Efficient Algorithms for Airline Problem

Shin-ichi Nakano[1], Ryuhei Uehara[2], and Takeaki Uno[3]

[1] Department of Computer Science, Faculty of Engineering, Gunma University, Gunma 376-8515, Japan. nakano@cs.gunma-u.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp
[3] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan. uno@nii.jp

Abstract. The airlines in the real world form small-world network. This implies that they are constructed with an ad hoc strategy. The small-world network is not so bad from the viewpoints of customers and managers. The customers can fly to any destination through a few airline hubs, and the number of airlines is not so many comparing to the number of airports. However, clearly, it is not the best solution in either viewpoint since there is a trade off. In this paper, one of the extreme cases, which is the standpoint of the manager, is considered; we assume that customers are silent and they never complain even if they are required to transit many times. This assumption is appropriate for some transportation service and packet communication. Under this assumption, the airline problem is to construct the least cost connected network for given distribution of the populations of cities with no a priori connection. First, we show an efficient algorithm that produces a good network which is minimized the number of vacant seats. The resultant network contains at most n connections (or edges), where n is the number of cities. Next we aim to minimize not only the number of vacant seats, but also the number of airline connections. The connected network with the least number of edges is a tree which has exactly $n-1$ connections. However, the problem to construct a tree airline network with the minimum number of vacant seats is $\mathcal{NP}$-complete. We also propose efficient approximation algorithms to construct a tree airline network with the minimum number of vacant seats.
Keywords: Airline problem, approximation algorithm, efficient algorithm, $\mathcal{NP}$-completeness.

1 Introduction

Small-world networks are the focus of recent interest because of their potential as models for the interaction networks of complex systems in real world [2, 8]. In a small-world network, the node connectivities follow a scale-free power-law distribution. As a result, a very few nodes are far more connected than other nodes, and they are called *hubs*. Through those hubs, any two nodes are connected by a short path (see, e.g., [5]). There are many well known small-world networks including the Internet and World Wide Web. Among them, airlines in the real world form small-world networks [1]. In fact, some airports are known as airline "hubs." The fact implies that they can be constructed in the same manner as the Internet and World Wide Web; in other words, there were

few global strategies for designing efficient airlines. The main reason is that there are so many considerable parameters to be optimized, and some objective functions conflict according to viewpoints; for example, passengers hate to transit, but only complete graph satisfies their demands, which is an impossible solution for airline companies. In fact, it is intractable to design the least cost airline network in general. Even if we fix three hubs, and aim to connect each non-hub to one of the hubs, to design the least cost airline network is $\mathcal{NP}$-hard problem [6, 7].

In this paper, we simplify the design problem of an airline network to a simpler graph theoretical problem. We consider the design problem of an airline network as an optimization problem; we aim to give a reasonable strategy to design a network with minimum loss, which corresponds to the number of vacant seats. Let $\mathbb{Z}$ denote the set of positive integers. Then we define *airline problem* over weighted nodes as follows:

Input: A set V of nodes, and a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of edges $\{u, v\}$ in V^2, and a positive integer weight function $w : V \times V \to \mathbb{Z}$ such that for each $v \in V$, we have $w(v) \leq \sum_{\{v,u\}\in E} w(v, u)$, and the graph $G = (V, E)$ is connected[4].

Intuitively, each node v corresponds to a city, and the weight $w(v)$ gives the number of (potential) passengers in the city. If there are already airports, we can count the number of users; otherwise, the weights can be estimated from the populations of the cities. Each edge $\{u, v\}$ corresponds to an airline. An airplane can transport $w(u, v)$ passengers at one flight. Airplanes make regular flights along the edges, both ways, and simultaneously. Hence the number of passengers, $w(v)$, does not fluctuate in a long term. We consider that the condition $w(v) \leq \sum_{\{v,u\}\in E} w(v, u)$ for each v in V is appropriate condition as an airline network, if all cities are connected by the airline network. The condition is enough to supply the least service. In other words, we assume that passengers are silent; they never complain even if they have to transit many times. Therefore the airline problem can be an idealized model for planning some real network problem like transportation service, peer to peer file transfer network, and data network flow in the sense that producing a reasonable (or cheapest) network that can satisfy given demands. After designing the network, we will face the assignment problems. However, the assignment problems are separated in this context; we only mention that each cheap network dealt in this paper has at least one reasonable solution obtained by a random walk approach, which is omitted here.

To evaluate the "goodness" of a solution for the airline problem, we define the *loss* $L(v)$ at v by $(\sum_{\{v,u\}\in E} w(v, u)) - w(v)$. If the solution is feasible, we have $L(v) \geq 0$ for all $v \in V$. Intuitively, the loss $L(v)$ gives the total number of vacant seats of departure flights from the city v. We denote by $L(G) := \sum_{v\in V} L(v)$ the *total loss* of the graph (or solution) $G = (V, E)$. We here observe that $L(G)$ is given by $\sum_{v\in V} L(v) = \sum_{v\in V}((\sum_{\{v,u\}\in E} w(v, u)) - w(v)) = \sum_{v\in V} \sum_{\{v,u\}\in E} w(v, u) - \sum_{v\in V} w(v) = 2 \sum_{e\in E} w(e) - \sum_{v\in V} w(v)$.

