國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

具時間限制的廣播中心問題
Broadcast Centers in Trees with Time Constraints

黃煜翔

Yu-Shiang Huang

指導教授：陳健輝博士

Advisor: Gen-Huey Chen, Ph.D.

指導教授：林清池博士

Advisor: Ching-Chi Lin, Ph.D.

中華民國 104 年 6 月

June, 2015

國立臺灣大學碩士學位論文
口試委員會審定書

具時間限制的廣播中心問題

Broadcast Centers in Trees with Time Constraints

本論文係黃煜翔君（學號 R01922090）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 104 年 6 月 23 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

陳健輝
（指導教授）

胡家正　　周信宏

林清池

系主任　趙坤茂

誌謝

感謝神明安排我做這個論文題目。

Acknowledgements

I’m glad to thank the god for assigning this problem.

摘要

本論文提出一 $O(n)$ 時間複雜度的演算法來解決樹狀結構上的廣播中心點問題，使廣播中心點的數量最少。廣播按照異質郵寄模型的規則進行，需在時間限制內完成。

關鍵字: 廣播中心點問題，時間限制，異質郵寄模型，樹狀結構，貪進法。

Abstract

In this thesis, we present a $O(n)$-time exact algorithm to find a broadcast strategy such that broadcasting can be completed within the time constraint and the number of centers is minimal. The given graph is a tree and broadcasting is under the heterogeneous postal model.

Keywords: broadcast center problem, time constraint, heterogeneous postal model, trees, greedy method.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Broadcasting is an important problem in our life; the main objective is finding efficient strategies to deliver the messages. In many distributed or point-to-point systems, like people and mobile devices in a crowded place, every node can only contact with nodes nearby. Broadcasting makes every node in the system reveived the message.

1.1 Broadcast Problem and Models

$Broadcasting$ is an information dissemination problem. In a graph network $G(V, E)$, there are at least one $broadcast\ center$, which has the message before broadcasting starts. During broadcasting, nodes have the message can set up calls, which copy the message to their neighbors. A call from u to v is made up of two phases. The first one is $setup\ phase$; it takes α time. During setting up, u cannot do other things. The other one is $transmission\ phase$; it takes $w(u, v)$ time. Node u can set up a connection to another node, while transmitting messages to node v. Under these conditions, the time and the distances may affect our broadcasting strategies.

The $broadcast\ centers\ with\ time\ constraints$ problem is defined as below. Given Given a graph $G(V, E)$ and a time constraint t, the broadcast centers with time constraints problem is to determine $OPT(G, t)$, the minimal number of centers required.

There are several models in broadcasting problem. The uniform telephone model is the first introduced model; it is also the most widely-studied model. Under the nonuniform telephone model, the setup time, α, from u to v, is the length of the edge (u, v). Under the heterogeneous postal model, the setup time, α, is a non-negative number, and the transmission time, $w(u, v)$, is the length of the edge (u, v). This comparison tells us both the uniform telephone model and the postal model are special cases of the heterogeneous postal model. In other words, once problems under the heterogeneous postal model are solved, the same problems under the uniform telephone model is also solved. Table 1.1 lists several models.

Table 1.1: Models

Model	α	$w(u, v)$
Heterogeneous Postal [1]	fixed	not fixed
Postal [2], [3]	fixed	1
Telephone, nonuniform [4]	not fixed	0
Telephone, uniform [5]	1	0

1.2 Main Results and Thesis Organization

We propose an $O(n)$-time deterministic algorithm to solve the broadcast centers with time constraints problem on trees under heterogeneous postal model.

Chapter 2 presents many notations and definitions we will use. Chapter 3 presents the algorithm and its time complexity. Chapter 4 shows the correctness of the algorithm. Chapter 5 presents execution of the algorithm with two examples.

Chapter 2

Preliminaries

We use a graph to represent a network. Before introducing the algorithm, we introduce several notations, definitions, and related problems.

