アルゴリズム 73－4
（2000．5．19）

無向ネットワークにおける流量要求を満たす施設配置問題

新 浩治† 岩田 覚‡ 牧野 和久† 藤重 悟†
†大阪大学大学院 基礎工学研究科システム人間系専攻 システム科学分野
‡東京大学大学院 工学系研究科 計数工学専攻

概要

この論文では，無向ネットワークにおいて，各点 v に対して $d(v)$ の流量を送ることが可能な，最小費用の点集合 S を発見する問題を扱う．この問題は一般的には NP-困難であるが，田村らは，各点の費用が一定である特殊な場合の問題を解く貪欲解法を提案している．我々は線形計画双対性を用いて，この貪欲アルゴリズムにおける正当性のより単純な証明を与え，計算時間を $O(n^2M)$ から $O(nM)$ へと改良する．ここで n は ネットワークの点の数であり，M は n の点と m の枝を持つネットワークにおける最大フローの計算時間を定義する．我々はまた，一定の要求容量と任意の費用を持つ特殊な場合の問題を解く $O(n(m+n\log n))$ のアルゴリズムを提案する．

Locating Sources to Meet Flow Demands in Undirected Networks

Kouji Arata† Satoru Iwata‡ Kazuhisa Makino† Satoru Fujishige†
†Division of Systems Science, Graduate School of
Engineering Science, Osaka University,
‡Department of Mathematical Engineering and Information Physics,
Graduate School of Engineering, University of Tokyo

Abstract

This paper deals with the problem of finding a minimum-cost vertex subset S in an undirected network such that for each vertex v we can send $d(v)$ units of flow from S to v. Although this problem is NP-hard in general, Tamura et al. have presented a greedy algorithm for solving the special case with uniform costs on the vertices. We give a simpler proof on the validity of the greedy algorithm using linear programming duality and improve the running time bound from $O(n^2M)$ to $O(nM)$, where n is the number of vertices in the network and M denotes the time for max-flow computation in the network with n vertices and m edges. We also present an $O(n(m+n\log n))$ time algorithm for the special case with uniform demands and arbitrary costs.

1. Introduction

Let $\mathcal{N}=(G,u,d,c)$ be an undirected network on the underlying graph $G=(V,E)$ with the vertex set V and the edge set E. Let $n=|V|$ and $m=|E|$. It is endowed with a capacity function $u: E\to\mathbf{R}_+$, a demand function $d: V\to\mathbf{R}_+$, and a cost function $c: V\to\mathbf{R}_+$, where $\mathbf{R}_+$ denotes the set of nonnegative reals. This paper addresses the problem of finding a minimum-cost vertex subset $S\subseteq V$ such that for each $v\in V$ we can send $d(v)$ units of flow from S to v.

For a pair of disjoint subsets $X,Y\subseteq V$, we denote by $\lambda(X,Y)$ the maximum flow value between X and Y in $\mathcal{N}$. We simply write $\lambda(v,Y)$ and $\lambda(X,w)$ for $v,w\in V$ instead of $\lambda(\{v\},Y)$ and $\lambda(X,\{w\})$, respectively. For convenience, we assign $\lambda(X,Y)=+\infty$ if $X\cap Y\neq\emptyset$. Then the problem is formulated as follows.

$$\begin{array}{ll}\text{Minimize} & \sum_{v\in S} c(v)\\ \text{subject to} & S\subseteq V,\end{array}\qquad(1.1)$$

$$\lambda(S, v) \geq d(v) \quad (v \in V).$$

We call this problem SOURCE LOCATION. We say that a vertex set $S \subseteq V$ *covers* a vertex v if $\lambda(S, v) \geq d(v)$. Namely, SOURCE LOCATION asks for a minimum-cost subset $S \subseteq V$ that covers all the vertices in V.

A special case of this problem with a constant cost function was introduced by Tamura et al. [11]. They called it *plural cover problem*. They first considered the case in which both d and c are constant and described an algorithm that runs in $O(n^2M)$ time [10], where M denotes the time complexity for computing an s-t maximum flow in a given network $\mathcal{N}$ [1, 3, 4]. Later Tamura et al. [11] showed that a simple greedy algorithm solves problem SOURCE LOCATION in $O(n^2M)$ time even if the demand function d is arbitrary while the cost function c is still constant. Ito and Yokoyama [5] described another algorithm to improve the time complexity to $O(npM)$, where p is the number of distinct values of $d(v)$ $(v \in V)$, i.e., $p = |\{d(v) \mid v \in V\}|$.

