# Efficient Algorithms for Airline Problem

Shin-ichi Nakano* Ryuhei Uehara† Takeaki Uno‡

December 8, 2006

**Abstract**

It is known that the airlines in the real world form small-world network. This fact implies that they are constructed with an ad hoc strategy. The small-world network is good from the viewpoint of customers; they can fly to any destination through a few airline hubs. However, it is not the best solution for the managers. In this paper, we assume that customers are silent and they never complain. This assumption is appropriate for transportation service, and some network design. 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 from the viewpoint of the managers; the algorithms minimizes 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 spanning tree with the minimum number of vacant seats is $\mathcal{NP}$-complete. We also propose efficient approximation algorithms to construct a spanning tree with the minimum number of vacant seats.

**Keywords:** Airline problem, approximation algorithm, efficient algorithm, small-world network.

## 1 Introduction

Since early works by Watts & Strogatz [11] and Barabási & Albert [2], 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 [3, 10]. 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., [7]). 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." As an airline network, a small-world network has a good property for customers; in many cases, passengers can fly to most countries within a few transits. However, from the viewpoint of managers, the fact that airlines make a small-world network is lacking in efficiency. The fact implies that they are 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 the airline companies. Another reason seems that such an online ad hoc strategy, "connect the new airport to the closest airline hub," is easy decision and, anyway, of benefit to passengers.

In this paper, we consider the design problem of an airline network from the viewpoint of the managers. We suppose that there are independent cities (with no a priori connections), and passengers are silent; they never complain even if they have to transit many times. We also assume that all cities are given at first; that is, we consider offline algorithms. Our aim is to *design* the cheapest airline network that connects all cities and that can transport enough passengers. Remark that there are many works that aim to *assign* quantity over *given* network, e.g., network flow [5], distributed computing [8], the vehicle routing problem, and so on. We focus on the design problem of the network that we face before the assignment problems.

More precisely, we define *airline problem* over weighted nodes as follows:

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

**Input:** The set $V$ of nodes, and the positive integer weight function $w(v)$ for each $v$ in $V$.

**Output:** The set $E$ of edges $\{u, v\}$ in $V^2$, and the positive integer weight function $w(u, v)$ 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[1].

Intuitively, each node $v$ corresponds to a city, and the weight $w(v)$ gives the number of (potential) passengers in the city. We assume that they can be estimated since it can be proportional to the population of the city. 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. For example, suppose the following situation. An airplane transports at most $w(u, v)$ passengers along each edge $\{u, v\}$ in each time slot. Each passenger at the city $u$ randomly flies to the other city $v$ with probability proportional to $w(u, v)$. Under this random walk model, the populations of cities are not changed asymptotically, and each passenger can move to any city with at most $O(W|V|)$ transits on average, where $W = \sum_{\{u,v\} \in E} w(u, v)$ (we have the upper bound by replacing each edge $e$ of weight $w$ by $w$ multi edges of weight 1, and applying the classic theorem on the cover time of the simple random walk on $G$; see, e.g., [6, Theorem 6.8] for further details). Therefore the airline problem can be an idealized model for planning some real network problem like transportation service, 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 the random walk approach.

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 $|V|$ 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.

Since maintaining of an airline is costly, the problem to decrease the number of routes is important from the viewpoint of the manager. Hence our next aim is to construct a weighted spanning tree that has the minimum loss. However, unfortunately, the problem is intractable. More precisely, we show that the airline problem to construct a spanning tree (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 spanning tree $T$ of $V$ of approximation ratio 2 in $O(|V|)$ time and space. More precisely, the algorithm constructs a weighted spanning 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 spanning 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 the set of $n$ nodes, and $w$ be the positive integer weight function $w(v)$ for each $v \in V$. 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.*

We first have the following observation:

[1] 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.

**Input** : A set $V$ of $n$ nodes, positive integer weight function $w(v)$ for each $v \in V$.
**Output**: A set $E$ of $m$ edges $\{u, v\}$ such that $(V, E)$ is connected, and positive integer weight function $w(e)$ for each $e \in E$.