2.1 Notations and Definitions

A graph is made up of nodes and edges. We use $G(V, E)$ to represent a graph, where V is the set of nodes and E is the set of edges. An edge is a curve connects two endpoints. We use (u, v) to represent an edge where u and v are its two endpoints; we denote $w(u, v)$ as its length. An another graph $G'(V', E')$ is said a subgraph of G iff $V' \subseteq V$ and $E' = \{(u, v) \in E | u, v \in V'\}$. A node u is a neighbor of v iff there is an edge (u, v) in G. We use $N(v)$ to represent the set of neighbors of v. The degree of a node v is the number of edges where v is an endpoint. In this thesis, without mentioned particularly, every graph is undirected and simple. Simple means there are no (v, v), edges such that its two endpoints are identical and there is at most one edge for every node pair.

A sequence of edges is a walk if this sequence has the form $((u_0, u_1), (u_1, u_2), \cdots, (u_{k-1}, u_k))$. If $u_0, u_1, \cdots, u_k$ are distinct, it is a path; if they are distinct except $u_0 = u_k$, it is a cycle. A graph is connected if for every couple of nodes u and v, there is a path from u to v. A tree is a connected graph without cycles. A tree T is a spanning tree of a graph G if T can be obtained by removing edges from G.

After the basic knowledges are introduced, we introduce several terms we will use in this thesis. Let (u, v) be an edge in T. Removing (u, v) from T leads to two trees, $T(u, v)$ and $T(v, u)$ where $T(u, v)$ contains u and $T(v, u)$ contains v. The predecessor of v, $pred(v)$, is a neighbor of v such that v receives the message from $pred(v)$ when broadcasting. We use $pred(v) = \emptyset$ to represent v has no predecessor. The successors of v, $SUCC(V)$, is an ordered subset of the neighbors of v. A vertex $u \in SUCC(v)$ if u receives the message from v when broadcasting.

The broadcast time of v, $b\_time(v, S)$, is the time required to broadcast a message from v to all nodes in $\bigcup_{x \in S} T(x, v)$. The default value of S is $SUCC(v)$; that is, $b\_time(v) = b\_time(v, SUCC(v))$.

$$b\_time(v) = \begin{cases} 0, & \text{if } SUCC(v) = \emptyset; \\ max\{i\alpha + w(v, u_i) + t(u_i) | 1 \le i \le k\}, & \\ & \text{if } SUCC(v) = (u_1, u_2, \cdots, u_k). \end{cases}$$

The successor candidates of v, $SUCC^*(v)$, is a subset of previously processed neighbors of v. A vertex $u \in SUCC^*(v)$, if $pred(u) = \emptyset$ and $\alpha + w(v, u) + b\_time(u) \le t$. The predecessor candidates of v, $PRED^*(v)$, is a subset of previously processed neighbors of v. A vertex $u \in PRED^*(v)$, if $u \notin SUCC(v)$ and $spare(u) + \alpha + w(v, u) \le t$.

The arrive time of v, $arrive(v)$, is the earliest time v knows the message. The earliest spare time of v, $spare(v)$, is the earliest time that v can set up a connection with a receiver $u \notin SUCC(v)$ under the constraint that the broadcasting from v to the subtrees rooted by nodes on $SUCC(v)$ can still be completed in t. The unused edges of T, $E^-(T)$, is a subset of $E(T)$. An edge $(u, v) \in E^-(T)$ iff $u \neq pred(v)$ and $v \neq pred(u)$. We use $E^- = E^-(T)$ where T is the input tree. We use $E^-(v) = E^-(N[v])$.

2.2 Related Works

The broadcasting problem has been studied for several decades. It was firstly introduced by Slater *et al.* [5]; they proved that both finding the optimal broadcast center and finding the minimum broadcast time on general graphs are NP-complete. They also proposed a linear-time algorithm for finding the optimal broadcast center and minimum broadcast time on trees under the uniformed telephone model.

