アルゴリズム 81－5
（2001．11．27）

# 木構造の動的ネットワークにおける施設配置問題

間々田 聡子　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科学分野
560-8531 大阪府豊中市待兼山 1-3
E-mail: mamada@sflab.sys.es.osaka-u.ac.jp
{makino, fujishig}@sys.es.osaka-u.ac.jp

**概要**：本論文では，木構造ネットワークにおける動的フロー問題と施設配置問題を統合した問題を考察する．これは，木構造ネットワークと各点に供給量が与えられているとき，その全ての供給量を最速に輸送するような出口 $v$ を求める問題である．この問題は木構造ネットワークにおける 1-センター問題の動的フロー版と見なすことができる．我々は，この施設配置問題に対する $O(n^2)$ 時間アルゴリズムを与える．ただし，$n$ はネットワークの点数である．

**キーワード**：動的フロー，施設配置問題，木構造ネットワーク

# Optimal Sink Location Problem for Dynamic Flows in a Tree Network

SATOKO MAMADA　KAZUHISA MAKINO　SATORU FUJISHIGE

Division of Systems Science, Graduate School of Engineering Science,
Osaka University, Toyonaka, Osaka, 560-8531 Japan
E-mail: mamada@sflab.sys.es.osaka-u.ac.jp,
{makino, fujishig}@sys.es.osaka-u.ac.jp

**Abstract** : In this paper we consider a compound problem of dynamic flows and sink location in a tree network. Given a dynamic flow network of tree structure with initial supplies at vertices, the problem is to find a vertex $v$ as a sink in the network such that we can send all the initial suplies to $v$ as quick as possible. This problem can be regarded as a dynamic flow version of the 1-centerin problem in a tree network. We present an $O(n^2)$ time algorithm for the sink location problem, where $n$ is the number of vertices in the network.

**Keywords** : dynamic flow, location problem, tree network.

# 1. Introduction

Dynamic network flow problems have been considered in the literature (see, e.g., [4, 2]) and the multiple-source and multiple-sink dynamic network flow problem has recently been solved in strongly polynomial time [5].

In this paper we consider a compound problem of dynamic flows and sink location in a tree network. The problem is described as follows (see Section 2 for details). Suppose that we are given a dynamic flow network $T$ of tree structure with initial supplies at vertices. For a vertex $v$ in $T$ to be treated as a sink, denote by $C(v)$ the minimum time in which all the initial suplies at vertices can be sent to sink $v$. We call $C(v)$ the completion time for sink $v$. Then the problem is to find a vertex $v$ such that the completion time $C(v)$ is minimum among all vertices in $T$. This problem can be regarded as the 1-center problem of dynamic flow version in a tree network [7] and as one of the location problems based on flows [1, 6, 8, 9].

The rest of the present paper is organized as follows. Some definitions and preliminary results are given in Section 2. We describe an algorithm for the sink location problem in Section 3 and show the validity of the algorithm. Section 4 discusses the time complexity of our algorithm. Finally, we give some concluding remarks in Section 5.

# 2. Definitions and Preliminaries

Let $T = (V, E)$ be a tree with a vertex set $V$ and an edge set $E$. Let $\mathcal{N} = (T, c, \tau, b)$ be a dynamic flow network with the underlying undirected graph being the tree $T$, where $c : E \to \mathbf{R}_+$ is a capacity function, $\tau : E \to \mathbf{Z}_+$ a transit time function, and $b : V \to \mathbf{R}_+$ a supply function. Here, $\mathbf{R}_+$ denotes the set of nonnegative reals and $\mathbf{Z}_+$ the set of nonnegative integers. Suppose that we are given a sink $t$ in $T$. Then, $T$ is regarded as an in-tree with root $t$, i.e., each edge of $T$ is oriented toward the root $t$. Such an oriented tree with root $t$ is denoted by $\vec{T}(t) = (V, \vec{E}, t)$. Each oriented edge in $\vec{E}$ is denoted by the ordered pair of its end vertices and is called an arc. For each edge $\{u, v\} \in E$ we write $c(u, v)$ and $\tau(u, v)$ instead of $c(\{u, v\})$ and $\tau(\{u, v\})$. For any arc $(u, v) \in \vec{E}$, any $\theta \in \{0, 1, \cdots, \tau(u, v)\}$, and any $k \in \mathbf{Z}_+$, we denote by $f_k((u, v), \theta)$ the value of a flow along the $\theta$-th portion of arc $(u, v)$ at time $k$, where we define $f_0((u, v), \theta) = 0$ for $1 \leq \theta \leq \tau(u, v)$. We call $f_k((u, v), \theta)$ ($(u, v) \in \vec{E}$, $\theta \in \{0, 1, \cdots, \tau(u, v)\}$, $k \in \mathbf{Z}_+$) *a dynamic flow* in $\vec{T}(t)$ with a sink $t$ if it satisfies the following (a)$\sim$(c).