We first consider the airline problem to generate a connected network with the minimum loss $L(G)$. We show an efficient algorithm that minimizes the total loss of the flights on the network. The algorithm generates a connected network of at most $|V|$

[4] The weight of an edge $e = \{u, v\}$ should be denoted by $w(e) = w(\{u, v\}) = w(\{v, u\})$. However, we denote them by $w(e) = w(u, v)(= w(v, u))$ for short.

edges in $O(|V|)$ time and $O(|V|)$ space. Since the minimum number of the edges of a connected graph with $|V|$ vertices is $|V| - 1$, our algorithm produces the least cost connected network with $|V| - 1$ or $|V|$ airlines. Hence it is natural to ask that if we can restrict ourselves to construct a weighted tree with the minimum loss. It is worth mentioning that tree network has an advantage that a shortest route between two vertices are uniquely determined. However, interestingly, the problem is intractable; the airline problem to construct a tree airline network (of $|V| - 1$ edges) with the minimum weight (or the minimum loss) is $\mathcal{NP}$-complete. For the $\mathcal{NP}$-complete problem, we give two efficient approximation algorithms. First one always finds a tree airline network T of V of approximation ratio 2 in $O(|V|)$ time and space. More precisely, the algorithm constructs a weighted tree T that has additional weight $w_{\max}$ than the optimal weight among all weighted connected networks that is not necessarily a tree, where $w_{\max} = \max_{v \in V} w(v)$. The second one is based on an FPTAS for the weighted set partition. Assume we obtain a partition X and Y of V with $\left|\sum_{x \in X} w(x) - \sum_{y \in Y} w(y)\right| \leq \delta$ for some $\delta \geq 0$ by an FPTAS. Then, from X and Y, we can construct a weighted tree T with $L(T) \leq \max\{\delta, 2\}$ in $O(|V|)$ time and space.

2 Minimum cost network

In this section, we show efficient algorithms for constructing a connected network of minimum loss for given weighted nodes V. Hereafter, we denote $|V|$ by n and $\sum_{v \in V} w(v)$ by W. Since the case $n = 1$ is trivial, we assume that $n > 1$. The main theorem in this section is the following:

Theorem 1. *Let V be a set of n nodes, and w be a positive integer weight function $w : V \to \mathbb{Z}$. Then a connected network E over V of the minimum loss $L(G)$ with $|E| \leq n$ can be found in $O(n)$ time and $O(n)$ space.*

Procedure `Span`(u, v, w)

Input : Two vertices u and v, and weight w with $w(u), w(v) \geq w$.
Output: An edge $\{u, v\}$ of weight w.

1 $w(u, v) := w$;
2 $w(u) := w(u) - w$; $w(v) := w(v) - w$;
3 **if** *w(u)=0* **then** remove u;
4 **if** *w(v)=0* **then** remove v;
5 **return** $(\{u, v\})$;

Throughout the paper, we will use procedure `Span` as a basic operation. Mainly, given two vertices u and v, we span an edge $\{u, v\}$ of weight $\min\{w(u), w(v)\}$ and remove one of them. We first have the following lemma:

Lemma 1. *If $L(G) = 1$, the solution is optimal.*

Proof. We remind that $L(G)$ is given by $\sum_{v\in V} L(v) = \sum_{v\in V}((\sum_{\{v,u\}\in E} w(v,u)) - w(v)) = \sum_{v\in V}\sum_{\{v,u\}\in E} w(v,u) - \sum_{v\in V} w(v) = 2(\sum_{e\in E} w(e)) - W$. Thus when $L(G) = 1$, W is odd, which is the input, and hence we cannot improve it. □

We start with the following three special cases:

Star condition: Let $v_{\max}$ be a heaviest vertex, i.e., $w(v_{\max}) \geq w(v)$ for all $v \in V$. We say *star condition* if we have $w(v_{\max}) - \sum_{v\in V\setminus\{v_{\max}\}} w(v) \geq 0$.

Uniform condition: When $w(v) = w > 0$ for all $v \in V$, we call it *uniform condition*.

Many-ones condition: If V contains at least two vertices of weight 1, we call that *many-ones condition*. To distinguish it from the uniform condition, we assume that V also contains at least one vertex of weight greater than 1.

We have either the star or uniform condition if $|V| \leq 2$. Hence, hereafter, we assume that $|V| > 2$. Under the star condition, Algorithm `Star` computes the solution with minimum loss $L(G) = w(v_{\max}) - \sum_{v\in V\setminus\{v_{\max}\}} w(v)$ in $O(n)$ time and space. It is also easy to see that $|E|$ contains $n - 1$ edges.