In this paper, we analyze the greedy algorithm of Tamura et al. [11] to give a simpler proof based on the linear programming duality. We then improve the greedy algorithm to run in $O(nM)$ time.

As for the case in which the demand function d is constant, we give an $O(n(m+n\log n))$ time algorithm. The algorithm makes use of maximum adjacency (MA) ordering (see Section 4 for MA ordering). The MA ordering has been used by Nagamochi and Ibaraki for solving the problems of minimum cut [6] and of edge-connectivity augmentation [7].

Finally, we show that SOURCE LOCATION is in general NP-hard. We show this by reducing the knapsack problem to SOURCE LOCATION. Hence, it remains open to prove the NP-hardness in the strong sense or to devise a pseudo-polynomial time algorithm.

We summarize the time complexity of SOURCE LOCATION in Table 1, where bold letters indicate the results obtained in this paper.

The rest of the paper is organized as follows. Section 2 formulates SOURCE LOCATION as an integer programming problem, Section 3 considers SOURCE LOCATION when the cost function c is constant, and Section 4 discusses the case in which the demand function d is constant. In Section 5, we show that SOURCE LOCATION is in general NP-hard.

2. Integer Programming Formulation

In this section, we formulate SOURCE LOCATION as an integer programming problem with an exponential number of constraints.

A *cut* is a proper nonempty subset of V. For a cut X, let ΔX denote the set of edges that cross X, i.e., $\Delta X = \{e \mid e = (v, w) \in E, v \in X, w \in V - X\}$, and $\kappa(X)$ its capacity, i.e.,

$$\kappa(X) = \sum_{e \in \Delta X} u(e).$$

Table 1: Summary of the results on SOURCE LOCATION

	c: constant		c: arbitrary
d: constant	$O(n^2M)$	Tamura et al. (1992) [10]	$\boldsymbol{O(n(m + n\log n))}$
	$O(nM)$	Ito and Yokoyama (1997) [5]	
	$\boldsymbol{O(n(m + n\log n))}$		
d: arbitrary	$O(n^2M)$	Tamura et al. (1998) [11]	**NP-hard**
	$O(npM)$	Ito and Yokoyama (1997) [5]	
	$\boldsymbol{O(nM)}$		

M: the time complexity for computing a maximum s-t flow in $\mathcal{N}$.
p: the number of distinct values of $d(v)$ $(v \in V)$.

If $X = \{v\}$, we write $\kappa(v)$ instead of $\kappa(\{v\})$. For a disjoint pair of vertex subsets $X, Y \subseteq V$, we denote $\kappa(X,Y) = \sum_{e \in \Delta X \cap \Delta Y} u(e)$. For $v \in V$, we simply write $\kappa(X,v)$ instead of $\kappa(X,\{v\})$.

We also denote by $d(W)$ the maximum demand in W, i.e.,

$$d(W) = \max\{d(v) \mid v \in W\}.$$

We say that a vertex $v \in W$ *attains the maximum demand in* W if $d(v) = d(W)$. A cut W is called *deficient* if $\kappa(W) < d(W)$. If a cut W is deficient and no other subset $X \subset W$ is deficient, W is called a *minimal deficient set.*

Lemma 2.1 ([11]): *Let* $\mathcal{N} = (G = (V,E), u, d, c)$ *be an undirected network. Then* $S \subseteq V$ *covers all vertices in* V *if and only if* $S \cap W \neq \emptyset$ *holds for every minimal deficient set* W.

Let $\mathcal{W} = \{W_1, W_2, \cdots, W_l\}$ be the family of all the minimal deficient sets and let $V = \{v_1, v_2, \cdots, v_n\}$. Define an $l \times n$ matrix $A = (A_{ij})$ by $A_{ij} = 1$ if $v_j \in W_i$ and $A_{ij} = 0$ otherwise. From Lemma 2.1, SOURCE LOCATION can be written as the following 0-1 integer programming problem:

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} A_{ij} x_j \geq 1 \quad (i = 1, 2, \cdots, l)$$

$$x_j \in \{0,1\} \quad (j = 1, 2, \cdots, n),$$

where $c_j = c(v_j)$ $(j = 1, 2, \cdots, n)$, and $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ is the characteristic vector of a subset of V.