(a) (Capacity constraints): For any arc $(u, v) \in \vec{E}$, any $\theta \in \{0, 1, \cdots, \tau(u, v)\}$, and any $k \in \mathbf{Z}_+$,

$$0 \leq f_k((u, v), \theta) \leq c(u, v). \tag{2.1}$$

(b) (Flow transition): For any arc $(u, v) \in \vec{E}$, any $\theta \in \{0, 1, \cdots, \tau(u, v) - 1\}$, and any $k \in \mathbf{Z}_+$,

$$f_k((u, v), \theta) = f_{k+1}((u, v), \theta + 1). \tag{2.2}$$

(c) (Flow conservation): For any vertex $v \in V \setminus \{t\}$ and any $k \in \mathbf{Z}_+$,

$$\sum_{u:(u,v)\in\vec{E}} f_k((u, v), \tau(u, v)) + h_{k-1}(v) = f_k((v, v'), 0) + h_k(v), \tag{2.3}$$

$$h_{-1}(v) = b(v),\ h_k(v) \geq 0 \quad (k = 0, 1, \cdots) \tag{2.4}$$

where $(v, v')$ is the unique arc leaving $v$ toward root $t$ in $\vec{T}(t)$ (i.e., $v'$ is the parent of $v$ in $\vec{T}(t)$) and $h_k(v)$ is the value of holdover at vertex $v$ at time $k$, i.e., we allow intermediate storage or holding inventory at each vertex.

This paper addresses the problem of finding a sink $t \in V$ such that we can send given initial supplies $b(v)$ ($v \in V \setminus \{t\}$) to sink $t$

as quick as possible. This problem can be regarded as a dynamic version of the 1-center location problem (for a tree) [7]. In particular, if $c(v,w) = +\infty$ (sufficiently large) for all edges $\{v,w\} \in E$, our problem represents the 1-center location problem [7].

To describe our algorithm we need further definitions. For each vertex $v \in V$ we keep two tables, *Arriving Table* $A_v$ and *Sending Table* $S_v$. Arriving Table $A_v$ represents the total value of the flows reaching the vertex $v$ as a function of time $k$, i.e.,

$$\sum_{u:(u,v)\in\vec{E}} f_k((u,v),\tau(u,v)) + \eta_k(v), \qquad (2.5)$$

where $\eta_0(v) = b(v)$ and $\eta_k(v) = 0$ $(k = 1,2,\cdots)$. Also, Sending Table $S_v$ represents the value of the flow leaving the vertex $v$ to its parent $v'$ (toward the sink $t$) as a function of time $k$, i.e.,

$$f_k((v,v'),0). \qquad (2.6)$$

Let us consider a function $g : \mathbf{Z}_+ \to \mathbf{R}_+$ in time $k \in \mathbf{Z}_+$, which will be some Arriving (Sending) Table in the sequel. If $g(k) \neq g(k-1)$, we call $k$ a *jump time* of $g$. Here, we assume $g(k) = 0$ for $k < 0$. We denote by *Pattern I at time* $k$ a time interval from $k-2$ to $k+1$ for some $k$ satisfying $g(k-2) < g(k-1) > g(k) = g(k+1)$, and by *Pattern II at time* $k$ a time interval from $k-\alpha$ to $k+1$ for some $\alpha \geq 3$ and $k$ satisfying $g(k-\alpha) \neq g(k-\alpha+1) = \cdots = g(k-1) > g(k) = g(k+1)$. See Figure 1, where (a) and (b) in Figure 1 show Patterns I and II at time $k$, respectively. A double circle denotes $g(l)$ at a jump time $l$. In order to represent the function $g$, we only keep all jump times $k_1, k_2, \cdots$ of $g$ and their values $g(k_1), g(k_2), \cdots$. Arriving Table $A_v$ $(v \in V)$ and Sending Table $S_v$ $(v \in V \setminus \{t\})$ are given in such a way.