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**
  $w(v, v_{\max}) := w(v)$;
  $w(v_{\max}) := w(v_{\max}) - w(v)$;
**end**
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 1**: Star

**Input** : A set $V$ of $n$ nodes, positive integer weight function $w(v)$ for each $v \in V$.
**Output**: A set $E$ of $m$ edges $\{u, v\}$ such that $(V, E)$ is connected, and positive integer weight function $w(e)$ for each $e \in E$.

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**
  make a cycle $C := (v_1, v_2, \ldots, v_n, v_1)$ over $V$;
  **foreach** *odd* $i = 1, 3, 5, \ldots$ **do** $w(v_i, v_{i+1}) := \lceil w/2 \rceil$;
  **foreach** *even* $i = 2, 4, 6, \ldots$ **do** $w(v_i, v_{i+1}) := \lfloor w/2 \rfloor$;
**end**
**return** $(E := \{e \mid w(e) > 0\})$;

**Algorithm 2**: Uniform

**Observation 2** *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 three special cases. The first case is that $V$ contains one very heavy vertex. Let $v_{\max}$ be a heaviest vertex, i.e., $w(v_{\max}) \geq w(v)$ for all $v \in V$. Then we say *star condition* if we have

$$w(v_{\max}) - \sum_{v \in V \setminus \{v_{\max}\}} w(v) \geq 0.$$

Under the star condition, Algorithm 1 clearly 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 $|V| - 1$ edges.

The second case is that all vertices have the same weight, which is called *uniform condition*: $w(v) = w > 0$ for all $v \in V$.

**Lemma 3** *In the uniform condition, Algorithm 2 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. It is easy to see that the algorithm produces a connected network $E$ in $O(n)$ time and space, and $|E| \leq n$. Hence we show that $L(G)$ is the minimum loss. When $w = 1$, to make $G$ connected, a spanning tree $T$ is required. Thus $T$ gives the minimum loss $L(G) = \sum_{v \in V} L(v) = 2 \sum_{e \in E} w(e) - \sum_{v \in V} w(v) = 2(n-1) - n = n - 2$. When $w > 1$, we have $L(G) = 0$ or $L(G) = 1$, which is the minimum loss by Observation 2. ∎

We next turn to the third case. If $V$ contains at least two vertices of weight 1, we call that *many-ones condition*. To distinguish with the uniform case, we assume that $V$ also contains at least one vertex of weight greater than 1. In the case, 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 if and only if $|V_1| > 1$ 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. We have to check the condition in each addition of an edge. The purpose here is to reduce the number of vertices of weight 1 to one. Hence we

**Input** : A set $V$ of $n$ nodes, positive integer weight function $w(v)$ for each $v \in V$.
**Output**: Two sets $V'$ of $n'$ nodes, and $E$ of $n - n'$ edges such that $V$ contains at most one vertex of weight 1.

```
1  let v_max be a vertex such that w(v_max) ≥ w(v) for all v ∈ V;
2  let V_1 := {v | w(v) = 1} and V_2 := {v | w(v) ≥ 2} \ {v_max};
3  while |V_1| > 1 and |V_2| > 0 do
4  |  if w(v_max) ≥ Σ_{v∈V\{v_max}} w(v) then call Algorithm 1 as a subroutine and halt;
5  |  make an edge {v, v'} with w(v, v') = 1 for any vertices v ∈ V_1 and v' ∈ V_2;
6  |  remove v from V_1;
7  |  w(v') := w(v') − 1;
8  |  if w(v') = 1 then move v' from V_2 to V_1;
9  end
10 if |V_2| = 0 then
11 |  pick up any w(v_max) − 1 vertices from V_1, and join them to v_max by an edge of weight 1;
12 |  remove the w(v_max) − 1 vertices from V_1;
13 |  w(v_max) := 1;
14 |  call Algorithm 2 as a subroutine and halt;
15 end
16 return (E := {e | w(e) > 0} and V);
```

**Algorithm 3**: Many-ones

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 3 runs in $O(n)$ time and space by maintaining $V_1$ and $V_2$ by two queues. Algorithm 3 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.

Hence, hereafter, we assume that the input $V$ satisfies neither the star condition, the uniform condition, nor the many-ones condition. The details of the algorithm is given in Algorithm 4.