3. The Uniform Cost Case

3.1. A Greedy Algorithm

In this section, we consider SOURCE LOCATION with a constant cost function. Tamura et al. [11] proposed the following greedy algorithm to solve SOURCE LOCATION.

Algorithm GREEDY

Step 0: Arrange the vertices $v_1, v_2, \cdots, v_n$ in V such that $d(v_1) \leq d(v_2) \leq \cdots \leq d(v_n)$.

Step 1: Initialize j:=1 and S:=V.

Step 2: If $S - \{v_j\}$ covers all vertices in V, then S:=$S - \{v_j\}$.

Step 3: If $j = n$ then output S and halt. Otherwise, j:=$j+1$ and go to Step 2. □

In order to show the correctness of algorithm GREEDY, we consider the linear programming relaxation of (2.1):

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{3.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} A_{ij} x_j \geq 1 \quad (i = 1, 2, \cdots, l),$$

$$x_j \geq 0 \quad (j = 1, 2, \cdots, n),$$

and its dual:

$$\text{Maximize} \quad \sum_{i=1}^{l} y_i \tag{3.2}$$

$$\text{subject to} \quad \sum_{i=1}^{l} A_{ij} y_i \leq c_j \ (j = 1, 2, \cdots, n),$$

$$y_i \geq 0 \quad (i = 1, 2, \cdots, l).$$

Recall that $c_j = 1$ $(j = 1, 2, \cdots, n)$ is assumed in this section.

We also replace Steps 1 and 2 in algorithm GREEDY as follows.

Step 1′: Initialize $j := 1$, $S := V$ and $y_i := 0$ for $i = 1, 2, \cdots, l$.

Step 2′: (2-1) If $S - \{v_j\}$ covers all vertices in V, then S:=$S - \{v_j\}$.

(2-2) Otherwise, choose a $W_i \in \mathcal{W}$ with $W_i \cap S = \{v_j\}$, and $y_i := 1$.

Note that Step 1′ (initialization of y) in the revised version might take exponential time (since $|\mathcal{W}|$ might be exponential). However, this causes no trouble since we are now interested in the validity of the algorithm. Obviously, the algorithm always keeps a feasible solution S (i.e., S covers all vertices in V).

Let $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ be the primal and dual variables obtained at the end of the revised greedy

algorithm. Note that $\boldsymbol{x}^*$ is the characteristic vector of the output S of the algorithm.

The algorithm does not delete v_j from S if and only if it updates y_i as $y_i := 1$ for some i with $W_i \cap S = \{v_j\}$. Hence, at the termination, we have

$$\sum_{j=1}^{n} x_j^* = \sum_{i=1}^{l} y_i^*. \tag{3.3}$$

Therefore, $\boldsymbol{x}^*$ is a 0-1 solution satisfying (3.1) and (3.3). By the weak duality of linear programming problems (3.1) and (3.2), we only need to prove the feasibility of $\boldsymbol{y}^*$ in (3.2) to show the correctness of the algorithm GREEDY. The feasibility of $\boldsymbol{y}^*$ will be proved by Lemmas 3.1 and 3.2 given below.

Recall that the cut capacity function κ satisfies

$$\kappa(X) + \kappa(Y) \geq \kappa(X - Y) + \kappa(Y - X) \quad (X, Y \subseteq V). \tag{3.4}$$

A set function satisfying (3.4) is called *posimodular* in [8].

Lemma 3.1 ([11]): *Let $\mathcal{N} = (G = (V, E), u, d, c)$ be an undirected flow network. Let W_1 and W_2 be minimal deficient sets in $\mathcal{N}$, and for each $i = 1, 2$, let $v_i \in W_i$ be a vertex that attain the maximum demand in W_i. If $W_1 \cap W_2 \neq \emptyset$, then we have $v_1 \in W_1 \cap W_2$ or $v_2 \in W_1 \cap W_2$.*