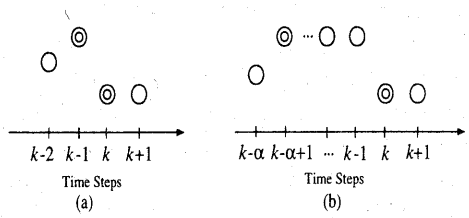

Figure 1: (a) Pattern I and (b) Pattern II

For each time $k$, we denote by $A_v(k)$ (resp., $S_v(k)$) the value, at time $k$, of the function represented by Arriving (resp., Sending) Table $A_v$ (resp., $S_v$) at vertex $v$.

# 3. Algorithm

In this section, we give an algorithm to solve the sink location problem described in the previous section.

Our algorithm consists of two phases, Phases I and II. Given a network $\mathcal{N}$ and an arbitrary vertex $t \in V$ as a candidate sink, Phase I computes the completion time $C(t)$ for a sink $t$ (i.e., the quickest time $C(t)$ in which all the initial supplies $b(v)$ $(v \in V)$ can be sent to sink $t$). In Phase II, we repeatedly pick up a new sink $\hat{t}$ that is adjacent to the previous sink $t$ and compute the completion time $C(\hat{t})$ for the new sink $\hat{t}$. An efficient way of choosing a new sink and a stopping rule are given below.

**Algorithm Phase I**

**Input:** A tree network $\mathcal{N} = (T = (V,E), c, \tau, b)$ and a sink $t \in V$.

**Output:** The completion time $C(t)$.

**Step 0:** Put $T' \leftarrow \vec{T}(t)$.

**Step 1:** If $T'$ consists of $t$ alone, then go to Step 3.
For each leaf $v(\neq t)$ of $T$, construct Sending Table $S_v$ from Arriving Table $A_v$.

**Step 2:** For each non-leaf vertex $w$ whose children are all leaves, construct Arriving Table $A_w$ based on Sending Tables of its children.
Remove all the leaves $v(\neq t)$ from $T'$ and denote the resultant tree by $T'$ again.
Go to Step 1.

**Step 3:** Compute the completion time $C(t)$ from $A_t$.
Return $C(t)$ and halt. □

Here, it should be noted that Arriving Table $A_v$ for a leaf $v$ of the original $\vec{T}(t)$ represents the initial supply given at $v$, i.e., $A_v(0) = b(v)$ and $A_v(k) = 0$ for $k \neq 0$. It should also be noted that if the value of the left-hand side of (2.3) for a vertex $v(\neq t)$ is at least the capacity $c(v, v')$, then we put $f_k((v, v'), 0) = c(v, v')$ to attain the (quickest) completion time $C(t)$. Also, note that $f_k((v, v'), 0) < c(v, v')$ only if $h_k(v) = 0$. This gives a procedure for constructing Sending Table $S_v$ by using Arriving Table $A_v$. We can easily see that algorithm Phase I correctly computes the completion time $C(t)$ as well as $A_v$ $(v \in V)$ and $S_v$ $(v \in V \setminus \{t\})$.

**Algorithm Phase II**

**Input:** A tree network $\mathcal{N} = (T = (V, E), c, \tau, b)$ and a sink $t \in V$ with Arriving Tables $A_v$ $(v \in V)$, Sending Tables $S_v$ $(v \in V \setminus \{t\})$, and the completion time $C(t)$.

**Output:** An optimal sink $t^*$ that has the minimum completion time $C(t^*)$ among all vertices of $T$.

**Step 0:** Find a child $v$ of root $t$ from which the last flow reaches $t$ at time $C(t)$. Put $\hat{t} \leftarrow v$ and consider $\hat{t}$ as a new sink.

**Step 1:** Compute the completion time $C(\hat{t})$ and the corresponding tables as follows.

**(1-I)** Compute new Arriving Table $A_t$ by subtracting the flow, sent from $\hat{t}$ according to Sending Table $S_{\hat{t}}$, from that expressed by old Arriving Table for $t$.