Algorithm 2: `Star`

Input : A set V of n nodes, a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of $m = n - 1$ edges $\{u, v\}$ such that (V, E) is connected, and a positive integer weight function $w : E \to \mathbb{Z}$.

let $v_{\max}$ be a vertex such that $w(v_{\max}) \geq w(v)$ for all $v \in V$;
foreach $v \in V \setminus \{v_{\max}\}$ **do** `Span`(v,$v_{\max}$,$w(v)$);
pickup any $e = (v', v_{\max})$ with $w(e) > 0$;
$w(e) := w(e) + w(v_{\max})$;
return $(E := \{e \mid w(e) > 0\})$;

Algorithm 3: `Uniform`

Input : A set V of n nodes, a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of m ($m = n$ or $m = n - 1$) edges $\{u, v\}$ such that (V, E) is connected, and a positive integer weight function $w : E \to \mathbb{Z}$.

let w be a positive integer such that $w = w(v)$ for all $v \in V$;
if $w = 1$ **then** make any spanning tree T over V, and $w(e) := 1$ for each edge $e \in T$;
else
    foreach *odd* $i = 1, 3, 5, \ldots$ **do** `Span`(v_i,v_{i+1},$\lfloor w/2 \rfloor$);
    foreach *even* $i = 2, 4, 6, \ldots$ **do** `Span`(v_i,v_{i+1},$\lceil w/2 \rceil$);
    `Span`(v_n,v_1,$\lceil w/2 \rceil$);
return $(E := \{e \mid w(e) > 0\})$;

Lemma 2. *In the uniform condition, Algorithm* `Uniform` *produces a connected network E over V of the minimum loss $L(G)$ in $O(n)$ time and $O(n)$ space, where $|V| = n$. Moreover, $|E| \leq n$.*

Proof. Omitted. □

Algorithm 4: `Many-ones`

Input : A set V of n nodes, a positive integer weight function $w : V \to \mathbb{Z}$.
Output: V' of n' nodes and E of $n - n'$ edges s. t. V' has at most one vertex of weight 1.

1 let $v_{\max}$ be a vertex such that $w(v_{\max}) \geq w(v)$ for all $v \in V$;
2 let $V_1 := \{v \mid w(v) = 1\}$ and $V_2 := \{v \mid w(v) \geq 2\} \setminus \{v_{\max}\}$;
3 **while** $|V_1| > 1$ *and* $|V_2| > 0$ **do**
4 **if** *V satisfies the star condition* **then** call `Star` as a subroutine and halt;
5 for any vertices $v \in V_1$ and $v' \in V_2$, `Span`(v,v',1);
6 **if** $w(v') = 1$ **then** move v' from V_2 to V_1;
7 **if** $|V_2| = 0$ **then**
8 **if** *V satisfies the star condition* **then** call `Star` as a subroutine and halt;
9 call `Span`($v_{\max}$,v,1) for any $w(v_{\max}) - 1$ vertices $v \in V_1$; `// to make` $w(v_{\max}) = 1$
10 call `Uniform` as a subroutine and halt;
11 **return** (V and $E := \{e \mid w(e) > 0\}$);

We next turn to the many-ones condition. We first partition V into two disjoint subsets $V_1 := \{v \mid w(v) = 1\}$ and $V_2 := \{v \mid w(v) \geq 2\}$. Then we have many-ones condition iff $|V_1| > 1$ and $|V_2| > 0$. We also pick up a vertex $v_{\max}$ of the maximum weight as a special vertex to check if the vertex set satisfies the star condition. The purpose here is to reduce the number of vertices of weight 1 to one. Hence we join the vertices in V_1 to the other vertices and output the remaining vertices, which contains at most one vertex of weight 1. This is the preprocess of the main algorithm.

The vertex $v_{\max}$ can be found in $O(n)$ time, and $\sum_{v \in V \setminus \{v_{\max}\}} w(v)$ can be maintained decrementally. Hence Algorithm `Many-ones` runs in $O(n)$ time and space by maintaining V_1 and V_2 by two queues. `Many-ones` terminates if the vertex set satisfies the star condition, the uniform condition, or the set contains at most one vertex of weight 1. In the former two cases, we already have a solution.

Now, we can assume that the input V satisfies neither the star, uniform, nor many-ones conditions. Then, we have the following lemmas:

Lemma 3. *Algorithm* `Network` *outputs a connected network* (V, E) *with* $|E| \leq n$.

Proof. In the while-loop, the algorithm either (0) calls `Star` or `Uniform` and halts, or (1) sets $w(v, v')$ for some v, v' and removes one of v and v'. The algorithm also joints all vertices in some vertex set $\hat{V}$ with $\left|\hat{V}\right|$ edges. Hence we have an invariant that the total number of removed vertices is greater than or equal to the total number of added edges. Hence $|E|$ contains at most n edges. It is easy to see that the resultant network is connected. □

Lemma 4. *Algorithm* `Network` *always outputs a network with the minimum loss* $L(G)$.

Proof. Let V_1 be the set of vertices of weight 1. Then we have an invariant $|V_1| \leq 1$ throughout the execution of the algorithm. Let $v_{\max}$ be the heaviest vertex chosen in step 2. We have two cases; (I) all vertices in $V \setminus \{v_{\max}\}$ have the same weight w, and (II) there are two vertices v_i and v_j in $V \setminus \{v_{\max}\}$ with $w(v_i) < w(v_j)$.

Algorithm 5: `Network`

Input : A set V of n nodes, and a positive integer weight function $w : V \to \mathbb{Z}$.
Output: A set E of m ($m = n$ or $m = n - 1$) edges $\{u, v\}$ such that (V, E) is connected, and positive integer weight function $w : E \to \mathbb{Z}$.