**Lemma 4** *Algorithm 4 outputs a connected network* $(V, E)$ *with* $|E| \leq n$.

Proof. In the while-loop, either (0) the algorithm calls Algorithms 1 or 2 and halts, (1) it sets $w(v, v')$ for some $v, v'$ and removes one of $v$ and $v'$, or (2) it joints all vertices in some vertex set $\hat{V}$ with $\left|\hat{V}\right|$ edges and halts. Thus the algorithm always halts after at most $|V|$ iterations of the while-loop. Now we assume that the algorithm removes all remaining vertices just before halting. Then it is easy to see that we have the following invariant; the number of removed vertices is equal to the number of added edges except calling Algorithm 1. Hence $|E|$ contains at most $n$ edges. It is easy to see that the resultant network is connected. ∎

**Lemma 5** *Algorithm 4 always outputs a network with the minimum loss* $L(G)$.

Proof. Without loss of generality, we assume that the input $V$ satisfies neither the star condition, the uniform condition, nor the many-ones condition. Let $V_1$ be the set of vertices of weight 1. Then we have an invariant that $|V_1| \leq 1$ throughout the algorithm.

Let $v_{\max}$ be the heaviest vertex chosen in step 2, and $V'$ be the set $V \setminus \{v_{\max}\}$. We have two cases; all vertices in $V'$ have the same weight $w$, and there are two vertices $v_i$ and $v_j$ in $V'$ with $w(v_i) < w(v_j)$.
(I) All vertices in $V'$ have the same weight $w$, which is handled in steps 8 to 24 in the algorithm. We first note that $V$ is not in neither the uniform case nor the star case. Thus, we have $w(v_{\max}) > w$ and $w(v_{\max}) < (n-1)w$, where $|V'| = |V| - 1 = n - 1$. 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 the resultant graph is disconnected. Hence, in the case, the algorithm matches $q - 1$ vertices of weight $w$ to $v_{\max}$, and then $v_{\max}$ has weight $w$. That is, in the case, we will have the uniform case in the next iteration. Through the process, the algorithm generates no loss. This case is handled in steps 16, and from 20 to 23. Hence if the uniform case will be handled properly, the algorithm generates no loss, which will be discussed later.

**Input** : A set $V$ of $n$ nodes, and positive integer weight function $w(v)$ for each $v \in V$.
**Output**: A set $E$ of $m$ edges $\{u, v\}$ such that $(V, E)$ is connected, and positive integer weight function $w(e)$ for each $e \in E$.

$n := |V|$, and $W := \sum_{v \in V} w(v)$;
find $v_{\max}$ such that $w(v_{\max}) \geq w(v)$ for any other $v \in V$;
let $V_1 := \{v \mid w(v) = 1\}$ (we have $|V_1| < 2$);
**while** true **do**
  **if** *$V$ satisfies the star condition* **then** call Algorithm 1 as a procedure and halt;
  **if** *all vertices in $V$ have the same weight* **then** call Algorithm 2 as a procedure and halt;
  **if** *all vertices in $V \setminus \{v_{\max}\}$ have the same weight $w$* **then**
    **if** $(n-2)w < w(v_{\max})$ **then** // we also have $w(v_{\max}) < (n-1)w$
      let $v_i$ and $v_j$ be any two vertices in $V \setminus \{v_{\max}\}$;
      $w(v_i, v_j) := w - \lfloor (w(v_{\max}) - (n-3)w)/2 \rfloor$;
      $w(v_i) := w(v_i) - \lfloor (w(v_{\max}) - (n-3)w)/2 \rfloor$;
      $w(v_j) := w(v_j) - \lceil (w(v_{\max}) - (n-3)w)/2 \rceil$ ; // we now have the star condition
    **else**
      $r := w(v_{\max}) \bmod w$;
      **if** $r = 0$ **then**
        let $V''$ consist of any $(w(v_{\max})/w) - 1$ vertices from $V \setminus \{v_{\max}\}$;
      **else**
        let $V''$ consist of any $\lfloor w(v_{\max})/w \rfloor$ vertices from $V \setminus \{v_{\max}\}$;
      **end**
      **foreach** *$v$ in $V''$* **do** $w(v, v_{\max}) = w$;
      $w(v_{\max}) := w(v_{\max}) - |V''|w$;
      remove all vertices in $V''$;
      **if** $w(v_{\max}) = 1$ **then** put $v_{\max}$ into $V_1$;
    **end**
  **else**
    **if** $V_1 \neq \emptyset$ **then**
      let $v_i$ be the vertex in $V_1$, and $v_j$ be any vertex in $V \setminus \{v_{\max}\}$ with $w(v_i) < w(v_j)$;
    **else**
      let $v_i$ and $v_j$ be any vertices in $V \setminus \{v_{\max}\}$ with $w(v_i) < w(v_j)$;
    **end**
    **if** $W - 2w(v_i) > 2w(v_{\max})$ **then**
      $w(v_i, v_j) := w(v_i)$;
      $w(v_j) := w(v_j) - w(v_i)$;
      remove $v_i$ from $V$;
      **if** $w(v_j) = 1$ **then** put $v_j$ into $V_1$;
    **else**
      $w(v_i, v_j) := \lceil (W - 2w(v_{\max}))/2 \rceil$;
      $w(v_i) := w(v_i) - \lceil (W - 2w(v_{\max}))/2 \rceil$;
      $w(v_j) := w(v_j) - \lfloor (W - 2w(v_{\max}))/2 \rfloor$ ; // we now have the star condition
    **end**
  **end**
  update $V$, $n$, $W$, $v_{\max}$ if necessary;
**end**

