} is at most logarithmic in the number of vertices.

Next, Section 4 shows how to use our algorithm for unweighted MaxMinO to directly obtain an $\frac{w_{max}}{w_{min}}$-approximation algorithm running in the same time for the general (weighted) case of MaxMinO, i.e., it always outputs an orientation Λ' of G which satisfies $OPT(G) \leq \frac{w_{max}}{w_{min}} \cdot \delta_{\Lambda'}(G)$. This simple approximation algorithm is the best possible for the case where the weights of edges belong to $\{w_{min}, w_{max}\}$ with $w_{max} \leq 2w_{min}$ since the lower bound of approximation ratios is $\min\{2, \frac{w_{max}}{w_{min}}\}$ as described above.

In the field of combinatorial optimization, much work is often devoted to seek a subset of instances that is tractable and as large as possible. For example, if the input graph G is a tree, then $OPT(G)$ is always 0 because the number of vertices is larger than the number of edges, and in any orientation of G, at least one vertex must have no outgoing edges. Also, for the case of cycles, MaxMinO is quite trivial since the clockwise or counterclockwise orientation along the cycle gives us the optimal value of w_{min}. On the other hand, the class of planar graphs is too large to allow a polynomial-time optimal algorithm (under the assumption of P$\neq$NP) as shown in Section 2. Hence, our goal in Section 5 is to find a (pseudo-)polynomially solvable subset between trees and planar graphs. We first show in Section 5.1 that MaxMinO remains in P even if we make the set of instances so large that it contains the class of cactus graphs, and then show in Section 5.2 that the problem becomes solvable in pseudo-polynomial time for series-parallel graphs. Finally, we conclude the paper with some remarks in Section 6.

2. Hardness Results

In this section, we prove the strong NP-hardness and inapproximability of the MaxMinO problem for bipartite graphs and planar graphs.

Theorem 1. *Even if the edge weights are in $\{w_{min}, w_{max}\}$ and the input graph is bipartite and planar in which the unweighted degree is bounded by three, (i)* MaxMinO *is strongly NP-hard, and (ii)* MaxMinO *has no pseudo-polynomial time algorithm whose approximation ratio is smaller than* $\min\{2, \frac{w_{max}}{w_{min}}\}$*, unless P = NP.*

Proof. For the purpose of making our basic ideas clear, we assume for a while that the input graph is general, i.e., it is not limited to bipartite or planar. Then, we first show that the MaxMinO problem is strongly NP-hard even if all the edge weights belong to $\{w_{min}, w_{max}\}$ for any positive integers $w_{min} < w_{max}$. The proof is by a reduction from At-Most-3-SAT(2L).

At-Most-3-SAT(2L) is a restriction of 3-SAT where each clause contains at most three literals and each literal (not variable) appears at most twice in a formula. It can be easily proved that At-Most-3-SAT(2L) is NP-hard by using problem [LO1] on p. 259 of [11].

First, we pick any fixed positive integers for w_{min} and w_{max} such that $w_{min} < w_{max}$. Given a formula ϕ of At-Most-3-SAT(2L) with n variables $\{v_1, \ldots, v_n\}$ and m clauses $\{c_1, \ldots, c_m\}$, we then construct a graph G_ϕ including gadgets that mimic (a) variables and (b) clauses. To define these, we prepare a gadget consisting of a cycle of 3 vertices and 3 edges (i.e., a triangle) where each edge of the cycle has weight w_{max}. We call this a *triangle gadget*. Apart from these triangle gadgets, we define gadgets for (a) variables and (b) clauses: (a) Each variable gadget corresponding to a variable v_i consists of two vertices labeled by v_i and $\overline{v_i}$ and one edge $\{v_i, \overline{v_i}\}$ between them. The weight of $\{v_i, \overline{v_i}\}$ is w_{max}. By the definition of At-Most-3-SAT(2L), some literals (say v_i for example) do not occur (or may occur only once). In such a case, we attach a triangle gadget to the variable gadget by adding two edges (one edge) of weight w_{min} that connects vertex v_i and two different vertices (one vertex) of the triangle gadget. (b) Each clause gadget consists of one representative vertex labeled by c_j, corresponding to clause c_j of ϕ, and a triangle gadget connected to this c_j-vertex by an edge of weight w_{min}. The representative vertex c_j is also connected to at most three vertices in the variable gadgets that have the same labels as the literals in the clause c_j, by edges of weight w_{min}. For example, if $c_1 = x \vee \overline{y}$ appears in ϕ, then vertex c_1 is connected to vertices x and $\overline{y}$. (See Figure 1.) We have the following lemma.

Lemma 2. *For the reduced graph G_ϕ, the following holds:*

(i) $OPT(G_\phi) \geq \min\{2w_{min}, w_{max}\}$ if ϕ is satisfiable.
(ii) $OPT(G_\phi) \leq w_{min}$ if ϕ is not satisfiable.

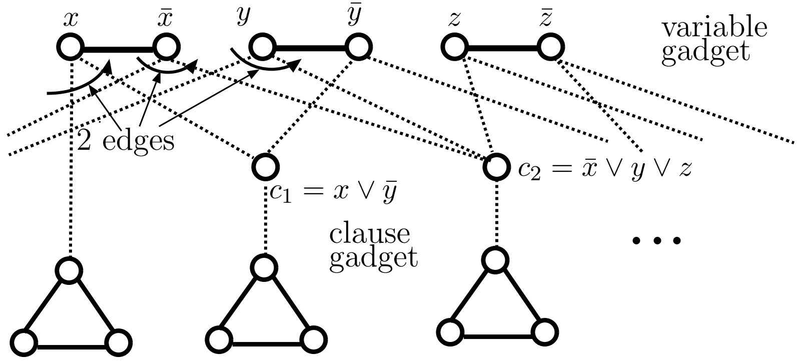

Fig. 1. Reduction from At-Most-3-SAT(2L) (Solid and dotted edges have weight w_{max} and w_{min}, respectively.)