1 $n := |V|$, and $W := \sum_{v \in V} w(v)$;
2 find $v_{\max}$ such that $w(v_{\max}) \geq w(v)$ for any other $v \in V$;
3 let $V_1 := \{v \mid w(v) = 1\}$ (we have $|V_1| < 2$);
4 **while** true **do**
5 **if** *V satisfies the star condition* **then** call `Star` as a procedure and halt;
6 **if** *V satisfies the uniform condition* **then** call `Uniform` as a procedure and halt;
7 **if** *V satisfies the uniform condition* **then**
8 **if** $(n-2)w < w(v_{\max})$ **then** `// we also have` $w(v_{\max}) < (n-1)w$
9 let v_i and v_j be any two vertices in $V \setminus \{v_{\max}\}$;
10 `Span`$(v_i, v_j, \lceil((n-1)w - w(v_{\max}))/2\rceil)$; `// to have star condition`
11 **else**
12 $r := w(v_{\max}) \bmod w$;
13 **if** $r = 0$ **then**
14 let S consist of any $(w(v_{\max})/w) - 1$ vertices from $V \setminus \{v_{\max}\}$;
15 **else**
16 let S consist of any $\lfloor w(v_{\max})/w \rfloor$ vertices from $V \setminus \{v_{\max}\}$;
17 **foreach** $v \in S$ **do** `Span`$(v, v_{\max}, w)$; ; `// we have` $S = \emptyset$
18 **if** $w(v_{\max}) = 1$ **then** put $v_{\max}$ into V_1;
19 **else**
20 **if** $V_1 \neq \emptyset$ **then**
21 let v_i be the vertex in V_1, and v_j be any vertex in $V \setminus \{v_{\max}\}$;
22 **else**
23 let v_i and v_j be any vertices in $V \setminus \{v_{\max}\}$ with $w(v_i) < w(v_j)$;
24 **if** $W - 2w(v_i) > 2w(v_{\max})$ **then**
25 `Span`$(v_i, v_j, w(v_i))$;
26 **if** $w(v_j) = 1$ **then** put v_j into V_1;
27 **else**
28 `Span`$(v_i, v_j, \lceil(W - 2w(v_{\max}))/2\rceil)$; `// to have star condition`
29 update V, n, W, $v_{\max}$ if necessary;

(I) This case is handled in steps 8 to 18. We first note that V is in neither the uniform nor star cases. Thus, we have $w(v_{\max}) > w$ and $w(v_{\max}) < (n-1)w$. Mainly, in the case, the algorithm takes the vertices of weight w by matching with $v_{\max}$ as follows. Let q be $\lfloor w(v_{\max})/w \rfloor$ and r be $w(v_{\max}) \bmod w$.

If $r = 0$, the last vertex of weight w cannot be matched to $v_{\max}$ since all vertices spanned by the edges have weight 0, and we will have a loss. Hence, in the case, the algorithm matches $q - 1$ vertices of weight w to $v_{\max}$, and then $w(v_{\max})$ becomes w. That is, we will have the uniform case in the next iteration. Through the process, the algorithm generates no loss, which is handled in steps 14, 17, 18. Hence if the uniform

case will be handled properly, the algorithm generates no loss, which will be discussed later.

If $r \neq 0$, we can match q vertices of weight w to $v_{\max}$ by edges of weight w. After the matching, we remove q vertices from $V \setminus \{v_{\max}\}$, and $w(v_{\max})$ is updated by $w(v_{\max}) - qw$. If $w(v_{\max}) - qw$ is enough large comparing to the total weight of the remaining vertices of weight w, the process is done properly in steps 16, 17, 18. However, the process fails when $w(v_{\max}) - qw$ is too light; for example, when $V = \{v_1, v_2, v_3\}$ with $w(v_1) = 8, w(v_2) = w(v_3) = 5$, we cannot make an edge $\{v_1, v_2\}$ of weight 5. The resultant vertex v_3 will generate loss 2. In the case, we have to make $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\}$ with $w(v_1, v_2) = w(v_1, v_3) = 4$ and $w(v_2, v_3) = 1$. To consider the case, we partition $V \setminus \{v_{\max}\}$ into V_a of q vertices and V_b of $n - q - 1$ vertices. The loss will be generated, after removing all vertices in V_a which are matched with $v_{\max}$, if (1) $\{v_{\max}\} \cup V_b$ satisfies the star condition, and (2) $w(v_{\max}) < w$. They occur only if $|V_b| = 1$, which is equivalent to $(n-2)w < w(v_{\max}) < (n-1)w$. This case is handled in steps 9 and 10. In the case, we can have the optimal solution with the following assignments of weights; pick up any two vertices v_i and v_j from $V \setminus \{v_{\max}\}$, and add the edge $\{v_i, v_j\}$ of weight $\lceil((n-1)w - w(v_{\max}))/2\rceil$. Then we have the star condition, and we have $L(G) \leq 1$, which is the optimal.