Proof. Suppose, to the contrary, that both $v_1 \in W_1 - W_2$ and $v_2 \in W_2 - W_1$ hold. Since W_1 and W_2 are deficient sets, $d(v_1) > \kappa(W_1)$ and $d(v_2) > \kappa(W_2)$ hold. It follows from (3.4) that

$$\begin{aligned} d(v_1) + d(v_2) &> \kappa(W_1) + \kappa(W_2) \\ &\geq \kappa(W_1 - W_2) + \kappa(W_2 - W_1). \end{aligned}$$

This means that $d(v_1) > \kappa(W_1 - W_2)$ or $d(v_2) > \kappa(W_2 - W_1)$ holds. Since we have $v_1 \in W_1 - W_2$ and $v_2 \in W_2 - W_1$ by the assumption, it follows that $W_1 - W_2$ or $W_2 - W_1$ is deficient, which contradicts the minimality of W_1 or W_2. □

Arrange the columns of A in such a way that $d(v_1) \leq d(v_2) \leq \cdots \leq d(v_n)$. For each index i with $1 \leq i \leq l$, let $k(i)$ denote the maximum number k with $v_k \in W_i$. Then Lemma 3.1 implies that the matrix A does not contain

$$\begin{array}{c} \\ i_1 \\ i_2 \end{array} \begin{array}{c} \begin{array}{ccc} j & k(i_1) & k(i_2) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{array} \tag{3.5}$$

as a submatrix.

Lemma 3.2: *The dual variable $\boldsymbol{y}^*$ obtained by the revised greedy algorithm is feasible to* (3.2).

Proof. Suppose, to the contrary, that $\boldsymbol{y}^*$ is infeasible. There is a pair of distinct rows, i_1, i_2 and a column j such that $y_{i_1}^* = y_{i_2}^* = 1$ and $A_{i_1 j} = A_{i_2 j} = 1$. Let j_0 be the largest such number j, where we assume that the columns of A is already arranged in such a way that $d(v_1) \leq d(v_2) \leq \cdots \leq d(v_n)$. Then we have $k(i_1) \neq k(i_2)$ since otherwise $y_{i_1}^*$ and $y_{i_2}^*$ must be updated in the same iteration in Step 2′, a contradiction. Note that $j_0 = k(i_1)$ implies $y_{i_2}^* = 0$ by the greedy algorithm. Hence we have $j_0 < k(i_1)$. Similarly, we also have $j_0 < k(i_2)$. Furthermore, we have $A_{i_1 k(i_2)} = 0$ due to the definition of j_0. Similarly, we have $A_{i_2 k(i_1)} = 0$. This implies that A contains submatrix (3.5) forbidden by Lemma 3.1. □

We have thus shown the following.

Theorem 3.3: *If the cost function c is constant, then algorithm* GREEDY *produces an optimal solution of* SOURCE LOCATION.

3.2. An Efficient Implementation

We now analyze the time complexity of algorithm GREEDY. Steps 0, 1 and 3 are clearly executed in $O(n \log n)$, $O(1)$ and $O(n)$ time, respectively. As for Step 2, Tamura et al. [11] checked if $S - \{v_j\}$ covers all vertices in V by computing $\lambda(S - \{v_j\}, v_i)$ (i.e., a max flow from $S - \{v_j\}$ to v_i) for all v_i. Clearly, this requires $O(nM)$ time, where M is the time complexity for computing a maximum s-t flow in the network $\mathcal{N}$ [1, 3, 4]. Since Step 2 is iterated n times, the required time is $O(n^2 M)$ in total [11].

However, the following lemma implies that Step 2 can be replaced by

Step 2″: If $S-\{v_j\}$ covers v_j, then S:=$S-\{v_j\}$.

Lemma 3.4: *If $S-\{v_j\}$ covers v_j in Step 2 of algorithm* GREEDY, *then $S-\{v_j\}$ covers v_i for all $i \leq j$.*

Proof. Assume that some v_i with $i < j$ is not covered by $S-\{v_j\}$. Then there exists a cut X with $v_i \in X$, $V-X \supseteq S-\{v_j\}$, and $\kappa(X) < d(v_i)$. Then, $S \cap X \subseteq \{v_j\}$ clearly holds. Moreover, we have $S \cap X = \{v_j\}$ since otherwise S does not cover v_i, which contradicts the property that GREEDY always keeps a feasible set S. Hence, X separates v_j and $S-\{v_j\}$. Since $\kappa(X) < d(v_i) \leq d(v_j)$, it follows that $S-\{v_j\}$ does not cover v_j, a contradiction. □

Thus we have improved the time complexity.