Proof. First we prove (i). We fix the orientation of edges in the triangle gadget. Each triangle gadget is oriented in such a way that the triangle forms a directed cycle, which guarantees that the minimum weighted outdegree among those vertices belonging to the triangle is at least w_{max}. Then, edges that connect triangle gadgets to other vertices are oriented towards the triangle gadgets.

Now suppose that there is a satisfying truth assignment τ for the formula ϕ. From τ, we construct an orientation with $OPT(G_\phi) \geq \min\{2w_{min}, w_{max}\}$. If $v_i = true$ in τ, the edge $\{v_i, \overline{v_i}\}$ is oriented from v_i to $\overline{v_i}$; otherwise, from $\overline{v_i}$ to v_i. At this moment, the weighted outdegree of vertices associated with the literals of *true* and *false* assignments is w_{max} and 0, respectively. (We call the vertices associated with literals of *true* (resp., *false*) assignments *true* (resp., *false*) *vertices*. For example, if a variable $x = false$ in a truth assignment, then the upper leftmost vertex x is called a false vertex and the second leftmost vertex $\overline{x}$ is called a true vertex in Figure 1.) Each false vertex has at most two edges connected to clause vertices, and in case a false vertex is connected to zero, one or two clause vertices, then it is connected respectively to two, one or zero triangle gadgets. We then orient such edges towards the clause vertices and triangle gadgets, which make the weighted outdegree of each false vertex $2w_{min}$. Thus the weighted outdegree of each vertex in a variable gadget is at least $\min\{2w_{min}, w_{max}\}$. Each clause vertex has at least one edge connected to a true vertex due to the truth assignment τ. We orient this edge towards the true vertex, which makes the weighted outdegree of the clause vertex at least $2w_{min}$, because it has an oriented edge (arc) towards to a triangle gadget (see the first paragraph of this proof). Hence, the weighted outdegree of every vertex is at least $\min\{2w_{min}, w_{max}\}$, which shows (i).

Next, we prove (ii) by showing that if the graph G_ϕ has an orientation whose minimum weighted outdegree is larger than w_{min}, i.e., at least $\min\{2w_{min}, w_{max}\}$, then ϕ is satisfiable by constructing the satisfying truth assignment. First of all, by the assumption that $\delta_\Lambda(G) \geq \min\{2w_{min}, w_{max}\} > w_{min}$ for an orientation Λ, we can assume that each triangle gadget is oriented clockwise (or counterclockwise); otherwise, it has a vertex whose weighted outdegree is at most w_{min}, which is a contradiction. Furthermore, without loss of generality, we can assume that edges that connect triangle gadgets to other vertices are oriented towards the triangle gadgets in any orientation.

If an edge in the ith variable gadget v_i is oriented from v_i to $\overline{v_i}$, then we assign $v_i = \mathit{true}$; otherwise, $v_i = \mathit{false}$. Then two edges between a vertex assigned with *false* and its two adjacent vertices (in clause and/or triangle gadgets) must be oriented towards these vertices. (Otherwise, the weighted outdegree of the false vertex is at most w_{min}, which contradicts the assumption.) Every clause vertex is connected with the variable and triangle gadgets. As mentioned above, every edge between a clause vertex and its triangle gadget can be assumed to be oriented towards the triangle gadget. It follows that for each clause vertex, there must be at least one edge directed from the clause vertex towards a vertex v in a variable gadget, and v must be a true vertex. This means that the above truth assignment satisfies all the clauses in ϕ. (End of the proof of Lemma 2) □

Lemma 2 means that MaxMinO is strongly NP-hard, and furthermore, unless P = NP, MaxMinO has no pseudo-polynomial time algorithm whose approximation ratio is smaller than $\min\{2, \frac{w_{max}}{w_{min}}\}$ even if the edge weights are in $\{w_{min}, w_{max}\}$ ($w_{min} < w_{max}$). In the following we strengthen those results; we prove the NP-hardness and the inapproximability of MaxMinO for planar bipartite graphs with the unweighted degree of at most three, by almost the same reduction as the above from Monotone-Planar-One-In-Three-3-SAT(2L), which is a variant of At-Most-3-SAT(2L), having both the planarity [17] and the monotonicity [12]. The *planarity* means that the graph constructed from an instance CNF, in which two vertices corresponding to a variable and a clause are connected by an edge if the variable occurs (positively or negatively) in the clause, is planar. The *monotonicity* means that in an input CNF formula each clause contains either only positive literals or only negative literals. In a graph constructed from a monotone CNF, the distance from a positive (resp., negative) vertex to a vertex of a clause consisting of only positive (resp., negative) literals is always odd, and then the distance from a positive (resp., negative) vertex to a vertex of a clause consisting of only negative (resp., positive) literals is always even. This implies that the constructed graph is bipartite.

One-In-Three-3-SAT itself is a variant of 3-SAT problem which asks whether there exists a truth assignment to the variables so that each clause has exactly one true literal and thus exactly two false literals (equivalently, each clause has exactly one false literal and exactly two true literals, and we use this variant for the reduction.) [19]. The reason why we use One-In-Three-3-SAT instead of At-Most-3-SAT is to bound the unweighted degrees of the constructed graphs. The above reduction from At-Most-3-SAT(2L) guarantees that the unweighted degrees of constructed graphs are bounded only by four, due to the clause gadgets

(As for the other gadgets, the unweighted degrees are at most three). In the new reduction from One-In-Three-3SAT(2L), we do not attach triangle gadgets to clause vertices, which makes the unweighted degrees of clause vertices three, and One-In-Three satisfiability guarantees that each clause vertex has two outgoing edges in an optimal MaxMinO solution.