(II) This case is handled in steps 20 to 28. Let v_i and v_j be any two vertices of different weights with $w(v_i) < w(v_j)$. If $|V_1| = 1$, the algorithm takes the unique vertex of weight 1 as v_i. When $w(v_j)$ is not so heavy, we add an edge $\{v_i, v_j\}$ with $w(v_i, v_j) = w(v_i)$ and remove v_i in step 25. The exception is that removing $w(v_i)$ results in the star condition with loss, which is equivalent to $\sum_{v \in V \setminus \{v_{\max}\}} w(v) = W - w(v_{\max}) - 2w(v_i) < w(v_{\max})$. Hence the case occurs when $W - 2w(v_i) < 2w(v_{\max})$. On the other hand, we did not have the star condition before removing $2w(v_i)$ from $w(v_i)$ and $w(v_j)$. Thus, before removing, we had $\sum_{v \in V \setminus \{v_{\max}\}} w(v) = W - w(v_{\max}) > w(v_{\max})$, or consequently, $W > 2w(v_{\max})$. In the case, we can have the star condition without loss by the edge $\{v_i, v_j\}$ with $w(v_i, v_j) = \left\lceil \frac{\sum_{v \in V \setminus \{v_{\max}\}} w(v) - w(v_{\max})}{2} \right\rceil = \left\lceil \frac{W - 2w(v_{\max})}{2} \right\rceil$, and then we have the optimal in step 28.

Hence, in most cases, the algorithm achieves the optimal network. The last case is in the following case: The algorithm does not call `Uniform` at first, and it calls `Uniform`, which outputs a spanning tree since all vertices have the weight 1. However, this case is impossible since we have an invariant $|V_1| \leq 1$.

Thus, Algorithm `Network` always outputs a network with $L(G) \leq 1$, which is optimal by Lemma 1, if V does not satisfy one of three special conditions. □

Lemma 5. *Algorithm* `Network` *runs in* $O(n)$ *time and space.*

Proof. If we admit to sort the vertices, it is easy to implement the algorithm to run in $O(n \log n)$ time and $O(n)$ space. To improve the time complexity to $O(n)$, we show how to maintain $v_{\max}$ and determine if all vertices in a vertex set $V \setminus \{v_{\max}\}$ have the same weight efficiently. In step 2, the algorithm first finds $v_{\max}$ in $O(n)$ time. Then we can check if V satisfies the star condition or the uniform condition in $O(n)$ time. In the while-loop, two special vertices $v_{\max}$ and the unique vertex, say v_1, in V_1 (if exist) are maintained directly, and all other vertices in $V' = V \setminus \{v_{\max}, v_1\}$ are maintained in a doubly linked list. The number n of vertices are also maintained.

We first assume that all vertices in V' have the same weight w. If $(n-2)w < w(v_{\max})$, the algorithm halts in $O(n)$ time. Hence we assume that $w(v_{\max}) \leq (n-2)w$. (Note that

$(n-1)w \leq w(v_{\max})$ implies the star condition.) In the case, the algorithm computes $r = w(v_{\max}) \bmod w$ in $O(1)$ time. If $r = 0$, the algorithm removes $(w(v_{\max})/w) - 1$ vertices from V'. After that, $w(v_{\max})$ becomes $w(v_{\max}) = w$, and we have the uniform case. Thus the algorithm can call `Uniform` without checking the condition. The time complexity can be bounded above by $O(|V'|)$. If $r \neq 0$, the algorithm removes $\lfloor w(v_{\max})/w \rfloor$ vertices from V'. After that, $w(v_{\max})$ becomes $w(v_{\max}) < w$, and the other vertices have the same weight w. We update $v_{\max}$ by any vertex in $V' \setminus \{v_{\max}\}$. Through the step, the running time is proportional to the number of the vertex removed.

Next, we assume that there are some different weight vertices in V'. The pair v_i and v_j of different weights can be found by traversing the doubly linked list. Let $v_2, v_3, \ldots$ be the consecutive vertices in the list. If $V_1 \neq \emptyset$, the pair $v_i = v_1$ and $v_j = v_2$ can be found in $O(1)$ time. Otherwise, the algorithm checks if $w(v_1) = w(v_2)$, $w(v_2) = w(v_3)$, or $w(v_3) = w(v_4)$, ... until it finds $w(v_k) \neq w(v_{k+1})$. Then set $v_i := \min\{v_k, v_{k+1}\}$ and $v_j := \max\{v_k, v_{k+1}\}$. Moreover, in the case, the algorithm knows that $w(v_1) = w(v_2) = \cdots = w(v_k)$. When $W - 2w(v_k) \leq 2w(v_{\max})$, the algorithm connects all vertices and halts in time $O(|V'|)$. Hence we assume that $W - 2w(v_k) > 2w(v_{\max})$. Then the algorithm removes v_i or v_j in $O(1)$ time from the linked list. After updating n and W, the algorithm has to check if all vertices in V' have the same weight. Since the algorithm knows that $w(v_1) = w(v_2) = \cdots = w(v_{i-1})$, it is enough to check from v_{i-1}. Thus the total time to check if V' contains at least two vertices of different weights is bounded above by $O(n)$.

Hence, the algorithm runs in $O(n)$ time and space. □

By Lemmas 2, 3, 4, and 5, we immediately have Theorem 1.

3 Minimum cost spanning tree

In this section, we first prove that the problem for finding a minimum loss tree airline network is $\mathcal{NP}$-complete. Next, we show approximation algorithms for the problem.