**Algorithm 4**: Network

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 $V'$, 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. This case is handled in steps 14, 18, from 20 to 23. However, the process fails when $w(v_{\max}) - qw$ is too small; 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 $V'$ into $V'_a$ of $q$ vertices and $V'_b$ of $n - q - 1$ vertices. All vertices in $V'_1$ are matched with $v_{\max}$, and removed from $V'$, and the loss will be generated when $\{v_{\max}\} \cup V'_b$ satisfies the star condition. Now $w(v_{\max}) < w$ after matching with $V'_a$, $V'_b$ contains the heaviest vertex of weight $w$. If $\left|V'_b\right| > 1$, it is impossible. Thus $\left|V'_b\right| = 1$. Hence this case occurs only if $(n-2)w < w(v_{\max}) < (n-1)w$, which is handled in steps from 9 to 12. In the case, we can have the optimal solution with the following assignments of weights; (0) pick up any two vertices $v_i$ and $v_j$ from $V'$, and (1) add the edge $\{v_i, v_j\}$ of weight $w - \lfloor (w(v_{\max}) - (n-2)w)/2 \rfloor$. Then we have the star condition, and we have $L(G) \leq 1$, which is the optimal.
(II) We next assume that the vertex set contains at least two vertices with different weights, which is handled in steps 26 to 39. 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 heavy enough, we add an edge $\{v_i, v_j\}$ with $w(v_i, v_j) = w(v_i)$ and remove $v_i$ (in steps 32 to 35). The exception is that $\sum_{v \in V'} w(v) < w(v_{\max})$ after removing $w(v_i)$. This is equivalent to $\sum_{v \in V'} w(v) = W - w(v_{\max}) - 2w(v_i) < w(v_{\max})$. Hence the case occurs when $W - 2w(v_i) < w(v_{\max})$. 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 $\sum_{v \in V'} w(v) = W - w(v_{\max}) > w(v_{\max})$, or consequently, $W > w(v_{\max})$. In the case, we can have the star condition by the edge $\{v_i, v_j\}$ with $w(v_i, v_j) = \left\lceil \frac{\sum_{v \in V'} w(v) - w(v_{\max})}{2} \right\rceil = \left\lceil \frac{W - 2w(v_{\max})}{2} \right\rceil$, and then we have the optimal. This case is handled in steps 37 to 39.

Hence, in most cases, the algorithm achieves the optimal network. The last case is in the following case: First, the algorithm does not call Algorithm 2, and later, it calls Algorithm 2 and the subroutine outputs a spanning tree since all vertices have the weight 1 when the subroutine is called. However, this case is impossible since we have an invariant $|V_1| \leq 1$.