Planar-One-In-Three-3-SAT is shown to be NP-complete in [18]. By applying an operation used in [2], we can transform an instance of Planar-One-In-Three-3-SAT into one of Monotone-Planar-One-In-Three-3-SAT. Moreover, by applying another operation used in the same paper [2], we can transform an instance of Monotone-Planar-One-In-Three-3-SAT into Monotone-Planar-One-In-Three-3-SAT(2L). This implies that the constructed graph is planar and bipartite and its unweighted degree is at most three. (To preserve the bipartiteness, we need to use bipartite gadgets, e.g., square gadgets, instead of triangle gadgets. Note that the new bipartite gadgets are connected only to variable gadgets whose corresponding literals do not appear or appear only once in the instance CNF.) (End of the proof of Theorem 1) □

This result is tight in a sense, because if the unweighted degree of the input graph is bounded by two (i.e., cycles or paths), obviously MaxMinO can be solved in linear time.

3. An Exact Algorithm for Unweighted Cases

MaxMinO is closely related to the problem of computing a maximum flow in a flow network with positive edge capacities. Indeed, maximum-flow-based techniques have been used in [3] to solve the analogous problem of computing an edge orientation which *minimizes* the *maximum* outdegree of a given unweighted graph (MinMaxO) in polynomial time. In this section, we extend the results of [3] by showing how a maximum flow algorithm can be used to efficiently solve unweighted MaxMinO.

For any input graph $G = (V, E)$ to unweighted MaxMinO, let $\mathcal{N}_G = (V_G, E_G)$ be the directed graph with vertex set V_G and edge set E_G defined by:

$$V_G = E \cup V \cup \{s, t\},$$
$$E_G = \big\{(s, e) \mid e \in E\big\} \cup \big\{(v, t) \mid v \in V\big\} \cup \big\{(e, v_i), (e, v_j) \mid e = \{v_i, v_j\} \in E\big\},$$

and for any integer $q \in \{0, 1, \ldots, \Delta\}$, let $\mathcal{N}_G(q) = (V_G, E_G, cap_q)$ be the flow network obtained by augmenting $\mathcal{N}_G$ with edge capacities cap_q, where:

$$cap_q(a) = \begin{cases} 1, & \text{if } a = (s, e) \text{ with } e \in E; \\ 1, & \text{if } a = (e, v) \text{ with } e \in E,\ v \in V; \\ q, & \text{if } a = (v, t) \text{ with } v \in V. \end{cases}$$

See Fig. 2 and Fig. 3 for an example of a graph G and a corresponding network $\mathcal{N}_G$, respectively.

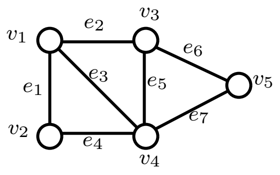

Fig. 2. An example of a graph G.

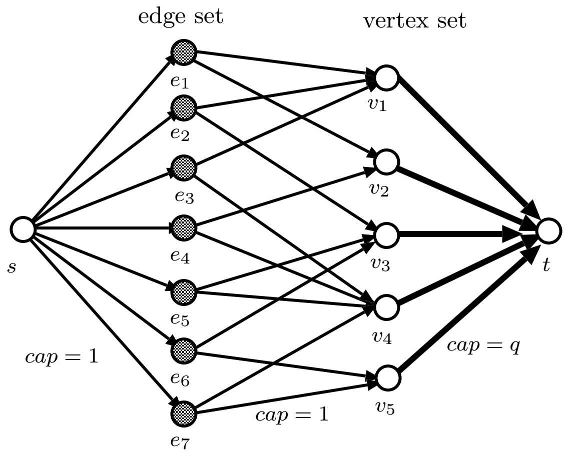

Fig. 3. A flow network $\mathcal{N}_G$ constructed from the graph G in Fig. 2.

Let $F(q)$ be an integral maximum directed flow[a] from vertex s to vertex t in $\mathcal{N}_G(q)$. Then, for each $e = \{v_i, v_j\} \in E$, either zero or one unit of flow in $F(q)$ passes through the corresponding vertex e in V_G, and thus at most one of the two edges (e, v_i) and (e, v_j) is assigned one unit of flow. This induces an orientation $\Lambda_{F(q)}$ of G based on $F(q)$ as follows: If the flow in $F(q)$ from vertex e to vertex v_i equals 1 then set $\Lambda_{F(q)}(e) := (v_i, v_j)$; else if the flow in $F(q)$ from e to v_j equals 1 then set $\Lambda_{F(q)}(e) := (v_j, v_i)$; else set $\Lambda_{F(q)}(e)$ arbitrarily.

Let $f(q)$ denote the value of a maximum directed flow from vertex s to vertex t in $\mathcal{N}_G(q)$. Then:

Lemma 3. *For any* $q \in \{0, 1, \ldots, \Delta\}$, $f(q) \leq q \cdot n$.

Proof. The sum of all edge capacities of edges leading into t in $\mathcal{N}_G(q)$ is $q \cdot n$. Clearly, the value of the maximum flow in $\mathcal{N}_G(q)$ cannot be larger than this sum. □

[a]Since all edge capacities are integers, we may assume by the integrality theorem (see, e.g., [7]) that the flow along each edge in $F(q)$ found by the algorithm in [13] is an integer.

Lemma 4. *For any $q \in \{0, 1, \ldots, \Delta\}$, $f(q) = q \cdot n$ if and only if $OPT(G) \geq q$.*

Proof.
$\Longrightarrow$) Suppose that $f(q) = q \cdot n$ and consider the maximum flow $F(q)$ defined above. For each $v \in V$, exactly q units of flow leave the corresponding vertex v in V_G because the edge capacity of (v, t) is q and there are n such vertices. This implies that q units of flow enter v, which is only possible if there are q edges of the form (e, v) in E_G that have been assigned one unit of flow each. Therefore, the induced orientation $\Lambda_{F(q)}$ ensures that $d_{\Lambda_{F(q)}}(v) \geq q$ for every $v \in V$, which yields $OPT(G) \geq q$.