3.1 $\mathcal{NP}$-hardness for finding a spanning tree of minimum loss

We first modify the optimization problem to the decision problem as follows; the input of the algorithm consists of a set V of nodes, a positive integer weight function $w(v)$ for each $v \in V$, and an integer k. Then the decision problem is to determine if there is the set E of edges and a positive integer weight function $w(u, v)$ such that they provide a feasible solution of the airline problem with $L(G) \leq k$ and (V, E) induces a (connected) tree.

Theorem 2. *The decision problem for finding a tree airline network is $\mathcal{NP}$-complete.*

Proof. The problem is clearly in $\mathcal{NP}$. We reduce it to the following well known $\mathcal{NP}$-complete problem [4, [SP12]]:

Problem: Weighted Set Partition
Input: Finite set A and weight function $w'(a) \in Z^+$ for each $a \in A$;
Output: Determine if there is a subset $A' \subset A$ s. t. $\sum_{a \in A'} w'(a) = \sum_{a \in A \setminus A'} w'(a)$.

Let $W := \sum_{a \in A} w'(a)$. Without loss of generality, we assume that W is even. For given $A = \{a_1, a_2, \ldots, a_n\}$ and the weight function w', we construct the input V and w of the airline problem as follows; $V = A \cup \{u, v\}$, and $w(a) = w'(a)$ for each vertex a in A. We define $w(u) = w(v) = \frac{W}{2} + 1$. The reduction can be done in polynomial time and space. We show that A can be partitioned into two subsets of the same weight if and only if V has a tree airline network with no loss. Let E be the set of weighted edges of the minimum loss. We first observe that if $G = (V, E)$ achieves $L(G) = 0$, E has to contain the positive edge $\{v, u\}$. Otherwise, the edges incident to u or v have to have total weight $W + 2 > W$, and then we have $L(G) > 0$.

First we assume that A has a partition A_1 and A_2 such that $A_1 \cup A_2 = A$, $A_1 \cap A_2 = \emptyset$, and $\sum_{a \in A_1} w(a) = \sum_{a \in A_2} w(a) = W/2$. We show that V has a tree airline network with no loss. We define the weight function w as follows; $w\{u, v\} = 1$, $w\{a, u\} = w(a)$ for all $a \in A_1$, and $w\{a, v\} = w(a)$ for all $a \in A_2$. By assumption and construction, the set E of positive weighted edges is a tree airline network with no loss.

Next we assume that V has a tree airline network with no loss, and show that A can be partitioned into A_1 and A_2 of the same weight. By the observation, the edge $\{u, v\}$ has a positive weight, say w'. We then partition the set A into A_1 and A_2 as follows; A_1 consists of vertices $a \in A$ of odd distance from u, and A_2 consists of vertices $a \in A$ of even distance from u. Since T is a tree, A_1 and A_2 satisfy $A_1 \cap A_2 = \emptyset$, $A_1 \cup A_2 = A$, and two sets A_1 and A_2 are independent sets. Moreover, since T has no loss, $\sum_{e \in (A_1 \setminus \{v\}) \times \{u\}} w(e) = \sum_{e \in (A_2 \setminus \{u\}) \times \{v\}} w(e) = \frac{W}{2} + 1 - w'$, and $\sum_{e \in (A_1 \setminus \{v\}) \times (A_2 \setminus \{u\})} = w' - 1$. Hence we have $\sum_{a \in A_1 \setminus \{v\}} w(a) = \sum_{a \in A_2 \setminus \{u\}} w(a) = \frac{W}{2}$. Thus A_1 and A_2 gives a solution of the weighted set partition problem.

Therefore, the weighted set partition problem can be polynomial time reducible to the problem for finding a tree airline network of minimum loss, which completes the proof. □

3.2 Approximation algorithms for a tree airline network

In this section, we show two approximation algorithms that aim at different goals. First one gives us a simple and efficient algorithm with approximation ratio 2. Second one is based on an FPTAS for the set partition problem, which gives us a polynomial time algorithm with arbitrary small approximation ratio.

Simple 2-approximation algorithm The simple algorithm is based on the algorithm stated in Section 2. The algorithm in Section 2 outputs a connected network with at most n edges. The algorithm outputs the nth edge when (1) it is in the uniform case, or (2) the edge $\{v_i, v_j\}$ is produced in step 10 or step 31 by Algorithm `Network`. We modify each case as follows and obtain a simple approximation algorithm.
(1) In the uniform case with $w > 1$, pick up any pair of vertices $\{v_i, v_{i+1}\}$ such that $w(v_i, v_{i+1}) = \lfloor w/2 \rfloor$. Then, cut the edge, and add their weight to adjacent edges; $w(v_{i-1}, v_i) := w(v_{i-1}, v_i) + \lfloor w/2 \rfloor$, and $w(v_{i+1}, v_{i+2}) := w(v_{i+1}, v_{i+2}) + \lfloor w/2 \rfloor$. In the case, $L(G)$ increases by $2 \lfloor w/2 \rfloor \leq w$.