**(1-II)** Compute Sending Table $S_t$ from $t$ to $\hat{t}$ based on new $A_t$.

**(1-III)** From this new table $S_t$ and $S_v$ for the other children $v$ of $\hat{t}$, compute new Arriving Table $A_{\hat{t}}$ and the completion time $C(\hat{t})$.

**Step 2:**

**(2-I)** If $C(t) < C(\hat{t})$, then return $t^* = t$ and halt.

**(2-II)** If $C(t) \geq C(\hat{t})$ and the last flow reaches sink $\hat{t}$ from $t$, then return $t^* = \hat{t}$ and halt.

**(2-III)** Otherwise, put $t \leftarrow \hat{t}$ and go to Step 0.

□

We shall show the validity of algorithm Phase II below.

For adjacent vertices $u$ and $v$ in $T$, deleting edge $\{u, v\}$ from $T$ yields two connected components. Denote by $T(u, v)$ the component containing $u$ and by $T(v, u)$ the one containing $v$.

**Lemma 3.1**: *Suppose that for a sink $t$ the last flow reaches sink $t$ from $\hat{t}$. Then, we have $C(v) \geq C(t)$ for any vertex $v$ in $T(t, \hat{t})$.*

**Proof.** For any new sink $v$ in $T(t, \hat{t})$, it requires at least $C(t)$ time for sending all supplies $b(w)$'s with $w$ in $T(\hat{t}, t)$ to $v$, due to the assumption. □

**Lemma 3.2**: *If $C(t) < C(\hat{t})$ holds in Step 2 of Phase II, then $t$ is an optimal sink.*

**Proof.** From Lemma 3.1, we have $C(v) \geq C(t)$ for any $v$ in $T(t, \hat{t})$. Moreover, by the definition of $\hat{t}$, the last flow reaching sink $\hat{t}$ comes from $t$. Hence, again from Lemma 3.1, we have $C(v) \geq C(\hat{t})$ for any $v$ in $T(\hat{t}, t)$. This completes the proof. □

**Lemma 3.3**: *If $C(t) \geq C(\hat{t})$ and the last flow reaches sink $\hat{t}$ from $t$ in Step 2 of Phase II, then $\hat{t}$ is an optimal sink.*

**Proof.** Similar to the proof of Lemma 3.2. □

It follows from Lemmas 3.2 and 3.3 that the algorithm correctly outputs an optimal sink $t^*$ if it halts. Note that, while Step (2-III) is repeated, the sequence of new sinks forms a path in $T$. Hence Step (2-III) is repeated at most $n - 1$ times, where $n$ is the cardinality of $V$. This completes the correctness of the algorithm.

## 4. The Time Complexity

In this section, we show the time complexity of our algorithm.

First, we show the following lemmas to get an upper bound for the number of jump times in Arriving and Sending Tables computed by Phase I.

For a vertex $v_i$ in $V$ (resp., $V \setminus \{t\}$), we denote by $d_i$ (resp., $d_i'$) the number of jump times and by $p_i$ (resp., $p_i'$) the total number of Pattern I's and Pattern II's in Arriving Table $A_{v_i}$ (resp., $S_{v_i}$). We simply call Pattern I or Pattern II *Pattern* in the following.

**Lemma 4.1**: *For any vertex $v_i \in V \setminus \{t\}$, we have*

$$d_i' + p_i' \leq d_i + p_i, \tag{4.1}$$

*i.e., when constructing Sending Table $S_{v_i}$ from Arriving Table $A_{v_i}$, the total number of jump times and Patterns does not increase.*

**Proof.** First note that for each Pattern I at time $k$ in $S_{v_i}$, either it comes from Pattern I at time $k$ in $A_{v_i}$ or $k+1$ is a jump time in $A_{v_i}$. In the latter case, $k+1$ is a jump time in $A_{v_i}$ within the time interval $I = [k+1, k+1]$ that does not appear in $S_{v_i}$. Similarly, for each Pattern II at time $k$ in $S_{v_i}$, either it comes from Pattern II at time $k$ in $A_{v_i}$ or at least one jump time in $A_{v_i}$ within the time interval $I = [k-\alpha+2, k+1]$ does not appear in $S_{v_i}$.