$\Longleftarrow$) Suppose that $OPT(G) \geq q$ and let Λ be a max-min orientation of G. Let F' be the following directed flow from s to t in $\mathcal{N}_G(\Delta)$:

$$F'(a) = \begin{cases} 1, & \text{if } a = (s, e) \text{ with } e \in E; \\ 1, & \text{if } a = (e, v_i) \text{ with } e = \{v_i, v_j\} \in E \text{ and } \Lambda(e) = (v_i, v_j); \\ 0, & \text{if } a = (e, v_i) \text{ with } e = \{v_i, v_j\} \in E \text{ and } \Lambda(e) = (v_j, v_i); \\ d_\Lambda(v), & \text{if } a = (v, t) \text{ with } v \in V. \end{cases}$$

For every $v \in V$, the flow in F' along the edge (v, t) in $\mathcal{N}_G(\Delta)$ is $d_\Lambda(v) \geq OPT(G) \geq q$. By reducing each such edge flow to q, one obtains a directed flow which obeys the (stricter) edge capacity constraints of the flow network $\mathcal{N}_G(q)$ and has flow value $n \cdot q$. Thus, there exists a maximum directed flow from s to t in $\mathcal{N}_G(q)$ with value $q \cdot n$, so $f(q) \geq q \cdot n$. It follows from Lemma 3 that $f(q) = q \cdot n$. □

Lemmas 3 and 4 suggest the algorithm for unweighted MaxMinO named Algorithm Exact-1-MaxMinO.

Algorithm 1 Algorithm Exact-1-MaxMinO

```
1: Construct $\mathcal{N}_G$.
2: Use binary search on $q$ in the interval $\{0, 1, \ldots, \Delta\}$ to find the integer $q$ such
   that $f(q) = q \cdot n$ and $f(q+1) < (q+1) \cdot n$.
3: Compute $F(q)$ as a maximum directed flow from $s$ to $t$ in $\mathcal{N}_G(q)$.
4: Return $\Lambda_{F(q)}$.
```

Theorem 5. *Exact-1-MaxMinO solves unweighted* MaxMinO *in $O(m^{3/2} \cdot \log m \cdot \log^2 \Delta)$ time.*

Proof. The correctness of Exact-1-MaxMinO is guaranteed by Lemmas 3 and 4. Recall that the algorithm of Goldberg and Rao [13] computes a maximum flow in a flow network with N vertices, M edges, and maximum edge capacity C in $O(M \cdot \min\{N^{2/3}, M^{1/2}\} \cdot \log(N^2/M) \cdot \log C)$ time. Thus, for any $q \in \{0, 1, \ldots, \Delta\}$, to compute a maximum flow in the flow network $\mathcal{N}_G(q)$ takes $O(m^{3/2} \cdot \log m \cdot \log \Delta)$

time because $\mathcal{N}_G(q)$ contains $m+n+2 = O(m)$ vertices and $3m+n = O(m)$ edges and the capacity of each edge in $\mathcal{N}_G(q)$ is upper-bounded by Δ. Algorithm Exact-1-MaxMinO can therefore be implemented to run in $O(m^{3/2} \cdot \log m \cdot \log^2 \Delta)$ time. □

Finally, we outline how Exact-1-MaxMinO can be applied to solve weighted MaxMinO exactly. Let X be the set of all edges in E with weight larger than w_{min}. First fix an orientation Λ of edges in X, and modify the flow network $\mathcal{N}_G(q)$ based on Λ:

$$cap_q(a) = \begin{cases} 1, & \text{if } a = (s,e) \text{ for } e \in E \setminus X; \\ 0, & \text{if } a = (s,e) \text{ for } e \in X; \\ 1, & \text{if } a = (e,v) \text{ for } e \in E \setminus X,\ v \in V; \\ 0, & \text{if } a = (e,v) \text{ for } e \in X,\ v \in V; \\ \max\left\{ \left\lceil \dfrac{q - \sum_{\substack{e=(v,u)\in X,\\ \Lambda(e)=(v,u)}} w(e)}{w_{min}} \right\rceil, 0 \right\}, & \text{if } a = (v,t) \text{ for } v \in V. \end{cases}$$

Note that the definition of $cap_q(a)$ for $a = (v,t)$ comes from how many edges of weight w_{min} should be oriented from v under Λ to achieve weighted outdegree at least q. Then, run Exact-1-MaxMinO for each Λ while testing whether the flow value is $\sum_{v \in V} cap_q((v,t))$. If the answer is yes, there exists an orientation whose minimum weighted outdegree is at least q. Thus by binary search on q in the interval $\{0, 1, \ldots, \lceil W/n \rceil\}$, we can select the best resulting orientation.

The asymptotic running time becomes the same as that of Exact-1-MaxMinO multiplied by $2^{|X|}$, which is the number of candidates of Λ, and with an increase due to the larger interval for the binary search on q and the edge capacities being upper-bounded by $\max\{w_{max}, W/n\}$ instead of Δ.

Theorem 6. *Weighted* MaxMinO *can be solved in* $O(m^{3/2} \cdot \log m \cdot \log(w_{max} + W/n) \cdot \log(W/n) \cdot 2^{|X|})$ *time, where* $X = \{e \in E \mid w(e) > w_{min}\}$. □

Corollary 7. *If* $|X| = O(\log n)$ *then weighted* MaxMinO *can be solved in polynomial time.* □

4. A Simple Approximation Algorithm for General Cases

Here, we prove that ignoring the edge weights of the input graph and applying Exact-1-MaxMinO on the resulting unweighted graph immediately yields a $\frac{w_{max}}{w_{min}}$-approximation algorithm for the general case of the problem. The algorithm is named Approximate-MaxMinO and is listed in Algorithm 2.