(2) In both cases, the vertices v_i and v_j will be joined to $v_{\max}$ in the next iteration since V satisfies the star condition. Hence we add the weight of the edge $\{v_i, v_j\}$ to $\{v_i, v_{\max}\}$ and

$\{v_j, v_{\max}\}$. In the former case, $L(G)$ increases by $2\lceil(w(v_{\max}) - (n-2)w)/2\rceil \leq w(v_{\max}) - (n-2)w + 1 < w(v_{\max})$. In the latter case, $L(G)$ increases by $2\lceil(W - 2w(v_{\max}))/2\rceil < w(v_{\max})$.

From above analysis, we immediately have the following theorem:

Theorem 3. *The modified algorithm always outputs a connected tree $T = (V, E)$ with $L(T) \leq w(v_{\max})$ in $O(n)$ time and space.*

Let E' be any feasible solution (which does not necessarily induce a tree) of the airline problem. Then, clearly, $\sum_{e\in E'} w(e) \geq w(v_{\max})$. Thus we have the following corollary.

Corollary 1. *Let E be the set produced by the modified algorithm, and E_{opt} be an optimal solution (with the minimum loss) of the airline problem. Let $T := (V, E)$ and $G := (V, E_{opt})$. Then, $\sum_{e\in E_{opt}} w(e) \leq \sum_{e\in E} w(e) < 2\sum_{e\in E_{opt}} w(e)$.*

Approximation algorithm based on FPTAS A weighted set partition problem has an FPTAS based on a pseudo-polynomial time algorithm. The idea is standard and can be found in a standard text book, for example, [3, Chapter 35.5]. Hence, using the FPTAS algorithm, we can compute a partition X and Y of V with $\frac{|\sum_{v\in X} w(v) - \sum_{v\in Y} w(v)| - |\sum_{v\in X^*} w(v) - \sum_{v\in Y^*} w(v)|}{\sum_{v\in V} w(v)} < \epsilon$ for any positive constant ϵ in polynomial time of $|V|$ and ϵ, where X^* and Y^* are an optimal partition of V.

In this section, we show a polynomial time algorithm that constructs a tree airline network from the output of the the FPTAS for the weighted set partition problem for the same input V and w.

By the results in Section 2, if V satisfies either the star condition or the uniform condition, we can obtain a tree airline network that is an optimal solution. On the other hand, if V contains many vertices of weight 1, we can reduce them by Algorithm `Many-ones`. Hence, without loss of generality, we assume that V is neither in the star condition nor in the uniform condition, and V contains at most one vertex of weight 1.

We first regard V and w as an input to the weighted set partition problem. Then we run the FPTAS algorithm for the weighted set partition problem. Let X and Y be the output of the algorithm. That is, $\delta := \left|\sum_{v\in X} w(v) - \sum_{v\in Y} w(v)\right|$ is minimized by the FPTAS algorithm. We note that an optimal partition X^* and Y^* of V gives the lower bound of the optimal solution for the airline problem; we cannot have $L(T) < \left|\sum_{v\in X^*} w(v) - \sum_{v\in Y^*} w(v)\right|$ for any weighted tree T that spans V. We can make a tree airline network that achieves the same performance by the FPTAS.

Theorem 4. *Let X and Y be the partition of V produced by an FPTAS for the weighted set partition problem, and $\delta := \left|\sum_{v\in X} w(v) - \sum_{v\in Y} w(v)\right|$. Then, from X and Y, we can construct a connected network E such that $T = (V, E)$ is a tree with $L(T) \leq \max\{\delta, 2\}$. The tree T can be constructed in $O(|V|)$ time and space.*

Proof. The algorithm consists of two phases.

Let v_0 be the vertex in V of the minimum weight, i.e., $w(v_0) \leq w(v)$ for any $v \in V$. If v_0 is uniquely determined (or $w(v_0) \neq w(v)$ for each $v \in V\setminus\{v_0\}$), the algorithm performs the first phase, and otherwise, the algorithm runs from the second phase.

We first show the first phase, which runs if v_0 is uniquely determined. Without loss of generality, we assume that $v_0 \in X$. We let $X = \{x_0 = v_0, x_1, x_2, \dots, \}$ and $Y = \{y_1, y_2, \dots, \}$. (We note that $x_1, x_2, \dots$ and $y_1, y_2, \dots$ are ordered in arbitrary way.) The first phase is given in Algorithm `Caterpillar`; it starts from a path $\{x_0, y_1\}$, and extend it as possible as it can until the next vertex pair becomes the same weight. (The resultant graph makes a graph that is known as a caterpillar which consists of path where each vertex on the path has some pendant vertices.) After the first phase, if $X = \emptyset$ or $Y = \emptyset$, we complete the tree by joining all vertices in the non-empty set (if it exists) to the last vertex touched in the empty set. In the case, the tree T admits $L(T) = \delta$. Hence we assume that $X \neq \emptyset$ and $Y \neq \emptyset$, and $w(x_i) = w(y_j) = w$ for some i, j, and w. By the algorithm and initial condition, we have $i > 0$ and one of $w(x_i)$ and $w(y_j)$ is updated, and the other one is not updated. Hence $w > w(v_0)$.