Thus, Algorithm 4 always outputs a network with $L(G) \leq 1$, which is optimal by Observation 2, if $V$ does not satisfy one of three special conditions. ∎

**Lemma 6** *Algorithm 4 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 $V$ 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. Now we assume that the algorithm performs the while-loop. 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 $(n-2)w < w(v_{\max}) < (n-1)w$, 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 Algorithm 2 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 $w$. Thus, in the case, the algorithm can determine which steps in the next iteration. The case $n \leq 2$ is impossible since this step is performed only if $w(v_{\max}) < (n-2)w$. Thus we have $n > 2$. In the case, the algorithm set $v_i := v_{\max}$, $v_j := v$ for any vertex in $V'$ with $w(v) = w$, and $v_{\max}$ is updated by another vertex in $V'$ with $w(v) = w$, and jump to the step 34 directly after updating $n$ and $W$ in $O(1)$ time. 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 doubly linked 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 $w(v_2) = w(v_3)$, $w(v_3) = w(v_4)$, or $w(v_4) = w(v_5)$, ... until it finds $w(v_k) \neq w(v_{k+1})$. Then setting $v_i := \min\{v_k, v_{k+1}\}$ and $v_j := \max\{v_k, v_{k+1}\}$, the algorithm can perform from the step 34. 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 with running 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 or not. However, in the time, the algorithm knows that $w(v_1) = w(v_2) = \cdots = w(v_{i-1})$. Hence, 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 3, 4, 5, and 6, we immediately have Theorem 1.

# 3 Minimum cost spanning tree

In this section, a feasible answer of the airline problem is restricted to a weighted spanning tree. We first prove that the problem for finding a minimum loss spanning tree of the airline problem is $\mathcal{NP}$-complete. Next, we show an approximation algorithm 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 the set $V$ of nodes, the positive integer weight function $w(v)$ for each $v$ in $V$, and an integer $k$. Then the decision problem is to determine whether 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 spanning tree.

**Theorem 7** *The decision airline problem for finding a spanning tree is $\mathcal{NP}$-complete.*

Proof. The problem is clearly in $\mathcal{NP}$. We reduce 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 $a \in A$; |
| **Question:** Determine if there is a subset $A' \subset A$ such that $\sum_{a \in A'} w'(a) = \sum_{a \in A \setminus A'} w'(a)$. |

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 $w(u) = w(v) = \frac{W}{2} + 1$. It is easy to see that the reduction can be done in polynomial time and space. We show that the set $A$ can be partitioned into two subsets of the same weight if and only if $V$ has a spanning tree that has loss 0. 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 $W + 2$, which is larger than $W = \sum_{a \in A} w(a)$. Thus we have $L(G) > 0$ in the case.

First we assume that $A$ has a partition $A'$ and $A''$ such that $A' \cup A'' = A$, $A' \cap A'' = \emptyset$, and $\sum_{a \in A'} w(a) = \sum_{a \in A''} w(a) = W/2$. We show that $V$ has a weighted spanning tree that has loss 0. We define the weight function $w$ as follows; $w\{u, v\} = 1$, $w\{a, u\} = w(a)$ for all $a \in A'$, and $w\{a, v\} = w(a)$ for all $a \in A''$. By assumption and construction, it is easy to see that the set $E$ of positive weighted edges is the spanning tree with no loss.

Next we assume that $V$ has a weighted spanning tree $T$ with loss 0, and show that $A$ can be partitioned into $A'$ and $A''$ of the same weight. By the above observation, the edge $\{u, v\}$ has a positive weight. We then partition the set $A$ into $A'$ and $A''$ as follows; $A'$ consists of vertices $a \in A$ of odd distance from $u$, and $A''$ consists of vertices $a \in A$ of even distance from $u$. Since $T$ is a tree, or, a connected bipartite graph, $A'$ and $A''$ satisfy $A' \cap A'' = \emptyset$, $A' \cup A'' = A$, and two sets $A'$ and $A''$ are independent sets. Moreover, since $T$ has no loss, $\sum_{a \in A'} w(a) = \sum_{a \in A''} w(a) = W/2$ (we note that $\sum_{e \in A' \times A''} w(e) = w(u, v) - 1$). Thus $A'$ and $A''$ 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 spanning tree of minimum loss for the airline problem, which completes the proof. ∎

## 3.2 Approximation algorithms for finding a spanning tree of minimum loss

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.

### 3.2.1 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 $n$th edge when (1) it is in the uniform case, or (2) the edge $\{v_i, v_j\}$ is produced in step 10 or step 37 by Algorithm 4. 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 $2\lfloor (w(v_{\max}) - (n-3)w)/2 \rfloor \leq w(v_{\max}) - (n-3)w < w(v_{\max})$. In the latter case, $L(G)$ increases by $2\lceil (W - w(v_{\max}))/2 \rceil < w(v_{\max})$.