Theorem 8. *Approximate-MaxMinO runs in* $O(m^{3/2} \cdot \log m \cdot \log^2 \Delta)$ *time and is a* $\frac{w_{max}}{w_{min}}$*-approximation algorithm for* MaxMinO.

Proof. The asymptotic running time of Algorithm Approximate-MaxMinO is the same as that of Exact-1-MaxMinO.

Algorithm 2 Algorithm Approximate-MaxMinO

```
1: Let $G'$ be the undirected graph obtained from $G$ by replacing the weight of every edge by 1.
2: Apply Algorithm Exact-1-MaxMinO on $G'$ and let $\Lambda'$ be the obtained orientation.
3: Return $\Lambda'$.
```

To analyze the approximation ratio, observe that $\delta_\Lambda(G) \geq w_{min} \cdot \delta_\Lambda(G')$ for any orientation Λ of G because the weight of any edge in G is at least w_{min} times larger than its weight in G'. Similarly, $w_{max} \cdot \delta_\Lambda(G') \geq \delta_\Lambda(G)$ for any orientation Λ of G. Now, let Λ' be the optimal orientation for G' returned by Approximate-MaxMinO and let Λ^* be an optimal orientation for G. Note that $\delta_{\Lambda'}(G') \geq \delta_{\Lambda^*}(G')$. Thus, $\delta_{\Lambda'}(G) \geq w_{min} \cdot \delta_{\Lambda'}(G') \geq w_{min} \cdot \delta_{\Lambda^*}(G') \geq \frac{w_{min}}{w_{max}} \cdot \delta_{\Lambda^*}(G) = \frac{w_{min}}{w_{max}} \cdot OPT(G)$. □

5. Exact Algorithms for Special Graphs

5.1. *An exact algorithm for cactus graphs*

In this section, we present a polynomial time algorithm which obtains optimal orientations for cactus graphs. A graph is a *cactus* if every edge is part of at most one cycle. To this end, we introduce vertex weight $\alpha_G(v)$ for each vertex v in a graph G which is considered as 0 in the input graph (we omit the subscript G of $\alpha_G(v)$ if it is apparent). Here we define the notion of weighted outdegree for a vertex in a vertex and edge weighted graph. The *weighted outdegree* $d_\Lambda(v)$ of a vertex v is defined as the weight of v itself plus the total weight of outgoing arcs of v, i.e.,

$$d_\Lambda(v) = \alpha(v) + \sum_{\substack{\{u,v\} \in E: \\ \Lambda(\{u,v\}) = (v,u)}} w(\{u,v\}).$$

In a cactus graph, a vertex in a cycle is a *gate* if it is adjacent to any vertex that does not belong to the cycle. Note that the unweighted degree of a gate is at least three. As for the number of gates in a cycle, the following is known:

Proposition 9 (Proposition 2 in [2]) *In a cactus graph G in which $deg(v) \geq 2$ for every vertex v, there always exists a cycle with at most one gate.* □

The main part of the proposed algorithm Exact-Cactus-MaxMinO is shown in Algorithm 3, which solves the decision version of the problem MaxMinO: Given a number K, this problem asks whether there exists an orientation whose value is at least K. We can develop an algorithm for the original problem MaxMinO by using this algorithm $O(\log \Delta)$ times in a binary search manner on optimal value, which is upper-bounded by Δ.

The correctness of Exact-Cactus-MaxMinO is based on the following property on optimal orientations for two graphs.

Algorithm 3 Algorithm Exact-Cactus-MaxMinO

```
1: **repeat**
2:   For a vertex $v$,
3:   **if** $\alpha(v) + d(v) < K$ **then**
4:     output No and halt.
5:   **else if** $\alpha(v) \geq K$ **then**
6:     $\Lambda(e) := (u, v)$ for every edge $e = \{u, v\}$ incident to $v$. Increase $\alpha(u)$ by
       $w(\{u, v\})$ for all edges $\{u, v\}$'s, and then remove $v$ and all such edges.
7:   **else if** $deg(v) = 1$ **then**
8:     (let its connecting edge be $e = \{v, u\}$)
9:     $\Lambda(e) := (v, u)$ and then remove $v$ and $e$.
10:  **else if** $deg(v) = 2$ **then**
11:    (let $e_1 = \{p, v\}$ and $e_2 = \{v, q\}$)
12:    **if** $\alpha(v) + w(e_1) < K$ and $\alpha(v) + w(e_2) < K$ **then**
13:      $\Lambda(e_1) := (v, p)$ and $\Lambda(e_2) := (v, q)$. Remove $v$, $e_1$, and $e_2$.
14:    **else if** $\alpha(v) + w(e_1) < K$ and $\alpha(v) + w(e_2) \geq K$ **then**
15:      $\Lambda(e_1) := (p, v)$ and $\Lambda(e_2) := (v, q)$ and also increase $\alpha(p)$ by $w(e_1)$.
         Remove $v$, $e_1$, and $e_2$.
16:    **end if**
17:  **end if**
18: **until** there does not exist a vertex $v$ satisfying either one of the above conditions

19: **for** a cycle $C := \langle v_0, v_1, \cdots, v_\ell = v_0 \rangle$ that has at most one gate **do**
20:   **if** $C$ does not have a gate **then**
21:     $\Lambda(\{v_i, v_{i+1}\}) := (v_i, v_{i+1})$ for $0 \leq i \leq \ell - 1$. Remove $C$.
22:   **else**
23:     (Let $v_0$ be the gate.)
24:     **if** $w(\{v_0, v_1\}) > w(\{v_0, v_{\ell-1}\})$ **then**
25:       $\Lambda(\{v_i, v_{i+1}\}) := (v_i, v_{i+1})$ for $0 \leq i \leq \ell - 1$ and increase $\alpha(v_0)$ by
          $w(\{v_0, v_1\})$
26:     **else**
27:       $\Lambda(\{v_i, v_{i+1}\}) := (v_{i+1}, v_i)$ for $0 \leq i \leq \ell - 1$ and increase $\alpha(v_0)$ by
          $w(\{v_0, v_{\ell-1}\})$.
28:     **end if**
29:     Remove $C$ except the gate $v_0$.
30:   **end if**
31: **end for**
32: **if** the graph is empty **then**
33:   output $\Lambda$ and halt.
34: **else**
35:   go back to line 1.
36: **end if**
```