Let us then consider a new jump time in $S_{v_i}$. Suppose that $k$ is a jump time in $S_{v_i}$ but not in $A_{v_i}$. Then we have

$$A_{v_i}(k-1) = A_{v_i}(k),\ S_{v_i}(k-1) \neq S_{v_i}(k). \tag{4.2}$$

Since we send out as much amount of flow as possible from $v_i$ to its parent $w$, if $A_{v_i}(k-1) = A_{v_i}(k) \geq c(v_i, w)$, we must have $S_{v_i}(k-1) = S_{v_i}(k) = c(v_i, w)$. Hence,

$$c(v_i, w) > A_{v_i}(k-1) = A_{v_i}(k). \tag{4.3}$$

It follows from (4.2) and (4.3) that

$$c(v_i, w) = S_{v_i}(k-2) \geq S_{v_i}(k-1) > S_{v_i}(k), \tag{4.4}$$

since we must have $h_{k-2}(v_i) > 0$. We separately consider the following two cases.

*Case 1:* $S_{v_i}(k-2) > S_{v_i}(k-1)$. If $k-2$ is a jump time in $S_{v_i}$, then we have $S_{v_i}(k-3) < S_{v_i}(k-2) > S_{v_i}(k-1) > S_{v_i}(k)$. It follows that $A_{v_i}(k-3) < c(v_i, w) < A_{v_i}(k-2) > A_{v_i}(k-1) = A_{v_i}(k)$. This is Pattern I at time $k-1$, that does not appear in $S_{v_i}$. Therefore, within the time interval $I_1 = [k-1, k]$, $S_{v_i}$ contains 2 jump times and no Pattern, while $A_{v_i}$ contains one jump time and one Pattern, where we say that a Pattern at time $k^*$ is within the time interval $I$ if $k^*$ belongs to $I$. This implies that the total number of jump times and Patterns in $S_{v_i}$ within the time interval $I_1$ is equal to that in $A_{v_i}$.

On the other hand, if $k-2$ is not a jump time in $S_{v_i}$, then for some $l \geq 4$ we have

$$S_{v_i}(k-l) < S_{v_i}(k-l+1) = \cdots = S_{v_i}(k-3) = S_{v_i}(k-2) > S_{v_i}(k-1) > S_{v_i}(k). \tag{4.5}$$

Since $h_{k-2}(v_i) > 0$, within the time interval $I_2 = [k-l+2, k]$ $A_{v_i}$ contains either (i) at least two jump times or (ii) one jump time and one Pattern that is given as

$$A_{v_i}(k-l) < c(v_i, w) < A_{v_i}(k-l+1) = \cdots = A_{v_i}(k-l') > c(v_i, w) > A_{v_i}(k-l'+1) = \cdots = A_{v_i}(k-1) = A_{v_i}(k)$$

for some $l'$ with $2 \leq l' \leq l-1$, while $S_{v_i}$ contains two jump times and no Pattern. Therefore, the total number of jump times and Patterns in $S_{v_i}$ within the time interval $I_2$ is at most that in $A_{v_i}$.

*Case 2:* $S_{v_i}(k-2) = S_{v_i}(k-1)$. Then for some $l \geq 3$, we have

$$S_{v_i}(k-l) < S_{v_i}(k-l+1) = \cdots = S_{v_i}(k-2) = S_{v_i}(k-1) > S_{v_i}(k). \tag{4.6}$$

Within the time interval $I_3 = [k-l+2, k]$ there exists at least one jump time in $A_{v_i}$ that does not appear in $S_{v_i}$. Therefore, if $S_{v_i}(k) \neq S_{v_i}(k+1)$, the total number of jump times and Patterns in $S_{v_i}$ within the time interval $I_3$ is at most that in $A_{v_i}$. Finally, if $S_{v_i}(k) =$

$S_{v_i}(k+1)$, then $S_{v_i}$ contains Pattern II at time $k$. In this case, within the time interval $I_4 = [k-l+2, k]$ $A_{v_i}$ contains either (i) at least two jump times or (ii) one jump time and one Pattern. Thus, the total number of jump times and Patterns in $S_{v_i}$ within the time interval $I_4$ is at most that in $A_{v_i}$.