Theorem 3.5: *If the cost function c is constant, then problem* SOURCE LOCATION *can be solved in $O(nM)$ time.*

4. The Uniform Demand Case

In this section, we consider SOURCE LOCATION with a constant demand function d. We assume that $d(v) = g$ (a fixed positive real) holds for all $v \in V$. We show that it can be solved in $O(n(m+n\log n))$ time without maximum flow computation. A key tool of the algorithm is the maximum adjacency (MA) ordering.

An ordering $v_1, v_2, \cdots, v_n$ of all vertices in V is called a *maximum adjacency (MA) ordering* if it satisfies

$$\kappa(\{v_1, v_2, \cdots, v_i\}, v_{i+1}) \geq \kappa(\{v_1, v_2, \cdots, v_i\}, v_j) \quad \text{for } 1 \leq i < j \leq n.$$

The MA ordering plays a crucial role in this section through the following lemma.

Lemma 4.1 ([6, 9]): *Let $G = (V, E)$ be an undirected graph with a nonnegative capacity function u. Then, the following statements hold.*

(i) *An MA ordering $v_1, v_2, \cdots, v_n$ can be computed in $O(m+n\log n)$ time.*

(ii) *The last two vertices v_{n-1} and v_n for every MA ordering in G satisfy*

$$\lambda(v_{n-1}, v_n) = \kappa(v_n). \tag{4.1}$$

□

We mention here that we can choose the first vertex v_1 arbitrarily.

Let us note that, if the demand function d is constant, minimal deficient sets are pairwise disjoint by the posi-modularity 3.4 of κ, i.e.,

$$W_1 \cap W_2 = \emptyset$$

holds for every pair of W_1 and W_2 in $\mathcal{W}$. Therefore, in order to solve SOURCE LOCATION, we try to find all minimal deficient sets $W \in \mathcal{W}$ and construct a minimum-cost source set $S \subseteq V$ by choosing from each $W \in \mathcal{W}$ a vertex $v \in W$ with the minimum cost $c(v)$ among W.

Since any source set S must contain $v \in V$ such that $\kappa(v) < g$, we initialize S as $S := \{v \in V \mid \kappa(v) < g\}$. To make use of MA orderings, we attach a new vertex $s\,(\notin V)$ to a given network $\mathcal{N}$ and, for each vertex $v \in S$, add the edge (s, v) with the capacity $u(s, v) = g$. By this modification of $\mathcal{N}$, every vertices $v \in V$ satisfies $\kappa(v) \geq g$, i.e., either $\kappa(v) \geq g$ holds in the original network or $v \in S$ (i.e., the (modified) network $\mathcal{N}$ contains the edge (s, v) with $u(s, v) = g$). We then apply to the network $\mathcal{N}$ an MA ordering $v_0\,(= s), v_1, \cdots, v_{n-1}, v_n$ starting from s. By Lemma 4.1, we have

$$\lambda(v_{n-1}, v_n) = \kappa(v_n) \geq g.$$

Namely, every cut X that separates v_{n-1} and v_n satisfies $\kappa(X) \geq g$. This means that every minimal deficient set $W \in \mathcal{W}$ (in the original network) that separates v_{n-1} and v_n forms $W = \{v_{n-1}\}$ or $\{v_n\}$, since by the modification of $\mathcal{N}$, such a W must contain a vertex $v \in V$ such that $\kappa(v) < g$ in the original network, and hence we have $|W| = 1$. Since we already checked whether a cut X of the type $X = \{v\}$ $(v \in V)$ is deficient, we do not have to consider the cut X separating v_{n-1} and v_n. We thus merge the vertices v_{n-1} and v_n into a single vertex $\hat{v}$, and check if $\hat{v}$ satisfies $\kappa(\hat{v}) \geq g$. Since $\kappa(\hat{v}) < g$ implies that $W = \{v_{n-1}, v_n\}$ is a

minimal deficient set in the original network, if $\kappa(\hat{v}) < g$, we update the network $\mathcal{N}$ by adding edge $(s, \hat{v})$ with the capacity $u(s, \hat{v}) = g$, and update S by adding v_{n-1} if $c(v_{n-1}) < c(v_n)$; otherwise, v_n.