Proposition 10. *Consider two graphs G and G' that differ only on their vertex weights. If $\alpha_G(v) \leq \alpha_{G'}(v)$ for every vertex v, then $OPT(G) \leq OPT(G')$ holds.* □

Theorem 11. *Given a cactus graph G and a target K, Exact-Cactus-MaxMinO outputs an orientation Λ such that $\delta_\Lambda(G) \geq K$ if such an orientation exists, in polynomial time.*

Proof. First we estimate the running time. Note that the number of edges of a cactus graph is $O(n)$ due to the planarity. Since each of executions of **repeat** loop or **for** loop determines the direction of at least one edge, the total number of times those steps are being processed is bounded by $O(n)$. Also all of those steps can be done in $O(n)$ time because they only find a vertex or a cycle with testing certain conditions by a constant time. Hence the total running time is $O(n^2)$.

The running time can be reduced to $O(n)$ by a careful implementation. Since now the input is a cactus graph, as preprocessing, (unweighted and weighted) degrees of the vertices are obtained and all cycles with a number of its gates are enumerated by, e.g., the depth-first-search in $O(n)$ time. Note here that we do not need to sort them, but maintain a list of vertices having unweighted degree one or two, and also a list of cycles having at most one gate. The **repeat** procedure of lines 1-18 just updates these information, i.e., unweighted/weighted degrees and vertex weights for the processed vertex v and its adjacent vertices. The total number of these updates is bounded above by $O(n)$ since they are updated only when the orientation of an edge incident to v is determined. As for the **for** procedure of lines 19-31, the discussion is similar and its running time is also bounded by $O(n)$. Therefore the total running time of this algorithm is $O(n)$.

Next we show the correctness of the algorithm. Each execution that removes some vertices and edges from the current graph H (lines 6, 9, 13, 15, 21 and 29) may increase the weight of some remaining vertices, and then obtains a modified graph H'. What we would like to show is that if $OPT(H) \geq K$, then (i) also $OPT(H') \geq K$, (ii) the determined directions of edges are correct, and so (iii) all the vertices removed at the step have weighted outdegree at least K.

Lines 3-4: If the condition is satisfied, the answer is clearly No.

Assume that $OPT(H) \geq K$.

Lines 5-9: It holds that $\alpha(v) + w(e) \geq K$ since it passes through the check at line 3. Let us consider the case that $\alpha(v) \geq K$. The weighted outdegree of the removed vertex v is at least $\alpha(v) \geq K$ in this case whichever the directions assigned to edges incident to v are. For simplicity, assume $deg(v) = 1$, i.e., only one edge $e = \{u, v\}$ is incident to v for a while. There are two possibilities: We assign either $\Lambda(e) := (u, v)$ or $\Lambda(e) := (v, u)$. Let the graph obtained by the former assignment with increasing $\alpha(u)$ by $w(e)$ be H', and let the graph obtained by the latter be H''. From Proposition 10, we observe that $OPT(H') \geq OPT(H'')$. If $OPT(H'') \geq OPT(H)$, then it holds that $OPT(H') \geq OPT(H'') \geq OPT(H)$ for both of the directions of the edge e. Conversely, the inequality $OPT(H'') < OPT(H)$ means that the weighted outdegree of the vertex u must be augmented by orienting the edge $e = \{v, u\}$ as (u, v) in the optimal orientation for H, because, otherwise it contradicts that $OPT(H)$ is the optimal value. Thus the assignment $\Lambda(e) := (u, v)$ is correct and it holds that $OPT(H') \geq OPT(H) \geq K$. The argument for the case $deg(v) \geq 2$ is similar and it also holds that $OPT(H') \geq OPT(H)$.

In the other case that $\alpha(v) < K$, if we assign $\Lambda(e) := (u, v)$, then the weighted outdegree of v is less than K, which contradicts the assumption $OPT(H) \geq K$. Hence the assignment $\Lambda(e) := (v, u)$ is correct and also the removed vertex v has weighted outdegree at least K. Also it holds that $OPT(H') \geq OPT(H)$, otherwise, it contradicts that $OPT(H)$ is the optimal value.

Lines 10-17: If we do not follow the rule at lines 12-13, the weighted outdegree of the processed vertex would be less than $K \leq OPT(H)$, which implies the operation is correct. As for the rule at lines 14-15, to guarantee the weighted outdegree of v is at least K, $\Lambda(e_2)$ should be (v, q). Then $\alpha(v) \geq K$ already holds, and we can fix $\Lambda(e_1) = (p, v)$, as well as the rule at line 6.

Lines 19-31: First of all, at the beginning of this part, it holds that $deg(v) \geq 2$ for every vertex v since it passes lines 7-9. Also for any vertex having degree two, if we orient one incident edge outward, its outdegree in the resulted orientation becomes at least K, because such vertices pass lines 10-17. From Proposition 9, we can always find a cycle having at most one gate.

Lines 20-21: It is obvious that the vertices removed at this step have weighted outdegree at least K because they passed lines 10-17. In addition to that it holds that $OPT(H') \geq OPT(H)$ since C is a connected component in H and $OPT(H) = \min\{OPT(C), OPT(H')\}$.