Since we have only to consider pairwise disjoint intervals $I$'s, this completes the proof. □

Let us consider a subtree $T_i$ rooted at $v_i$ which is depicted in Figure 2. Let $n_i$ be the

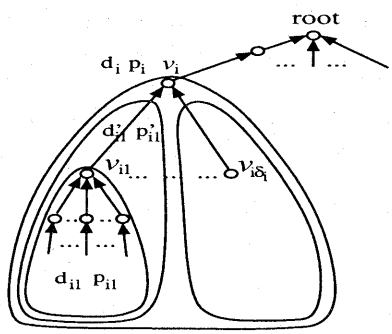


Figure 2: subtree $T_i$

number of the vertices in the subtree $T_i$. For each child $v_{ij}$ $(j = 1, 2, \cdots, \delta_i)$ of $v_i$ let $d_{ij}$ (resp., $d'_{ij}$) denote the number of the jump times in $A_{v_{ij}}$ (resp., $S_{v_{ij}}$), $p_{ij}$ (resp., $p'_{ij}$) the total number of Pattern I's and Pattern II's in $A_{v_{ij}}$ (resp., $S_{v_{ij}}$), and $n_{ij}$ the number of the vertices in the subtree $T_{v_{ij}}$ rooted at $v_{ij}$. Here, note that $n_i = n_{i1} + \cdots + n_{i\delta_i} + 1$.

**Lemma 4.2**: *For any non-leaf $v_i \in V$, we have*

$$d_i + p_i \leq \sum_{j=1}^{j=\delta_i} (d'_{ij} + p'_{ij}) + 3. \quad (4.7)$$

**Proof.** Note that $A_{v_i}$ is constructed from the initial supply $b(v_i)$ and Sending Table $S_{v_{ij}}$ $(j = 1, 2, \cdots, \delta_i)$. We assume that the initial supply $b(v_i)$ can be sent from a new vertex $v_{i0}$ to $v_i$ through a new arc $(v_{i0}, v_i)$ with $c(v_{i0}, v_i) \geq b(v_i)$ and $\tau(v_{i0}, v_i) = 0$. In other words, we assume that $b(v_i) = 0$ and $A_{v_i}$ is constructed from Sending Table $S_{v_{ij}}$ $(j = 0, 1, \cdots, \delta_i)$. Note that $S_{v_{i0}}$ has two jump times and one Pattern I.

If $k$ is a jump time in $A_{v_i}$, then $k - \tau(v_{ij'}, v_i)$ must be a jump time in Sending Table $S_{v_{ij'}}$ for some $j' \in \{0, 1, \cdots, \delta_i\}$, since $A_{v_i}$ is constructed from Sending Tables $S_{v_{ij}}$ $(j = 0, 1, \cdots, \delta_i)$. However, Pattern I and/or Pattern II may newly appear in $A_{v_i}$.

We first consider Pattern I at time $k$ in $A_{v_i}$. Then, there must be a child $w$ of $v_i$ such that $S_w(k-1-\tau(w,v_i)) > S_w(k-\tau(w,v_i))$. If $S_w(k-\tau(w,v_i)) \neq S_w(k+1-\tau(w,v_i))$, then $k+1-\tau(w,v_i)$ is a jump time in $S_w$, but $k+1$ is not a jump time in $A_{v_i}$. Therefore, in this case, we can regard that the Pattern I comes from the jump time $k+1-\tau(w,v_i)$ in $S_w$. On the other hand, if

$$S_w(k-\tau(w,v_i)) = S_w(k+1-\tau(w,v_i)), \quad (4.8)$$

we have the following two cases (see Figure 3 (a) and (b)). In the following, we simply write $\hat{\tau}$ instead of $\tau(w, v_i)$.