Algorithm 6: `Caterpillar`

```
i := 0; j := 1;
while w(x_i) ≠ w(y_j) and X ≠ ∅ and Y ≠ ∅ do
    if w(x_i) < w(y_j) then
        Span(x_i,y_j,w(x_i));
        i := i + 1;
    else
        Span(x_i,y_j,w(y_j));
        j := j + 1;
```

Now, we turn to the second phase. We now renumber the vertices as $X = \{x_0, x_1, x_2, \dots\}$ and $Y = \{y_0, y_1, y_2, \dots\}$ such that $w(x_0) \leq w(x_i)$ and $w(y_0) \leq w(y_i)$ for each $i > 0$. By assumption, the input V contains at most one vertex of weight 1. Hence now we have $w(x_0) = w(y_0) > 1$ by the first phase.

If the algorithm runs the first phase, one of x_0 and y_0 is an endpoint of the caterpillar. Without loss of generality, we assume that x_0 is the endpoint. (We regard that x_0 is the endpoint of the graph of size 1 if the algorithm runs from the second phase.) The algorithm extends the tree from x_i as follows. It searches y_j with $w(x_0) \neq w(y_j)$ from $\{y_1, y_2, \dots\}$.

If the algorithm finds $w(y_j)$ with $w(x_0) \neq w(y_j)$, it calls $\mathtt{Span}(x_0, y_j, \min\{w(x_0), w(y_j)\} = w(x_0))$. Then the algorithm repeats the first phase with the vertex pair x_1 and y_j; we remark that the algorithm knows that $w(y_0) = w(y_1) = \cdots = w(y_{j-1}) = w$, which will be preferred than the other vertices in the next phase, and the algorithm can omit to check if they have the same weight.

When the algorithm do not find $w(y_j)$ with $w(x_0) \neq w(y_j)$, we have $w(x_0) = w(y)$ for all $y \in Y$. In the case, the algorithm searches x_i with $w(x_i) \neq w(x_0)$. If the algorithm finds $w(x_i)$ with $w(x_i) \neq w(x_0)$, it calls $\mathtt{Span}(x_i, y_0, \min\{w(x_i), w(y_0)\} = w(y_0))$. Then the algorithm repeats to join the vertices in Y to x_i until x_i is removed. If x_i is removed while $Y \neq \emptyset$, the last touched vertex y_j in Y satisfies $w(y_j) < w(y)$ for each $y \in Y$ and $w(y_j) < w(x_0)$ since all vertices in Y had the same weight equal to x_0. Thus the

algorithm repeats the first phase for the pair $\{x_0, y_j\}$. If we have $Y = \emptyset$ and $w(x_i) > 0$, the algorithm picks up the last vertex y in Y and connect all vertices in X to y with their weights. In the case, the algorithm achieves the loss δ.

Now, we have the last case: $w(x) = w(y) = w$ for all $x \in X$ and $y \in Y$. We renumber $X = \{x_1, x_2, \ldots, x_k\}$ and $Y = \{y_1, y_2, \ldots, y_{k'}\}$. By the process above, every connected subtree has exactly one node in $X \cup Y$. Thus we have a weighted tree T spanning V by joining those vertices. Moreover, since the vertices are preferred if they are touched, both of X and Y contain at least one vertex whose weight was not updated by the algorithm, respectively. If $|k - k'| > 1$, we can improve δ by moving the untouched vertex. Hence we have $k = k'$ or $|k - k'| = 1$. First, we assume that $|k - k'| = 0$. If $k = k' = 1$, the algorithm completes the tree by joining $\{x_1, y_1\}$ with $w(x_1, y_1) = w$. When $k = k' > 1$, the algorithm completes a spanning tree T by the path $(x_1, y_1, \ldots, x_k, y_k)$ with $w(x_1, y_1) = w(x_k, y_k) = w$, $w(x_i, y_i) = w - 1$ and $w(y_i, x_{i+1}) = 1$ for $1 < i < k$. Then we have $L(T) = 2$. If $k = k' + 1$, the path $(x_1, y_1, \ldots, x_{k-1}, y_{k-1}, x_k)$ with $w(x_1, y_1) = w$, $w(y_{k-1}, x_k) = w$, $w(x_i, y_i) = w-1$ and $w(y_i, x_{i+1}) = 1$ for $1 < i < k-1$ gives us the tree T with $L(T) = \delta$. The case $k = k' - 1$ is symmetric.

Thus, the algorithm outputs a tree airline network T with $L(T) \leq \max\{2, \delta\}$. By similar implementation using queue of the vertices of the same weight in the proof of Lemma 5, the algorithm runs in $O(n)$ time and $O(n)$ space. □

4 Concluding remarks and acknowledgment

In this paper, we do not deal with the assignment problem over the constructed network. When each vertex has its destination, the assignment problem is further challenging problem. The authors are partially supported by the Ministry, Grant-in-Aid for Scientific Research (C).

References

1. L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. Classes of small-world networks. *Applied Physical Science*, 97(21):11149–11152, October 2000.
2. A.L. Barabasi. *Linked: The New Science of Networks.* Perseus Books Group, 2002.
3. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.
4. M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.
5. M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
6. M. O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32:393–404, 1987.
7. J. Sohn and S. Park. The Single Allocation Problem in the Interacting Three-Hub Network. *Networks*, 35:17–25, 2000.
8. D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 2004.