From above analysis, we immediately have the following theorem:

**Theorem 8** *The modified algorithm always outputs a spanning tree $T = (V, E)$ in $O(n)$ time and space. Then $L(T) \leq w(v_{\max})$.*

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 9** *Let $E$ be the set produced by the tree algorithm, and $E_{opt}$ be an optimal solution (with the minimum loss) of the airline problem. Let $T := (V, E)$ and $G := (V, E')$. Then, $\sum_{e \in E} w(e) < 2 \sum_{e \in E_{opt}} w(e)$.*

### 3.2.2 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, [9, Chapter 8][2]. Hence, using the FPTAS algorithm, we can compute a partition $X$ and $Y$ of $V$ with $\frac{\left|\sum_{v \in X} w(v) - \sum_{v \in Y} w(v)\right| - \left|\sum_{v \in X^*} w(v) - \sum_{v \in Y^*} w(v)\right|}{\sum_{v \in V} w(v)} < \epsilon$ for any positive constant $\epsilon$ in polynomial time of $|V|$ and $\epsilon$, where $X^*$ and $Y^*$ are an optimal partition of $V$.

In this section, we show a polynomial time algorithm that constructs a weighted spanning tree which is a feasible solution for the airline problem 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 the weighted spanning tree that is an optimal solution for the airline problem. On the other hand, if $V$ contains many vertices of weight 1, we can reduce them by Algorithm 3. 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 spanning tree $T$.

We can make a weighted spanning tree from the partition for the airline problem that achieves the same performance by the FPTAS.

**Theorem 10** *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 weighted spanning tree, and $L(T) \leq \max\{\delta, 2\}$. 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 $X = \{x_0 = v_0, x_1, x_2, \ldots, \}$ and $Y = \{y_1, y_2, \ldots, \}$. (We note that $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ are ordered in arbitrary way.) The first phase is given in Algorithm 5; it starts from a path $\{x_0, y_0\}$, and extend it as

[2] In [9, Chapter 8], the idea is applied to the knapsack problem. However it is easy to apply it to the set partition problem.

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

**Algorithm 5**: Make a caterpillar

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, we have $i > 0$ and one of $w(x_i)$ and $w(y_j)$ is updated, and the other one is not updated. Hence $w > w(v_0)$.

Now, we turn to the second phase. We now renumber the vertices as $X = \{x_0, x_1, x_2, \ldots\}$ and $Y = \{y_0, y_1, y_2, \ldots\}$ 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 also 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 $\{y_1, y_2, \ldots\}$.

If the algorithm finds $w(y_j)$ with $w(x_0) \neq w(y_j)$, it makes an edge $\{x_0, y_j\}$ with $w(x_0, y_j) = \min\{w(x_0), w(y_j)\} = w(x_0)$, remove $x_0$, and update the weight of $y_j$. 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 omits the 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 makes an edge $\{x_i, y_0\}$ with $w(x_i, y_0) = \min\{w(x_i), w(y_0)\} = w(y_0)$, and remove $y_0$, and update the weight of $x_i$. 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 $\delta$.

Now, we have the last case: $w(x) = w(y) = w$ for all $x \in X$ and $y \in Y$. Let we renumber $X = \{x_1, x_2, \ldots, x_k\}$ and $Y = \{y_1, y_2, \ldots, y_{k'}\}$. By the process above, every part of a tree has exactly one node in $X \cup Y$. Thus we have a weighted spanning tree $T$ of $V$ 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 $\delta$ 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 spanning weighted tree $T$ with $L(T) \leq \max\{2, \delta\}$. By similar implementation using queue of the vertices of the same weight in the proof of Lemma 6, the algorithm runs in $O(n)$ time and $O(n)$

space. ∎

# 4 Concluding remarks

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.

# Acknowledgment

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. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[3] A.L. Barabasi. *Linked: The New Science of Networks*. Perseus Books Group, 2002.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[5] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover, 2001.

[6] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge, 1995.

[7] M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[8] D. Peleg. *Distributed Computing: A Locally-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[9] V.V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[10] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 2004.

[11] D. J. Watts and D. H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 393:440–442, 1998.