Lines 22-31: The vertices removed in this step all have weighted outdegree at least K because of the conditions of lines 10-17. Also the proposed assignment of directions for the cycle C increases $\alpha(v_0)$ as much as possible without breaking optimality, and it derives $OPT(H') \geq K$ by a similar argument as the one used in lines 5-9.

By the above discussion, if all the vertices are removed without answering No, the weighted outdegree of every vertex by the orientation Λ is at least K. $\square$

From Theorem 11, we can solve MaxMinO for cactus graphs in polynomial time by using Exact-Cactus-MaxMinO as an engine of the binary search.

5.2. *Pseudo-polynomial time algorithms for series-parallel graphs*

In this subsection, we describe the main idea of a pseudo-polynomial time algorithm solving MaxMinO for series-parallel graphs. The definition of series-parallel graphs is as follows (e.g., see [15]):

Definition 12. *A* series-parallel graph *with distinguished terminals l and r is denoted (G, l, r) and is defined recursively as follows:*

- *The graph consisting of a single edge $\{v_1, v_2\}$ is a series-parallel graph (G, l, r) with $l = v_1$ and $r = v_2$.*

- *A series operation* $(G_1, l_1, r_1) \odot_s (G_2, l_2, r_2)$ *forms a series-parallel graph by identifying* r_1 *with* l_2. *The terminals of the new graph are* l_1 *and* r_2.
- *A parallel operation* $(G_1, l_1, r_1) \odot_p (G_2, l_2, r_2)$ *forms a series-parallel graph by identifying* l_1 *with* l_2 *and* r_1 *with* r_2. *The terminals of the new graph are* l_1 *and* r_1.
- *A jackknife operation* $(G_1, l_1, r_1) \odot_j (G_2, l_2, r_2)$ *forms a series-parallel graph by identifying* r_1 *with* l_2*; the new terminals are* l_1 *and* r_1.

The pseudo-polynomial time algorithm is based on a dynamic programming, and utilizes a decomposition tree [20] defined by the series, parallel and jackknife operations. It is known that determining whether a given graph $G = (V, E)$ is a series-parallel graph can be done in linear time [22, 6]. Moreover, we can also obtain a decomposition tree T of G in linear time if G is a series-parallel graph.

For an arbitrary series-parallel graph (G, l, r), where l and r are left and right terminals, respectively, and two values $w_l \in \{0, 1, \ldots, w_G(l) \overset{\text{def}}{=} \sum_{\{l,u\}\in E(G)} w(\{l, u\})\}$ and $w_r \in \{0, 1, \ldots, w_G(r) \overset{\text{def}}{=} \sum_{\{r,u\}\in E(G)} w(\{r, u\})\}$, we define

$$WSP(G, l, r, w_l, w_r) = \max_{\substack{\Lambda: d_\Lambda(l)=w_l,\\ d_\Lambda(r)=w_r}} \min_{v\in V(G)\setminus\{l,r\}} \{d_\Lambda(v)\},$$

where Λ is an orientation for G.

In a decomposition tree, let us assume that a (sub)tree T_a is composed from its subtrees T_b and T_c by an operation series, parallel, or jackknife, where T_a, T_b and T_c correspond to (G_a, l_a, r_a), (G_b, l_b, r_b) and (G_c, l_c, r_c), respectively. Roughly speaking, for series, parallel, and jackknife operations, the following equations (1), (2), and (3) hold, respectively:

$$WSP(G_a, l_a, r_a, w_l, w_r) = \max_{w_b, w_c} \min \left\{ \begin{array}{l} WSP(G_b, l_b, r_b, w_l, w_b), \\ WSP(G_c, l_c, r_c, w_c, w_r), \\ w_b + w_c \end{array} \right\}. \quad (1)$$

$$WSP(G_a, l_a, r_a, w_l, w_r) = \max_{\substack{w_{bl}+w_{cl}=w_l,\\ w_{br}+w_{cr}=w_r}} \min \left\{ \begin{array}{l} WSP(G_b, l_b, r_b, w_{bl}, w_{br}), \\ WSP(G_c, l_c, r_c, w_{cl}, w_{cr}) \end{array} \right\}. \quad (2)$$

$$WSP(G_a, l_a, r_a, w_l, w_r) = \max_{\substack{w_{br}+w_{cl}=w_r,\\ w_{cr}}} \min \left\{ \begin{array}{l} WSP(G_b, l_b, r_b, w_l, w_{br}), \\ WSP(G_c, l_c, r_c, w_{cl}, w_{cr}) \end{array} \right\}. \quad (3)$$

The above equations (1), (2) and (3) show a principle of optimality, which yields an algorithm based on the dynamic programming. Algorithm 4 shows the algorithm.

Now we discuss the time complexity of AlgSP. As mentioned above, Step 1 is done in $O(m)$ time. In Step 2, we keep $w_G(l) \times w_G(r)$ WSP values for each (G, l, r), and if we have all WSP values for its two children, the evaluation of equations (1), (2) and (3) can be done in $w_{G_b}(r_b) \times w_{G_c}(l_c)$, $w_{G_a}(l_a) \times w_{G_a}(r_a)$ and $w_{G_a}(r_a) \times w_{G_c}(r_c)$ time, respectively. All of these are bounded by Δ^2. The number

Algorithm 4 Algorithm AlgSP

```
1: Construct a decomposition tree $T$ for $G$, and let $l$ and $r$ be two terminals of $G$.
2: For all $w_l = 0, 1, \dots, w_G(l)$ and $w_r = 0, 1, \dots, w_G(r)$, compute $WSP(G, l, r, w_l, w_r)$ and its orientation in a recursive manner by equations (1), (2) and (3).
3: Output an orientation that gives $\max_{w_r \le \Delta, w_l \le \Delta}\{WSP(G, l, r, w_l, w_r) \mid WSP(G, l, r, w_l, w_r) \le \min\{w_l, w_r\}\}$.
```

of recursions is at most $m = O(n)$, so this step is done in $O(n\Delta^2)$. Step 3 can be done also in $O(\Delta^2)$ time. Therefore the total running time of AlgSP is $O(n\Delta^2)$, which is pseudo-polynomial for the input size.