(a) $S_w(k-2-\hat{\tau}) \geq S_w(k-1-\hat{\tau}) > S_w(k-\hat{\tau}) = S_w(k+1-\hat{\tau})$:
In this case there is another child $z$ such that $S_z(k-2-\tau(z,v_i)) < S_z(k-1-\tau(z,v_i))$ since $A_{v_i}(k-2) < A_{v_i}(k-1)$. Then $k-1-\tau(z,v_i)$ is a jump time in $S_z$. If $k-1-\hat{\tau}$ is also a jump time in $S_w$, then these two jump times become a single jump time $k-1$ in $A_{v_i}$. Since in this case we can regard that the Pattern I comes from the jump time $k-1-\tau(w,v_i)$ in $S_w$ (where we regard that jump time $k-1$ in $A_{v_i}$ comes from jump time $k-1-\tau(z,v_i)$ in $S_z$), we consider the case when $S_w(k-2-\hat{\tau}) = S_w(k-1-\hat{\tau})$. Then we have Pattern II at time $k-\hat{\tau}$ in $S_w$.

(b) $S_w(k-2-\hat{\tau}) < S_w(k-1-\hat{\tau}) > S_w(k-\hat{\tau}) = S_w(k+1-\hat{\tau})$:
This is Pattern I at $k-\hat{\tau}$ in $S_w$.

We next consider Pattern II at time $k$ in $A_{v_i}$. Similarly as for Pattern I, there must be a child $w$ of $v_i$ such that $S_w(k-1-\tau(w,v_i)) >$

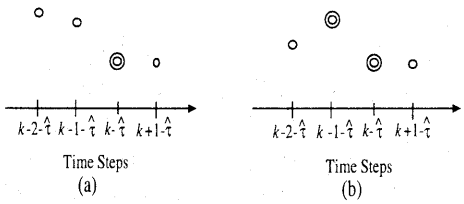


Figure 3: Cases (a) and (b)

$S_w(k-\tau(w,v_i))$. If $S_w(k-\tau(w,v_i)) \neq S_w(k+1-\tau(w,v_i))$, then we can regard that the Pattern II comes from the jump time $k+1-\tau(w,v_i)$ in $S_w$. Hence we suppose that

$$S_w(k-\tau(w,v_i)) = S_w(k+1-\tau(w,v_i)). \quad (4.9)$$

Now we have the following two cases (see Figure 3 (a) and (b)). Here, we also simply write $\hat{\tau}$ instead of $\tau(w,v_i)$.

(a) $S_w(k-2-\hat{\tau}) \geq S_w(k-1-\hat{\tau}) > S_w(k-\hat{\tau}) = S_w(k+1-\hat{\tau})$:
If $S_w(k-2-\hat{\tau}) = S_w(k-1-\hat{\tau})$, this is Pattern II. Otherwise there is another child $z$ such that $S_z(k-2-\tau(z,v_i)) < S_z(k-1-\tau(z,v_i))$ since $A_{v_i}(k-2) = A_{v_i}(k-1)$. Then we can regard that the Pattern II comes from the jump time $k-1-\hat{\tau}$ in $S_w$.

(b) $S_w(k-2-\hat{\tau}) < S_w(k-1-\hat{\tau}) > S_w(k-\hat{\tau}) = S_w(k+1-\hat{\tau})$:
Then this is Pattern I at time $k-\hat{\tau}$.

Since any two distinct patterns that newly appear in $A_{v_i}$ come from two distinct jump times, we have

$$\begin{aligned} d_i + p_i &\leq \sum_{j=0}^{j=\delta_i} (d'_{ij} + p'_{ij}) \\ &= \sum_{j=1}^{j=\delta_i} (d'_{ij} + p'_{ij}) + 3. \end{aligned}$$

□

We are now ready to derive an upper bound for the number of jump times. More precisely, we show by induction that for any vertex $v_i$ in $T$,

$$d_i + p_i \leq 3n_i. \quad (4.10)$$

This holds for any leaf $v_i$ in $T$, since $d_i = 2$ and $p_i = 1$. For a non-leaf $v_i$ in $T$ (having children $v_{ij}$, $j = 1,2,\cdots,\delta_i$), we assume that

$$d_{ij} + p_{ij} \leq 3n_{ij}. \quad (4.11)$$

Then it follows from (4.1), (4.7) and (4.11) that

$$\begin{aligned} d_i + p_i &\leq \sum_{j=1}^{j=\delta_i} (d'_{ij} + p'_{ij}) + 3 \quad \text{(by (4.7))} \\ &\leq \sum_{j=1}^{j=\delta_i} (d_{ij} + p_{ij}) + 3 \quad \text{(by (4.1))} \\ &\leq 3n_i \\ &\quad \text{(by (4.11) and } n_i = \sum_{j=1}^{j=\delta_i} n_{ij} + 1). \end{aligned}$$

Therefore, we have the following lemma.