Now we have $\kappa(\hat{v}) \geq g$ for all vertices except for s in the resulting network $\mathcal{N}$. By repeating the above argument for $\mathcal{N}$ (i.e., we apply MA ordering $v_0 (= s), v_1, \cdots, v_{h-1}, v_h$ to $\mathcal{N}$, merge the last two vertices v_{h-1} and v_h, and so on), we can compute a minimum-cost source set S. Formally it can be written as follows.

Algorithm CONTRACT

Input: A network $\mathcal{N} = (G = (V, E), u, d, c)$, where $d(v) = g$ for all v.

Output: A minimum-cost vertex set $S \subseteq V$ which covers all vertices in V.

Step 0: Initialize $S := \emptyset$, $V' := V \cup \{s\}$, $E' := E$, and $\alpha(v) := v$ for all $v \in V$.

Step 1: For each vertex $v \in V$ such that $\kappa(v) < g$, put $E' := E' \cup \{(s, v)\}$, $u(s, v) := g$, and $S := S \cup \{\alpha(v)\}$.

Step 2:

(2-I) Compute an MA ordering $v_0 (= s), v_1, \cdots, v_{h-1}, v_h$ starting from s in $G' = (V', E')$.

(2-II) Merge the last two vertices v_{h-1} and v_h in G' into a single vertex $\hat{v}$. Denote the resulting graph by G' again.

(2-III) If $c(\alpha(v_{h-1})) < c(\alpha(v_h))$, then $\alpha(\hat{v}) := \alpha(v_{h-1})$; Otherwise, $\alpha(\hat{v}) := \alpha(v_h)$.

(2-IV) If $\kappa(\hat{v}) < g$ in the current G', then update $E' := E' \cup \{(s, \hat{v})\}$, $u(s, \hat{v}) := g$, and $S := S \cup \{\alpha(\hat{v})\}$.

Step 3: If $|V'| \leq 2$ or E' contains the edges (s, v) for all $v \in V' - \{s\}$, then output S and halt. Otherwise go to Step 2. □

Note that the algorithm prepares $\alpha(\cdot)$ for computing from each $W \in \mathcal{W}$ a vertex $v \in W$ with the minimum cost $c(v)$ among W. Formally, $\alpha(v)$ $(v \in V')$ stores the vertex v^* in the original network $\mathcal{N}$ having the minimum cost $c(v^*)$ among P_v, where P_v is the set of all vertices v in V which are merged to v.

Theorem 4.2: *Problem* SOURCE LOCATION *can be solved in* $O(n(m + n \log n))$ *time if the demand function* d *is constant.*

Proof. Since the above discussion shows the correctness of algorithm CONTRACT, we only consider its time complexity. Clearly Steps 0, 1 and 3 take $O(n)$ time. Step 2 can be executed in $O(n(m+n \log n))$ time since it has $n-1$ iterations and each iteration takes $O(m + n \log n)$ time from Lemma 4.1. Therefore, in total, it requires $O(n(m + n \log n))$ time. □

5. NP-hardness of General Case

In this section, we show the NP-hardness of SOURCE LOCATION with non-constant cost and demand functions.

Theorem 5.1: *Problem* SOURCE LOCATION *is NP-hard, even if the undirected graph* $G = (V, E)$ *is a star, i.e.,* $E = \{(v, w) \mid w \in V \backslash \{v\}\}$ *for some* $v \in V$.

Proof. We transform Problem KNAPSACK to this problem, where KNAPSACK is known to be NP-hard [2].

Problem KNAPSACK

Input: A finite set $Z = \{z_1, z_2, \cdots z_n\}$ associated with a size function $\sigma : Z \to \mathbf{Z}_+$ and a value function $\omega : Z \to \mathbf{Z}_+$, and positive integer $b (\leq \sum_{z_i \in Z} \sigma(z_i))$. where $\mathbf{Z}_+$ denotes the set of all nonnegative integers.

Output: A subset $X \subseteq Z$ that is an optimal solution of

$$\begin{aligned} &\text{Maximize} && \sum_{z \in X} \omega(z) \\ &\text{subject to} && \sum_{z \in X} \sigma(z) \leq b, \quad (5.1) \\ & && X \subseteq Z. \end{aligned}$$

It is easy to see that KNAPSACK is polynomially equivalent to the problem of computing

a subset $Y \subseteq Z$ that solves

$$\begin{aligned} \text{Minimize} \quad & \sum_{z \in Y} \omega(z) \\ \text{subject to} \quad & \sum_{z \in Y} \sigma(z) \geq \sum_{z \in Z} \sigma(z) - b, \quad (5.2) \\ & Y \subseteq Z, \end{aligned}$$

by identifying Y with $Z - X$. Therefore, in the following we consider (5.2) instead of (5.1).