Theorem 13. *Given a series-parallel graph G, AlgSP computes an optimal orientation of G in $O(n\Delta^2)$ time.*

6. Concluding Remarks

In this paper, we have studied the problem of orienting all edges of an input graph so that the minimum weighted outdegree among all vertices is maximized, taken over all possible orientation of G. We have seen that MaxMinO is strongly NP-hard for planar graphs, but it is solvable in polynomial time for cactus graphs. Also, it is not difficult to show the weak NP-hardness of MaxMinO for series-parallel graphs [20], which is an intermediate class between cactus graphs and planar graphs. This hardness result is tight in the sense that it cannot be extended to strong NP-hardness due to the pseudo-polynomial time algorithm for series-parallel graphs shown in Section 5.2.

It is straightforward to extend Algorithm Exact-1-MaxMinO from Section 3 to solve a more general variant of unweighted MaxMinO in which the input consists of an unweighted *hypergraph* $G = (V, F)$, where each hyperedge $f \in F$ is defined as a non-empty subset of V. (If every hyperedge contains exactly two vertices then this is the original unweighted MaxMinO problem.) We only need to modify the construction of the flow network $\mathcal{N}_G$ in Step 1 of Algorithm Exact-1-MaxMinO so that for each hyperedge $f \in F$ and each vertex $v_i \in f$, we include the directed edge (f, v_i) in the edge set E_G:

$$V_G = F \cup V \cup \{s, t\},$$
$$E_G = \big\{(s, f) \mid f \in F\big\} \cup \big\{(v, t) \mid v \in V\big\} \cup \big\{(f, v_i) \mid v_i \in f \text{ and } f \in F\big\}.$$

In other words, for every hyperedge f, if f contains c vertices then the corresponding vertex f in the flow network $\mathcal{N}_G$ will have c outgoing edges. The new $\mathcal{N}_G$ still has $m + n + 2 = O(m)$ vertices but the number of edges increases to $m + \sum_{f_i \in F} |f_i| + n$, which is $O(mn)$ in the worst case. Hence, the running time of the modified Algorithm Exact-1-MaxMinO becomes $O(m^{3/2} \cdot n \cdot \min\{m^{1/6}, n^{1/2}\} \cdot$

$\log(m/n) \cdot \log^2 \Delta)$, but in case every hyperedge contains at most $O(1)$ vertices then the total number of edges in $\mathcal{N}_G$ is only $O(m)$ and the algorithm's running time remains $O(m^{3/2} \cdot \log m \cdot \log^2 \Delta)$ as in Theorem 5. Also note that by representing each gift in the Santa Claus problem (see Section 1) by one hyperedge, it follows that the unweighted version of the Santa Claus problem where each gift has value 0 or 1 for each child can be solved efficiently as above.

There are several open problems; the most important one is to improve the approximation algorithm for the general (weighted) case of MaxMinO provided in Section 4. To obtain a constant factor approximation ratio, however, might need considerable work.

Acknowledgments

We would like to thank the anonymous referees for their suggestions to improve this paper, and thank Tetsuo Shibuya for some inspiring discussions. This work is partially supported by Grant-in-Aid for Scientific Research (C) No. 20500017, Grant-in-Aid for Young Scientists (B) No. 18700015, (A) No. 21680001, Asahi glass foundation, Inamori foundation, and the Special Coordination Funds for Promoting Science and Technology.

References

[1] Y. Asahiro, J. Jansson, E. Miyano, H. Ono, and K. Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. to appear in Journal of Combinatorial Optimization, doi:10.1007/s10878-009-9276-z

[2] Y. Asahiro, E. Miyano, and H. Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. In *Proceedings of Theory of Computing 2008, Proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS2008)*, pp.97–106 (2008)

[3] Y. Asahiro, E. Miyano, H. Ono, and K. Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. *International Journal of Foundations of Computer Science*, 18(2), pp.197–215 (2007)

[4] N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC2006)*, pp.31–40 (2006)

[5] I. Bezáková and V. Dani. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3), pp.11–18 (2005)

[6] R. Borie, R. Parker, and C.Tovey. Solving problems on recursively constructed graphs. *Technical Report TR-2002-04, Dept. Comp. Sci., University of Alabama*, (2002)

[7] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press (1990)

[8] T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pp.483–490 (2008)

[9] U. Feige. On allocations that maximize fairness. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pp.287–293 (2008)

[10] D.K. Friesen and B.L. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Mathematics of Operations Research*, 6(1), pp.74–87 (1981)

[11] M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company (1979)
[12] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3), pp.302–320 (1987)
[13] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5), pp.783–797 (1998)
[14] D. Golovin. Max-min fair allocation of indivisible goods. *Technical Report* (2005)
[15] J.L. Gross and J. Yellen (eds). *Handbook of Graph Theory*. CRC Press (2004)
[16] L. Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In *Algorithms and Computation, 17th International Symposium (ISAAC 2006)*, pp.557–566 (2006)
[17] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2), pp.329–343 (1982)
[18] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. In *22nd Annual ACM Symposium on Computational Geometry (SoCG)*, pp.1–10 (2006)
[19] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pp.216–226 (1978)
[20] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM J. Computing*, 11, pp.298–313 (1982)
[21] V. Venkateswaran. Minimizing maximum indegree. *Discrete Applied Mathematics.*, 143(1–3), pp.374–378 (2004)
[22] T.V. Wimer and S.T. Hedetniemi. K-terminal recursive families of graphs. *Congressue Numerantium*, 63, pp.161–176 (1988)
[23] G. J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20, pp.149–154 (1997)