**Lemma 4.3**: *For any Arriving Tables $A_v$ and Senting Table $S_v$, the number of its jump times is at most $3n$.*

**Lemma 4.4**: *Algorithm Phase I requires $O(n^2)$ time.*

**Proof.** For each $v_i \in V \setminus \{t\}$, Sending table $S_{v_i}$ can be constructed from Arriving Table $A_{v_i}$ in $O(d_i)\,(= O(n))$ time. For each non-leaf $v_i \in V$, Arriving Table $A_{v_i}$ (see Figure 2) can be obtained from Sending Tables $S_{v_{ij}}$ $(i = 1,2,\cdots\delta_i)$ and the initial supply $b(v_i)$ as follows. We first construct $\delta_i$ intermediate tables by shifting $S_{v_{ij}}$ $(i = 1,2,\cdots\delta_i)$ right by $\tau(\{v_{ij},v_i\})$, and then add them and $\eta_k(v_i)$ $(k = 0,1,\cdots)$ given in (2.5). Clearly, it can be done in $O(\sum_{1\leq j\leq\delta_i} d_{ij})(= O(n))$ time. Since we have $n$ vertices in $T$, Algorithm Phase I terminates in $O(n^2)$ time. □

**Lemma 4.5**: *Algorithm Phase II requires $O(n^2)$ time.*

**Proof.** Clearly, Steps 0 $\sim$ 2 takes $O(n)$ time by Lemma 4.3. Note that the number of iterations between Step 0 and Step 2 is at most $n-1$, since the sequence of new sinks forms a path in $T$. Hence, Algorithm Phase II requires $O(n^2)$ time. □

By combining Lemmas 4.4 and 4.5, we have the following theorem.

**Theorem 4.6**: *The sink location problem can be solved in* $O(n^2)$ *time.*

## 5. Concluding Remarks

In this paper we have presented an $O(n^2)$ time algorithm for an optimal sink location problem for dynamic flows in a tree, which can be regarded as the 1-center problem of dynamic version. We have assumed that the network allows intermediate storage at vertices. However, optimal solutions remain the same if we do not allow intermediate storage. Moreover, we can deal with the sink location problem for continuous-time dynamic flows in the same way as developed in the present paper (cf. [3]). Our algorithm also works for dynamic tree networks with asymmetric capacities and transition times.

Finally, the sink location problem for dynamic flows can further be extended in many directions. Some of them are (1) to find a sink to which we can send a maximum value of flow from sources within given fixed time, (2) to consider the sink location problem on general (non-tree) dynamic networks, and (3) to consider a multiple-sink location problem. These are left for future research.

## Acknowledgement

This research is partially supported by the Grant-in-Aid for Creative Scientific Research of the Ministry of Education, Culture, Sports, Science and Technology.

## References

[1] K. Arata, S. Iwata, K. Makino and S. Fujishige: Locating sources to meet flow demands in undirected networks. *SWAT2000*, edited by M. M. Halldorsson, Bergen (Norway), Springer Lecture Notes in Computer Science **1851**, (2000) 300-313.

[2] J. E. Aronson: A survey of dynamic network flows. *Annals of Operations Research*, **20** (1989) 1-66.

[3] L. Fleischer and É. Tardos: Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, **23** (1998) 71-80.

[4] L. R. Ford, Jr. and D. R. Fulkerson: *Flows in Networks* (Princeton University Press, Princeton, NJ, 1962).

[5] B. Hoppe and É. Tardos: The quickest transshipment problem. *Mathematics of Operations Research*, **25** (2000) 36–62.

[6] H. Ito, H. Uehara and M. Yokoyama: A faster and flexible algorithm for a location problem on undirected flow networks. *IEICE Trans. Fundamentals*, **E83-A** (2000), 704–712.

[7] P. B. Mirchandani and R. L. Francis: *Discrete Location Theory* (John Wile & Sons, Inc., 1989).

[8] 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.

[9] 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).