For this problem instance, we consider an undirected network $\mathcal{N} = (G = (V, E), u, d, c)$ with $V = Z \cup \{z_0\}$, $E = \{(z_0, z_i) \mid z_i \in Z\}$, $u(z_0, z_i) = \sigma(z_i)$ for $i = 1, 2, \cdots, n$ and

$$d(z_i) = \begin{cases} \sum_{z_i \in Z} \sigma(z_i) - b & \text{if } i = 0 \\ 0 & \text{otherwise,} \end{cases}$$

$$c(z_i) = \begin{cases} \sum_{z_i \in Z} \omega(z_i) + 1 & \text{if } i = 0 \\ \omega(z_i) & \text{otherwise.} \end{cases}$$

Note that $d(z_i) = 0$ for all $z_i \in Z$. Therefore $S \subseteq V$ covers all vertices in V if and only if it covers z_0, i.e.,

$$\lambda(S, z_0) \geq d(z_0) = \sum_{z_i \in Z} \sigma(z_i) - b. \quad (5.3)$$

Moreover, since $\{z_0\}$ and Z covers z_0, and since $c(z_0) > \sum_{z_i \in Z} c(z_i)$, an optimal S is contained in Z. This implies that

$$\lambda(S, z_0) = \sum_{z_i \in S} u(z_0, z_i) = \sum_{z_i \in S} \sigma(z_i), \quad (5.4)$$

and hence (5.3) is equivalent to the constraint in (5.2).

Since $c(z_i) = \omega(z_i)$ for all $z_i \in Z$, $S \subseteq Z$ is an optimal solution for the instance of problem (5.2) if and only if it is optimal for the corresponding instance for SOURCE LOCATION. □

6. Conclusion

In this paper, we have analyzed the greedy algorithm of Tamura et al. [11] for SOURCE LOCATION with a constant cost function and given a simpler proof based on the linear programming duality. We have also improved the greedy algorithm to run in $O(nM)$ time. Moreover, we have given an $O(n(m + n \log n))$ time algorithm for SOURCE LOCATION with a constant demand function. Finally, we have shown that SOURCE LOCATION is in general NP-hard by reducing KNAPSACK to SOURCE LOCATION.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin: *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, (1993).

[2] M. R. Garey and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freedman, New York, (1979).

[3] A. V. Goldberg and S. Rao: Beyond the flow decomposition barrier, *Journal of the ACM*, **45** (1998), 783–797.

[4] A. V. Goldberg and S. Rao: Flows in undirected unit capacity networks, *SIAM J. Discrete Mathematics*, **12** (1999), 1–5.

[5] H. Ito and M. Yokoyama: Minimum size flow-sink-set location problem with various flow-demands of nodes, *IEICE Trans. Fundamentals*, to appear.

[6] H. Nagamochi and T. Ibaraki: Computing edge-connectivity of multigraphs and capacitated graphs, *SIAM J. Discrete Mathematics*, **5** (1992), 54–66.

[7] H. Nagamochi and T. Ibaraki: Deterministic $\tilde{O}(nm)$ time edge-splitting in undirected graphs, *J. Combinatorial Optimization*, **1** (1997), 5–46.

[8] H. Nagamochi and T. Ibaraki: A note on minimizing submodular functions, *Information Processing Letters*, **67** (1998), 169–178.

[9] M. Stoer, and F. Wagner: A simple min cut algorithm, *Journal of the ACM*, **44**, (1997) 585–591.

[10] H. Tamura, M. Sengoku, S. Shinoda, and T. Abe: Some covering problems in location theory on flow networks, *IEICE Trans. Fundamentals*, **E75-A** (1992), 678–683.

[11] H. Tamura, H. Sugawara, M. Sengoku, and S. Shinoda: Plural cover problem on undirected flow networks, *IEICE Trans. Fundamentals*, **J81-A** (1998), 863–869 (in Japanese).