There are approximation algorithms proposed for finding the minimum broadcast time. An $O(\frac{log^2(n)}{loglog(n)})$-approximate algorithm was presented in [6], where n is the number of vertices. An $O(\sqrt{n})$-additive-approximate algorithm was presented in [7]. Also, there are approximation algorithms for finding the minimum multicast time from a vertex to a subset of k vertices. An $O(\frac{log(n)}{loglog(k)})$-approximate algorithm was presented in [1]; an $O(\frac{log(k)}{loglog(k)})$-approximate algorithm was presented in [8]. Besides, there are some heuristic algorithms [9]–[11] proposed for finding the minimum broadcast time.

There are also polynomial-time exact algorithms for finding the minimum broadcast time on some special graphs, like unicyclic graphs [12], necklace graphs [13], fully connected trees [14], hypercube of trees [15], and others [16]. They all are under the uniformed telephone model. On the other hand, Su *et al.* [17] improved [5] by suggesting a linear-time exact algorithm for finding the optimal broadcast center on trees under the heterogeneous postal model.

There are also different variants [18]–[22] to the broadcasting problem. In [18], the concept of minimal broadcast graph was introduced and several instances of minimal broadcast graphs were shown. In [19], an efficient routing method was presented to transmit multiple messages from a specific node to the other nodes of a complete graph. In [20], a linear-time algorithm was presented for finding the minimal number of centers on trees under the uniformed telephone model with time constraint. In [21], assuming that the maximal vertex degree is 3 (and 4, respectively), the problem of how to augment edges so that the broadcast can be completed in logarithmic time was investigated. In [22], the problem of finding the optimal broadcast 1-median on general graphs was proved NP-complete and a linear-time algorithm was proposed for finding the optimal broadcast 1-median on trees under the heterogeneous postal model. Interested readers can refer to survey articles [23]–[26] for more detailed description.

Chapter 3

A Linear-Time Algorithm

In this chapter, we introduce the algorithm to find the broadcast centers in a tree $T = (V, E)$ with time constraint t is provided, then we give an $O(n)$-time algorithm where n is the number of vertices in T. The basic idea of the algorithm is using a greedy approach. We firstly describe the main structure of the algorithm, and then the implement details, and we will show that the algorithm runs in $O(n)$ time at the end of this chapter.

3.1 Algorithm Description

Like many algorithms on trees, the algorithm processes from leaves to the root. The algorithm is flexible; it does not require strictly process along the level of vertices. A vertex can be processed as long as it has at most one unprocessed neighbor. We will give two examples in Chapter 5; two different processing sequences will be illustrated in each example.

For each process on a vertex, the algorithm finds its successors, broadcast time and predecessor by its successor candidates and predecessor candidates. Then, the algorithm determines if this vertex can be a successor canditate of its unprocessed neighbor. If not, the algorithm evaluates its earliest spare time and then determines if this vertex can be a predecessor candidate of its unprocessed neighbor. The algorithm returns one plus the number of unused edges as the answer, the minimal number of centers needed. The algorithm is described as below.

Algorithm 1 Broadcast

```
Input:
    A weighted tree graph $T = (V, E)$.
    The time constraint $t$ and the connection time $\alpha$.
Output:
    The minimal number of centers needed.

 1: for each $v \in V(T)$ do
 2:     set $SUCC^*(v)$, $SUCC(v)$, $PRED^*(v)$, $pred(v)$ to empty;
 3: end for
 4: $T' \leftarrow T$;        /* the set of unhandled vertices */
 5: $E^- \leftarrow \emptyset$;        /* the set of unused edges */
 6: repeat
 7:     arbitrarily find a leaf $v$ in $T'$;
 8:     Compute $SUCC(v)$, $b\_time(v)$ and $pred(v)$;
 9:     $E^- \leftarrow E^- \cup \{(w, v) | w \in N(v) - \{u, pred(v)\} - SUCC(v)\}$;
10:     if $|V(T')| \geq 2$ then
11:         Let $u$ be the neighbor of $v$ in $T'$;
12:         if $pred(v) = \emptyset$ and $\alpha + w(u, v) + b\_time(v) \leq t$ then
13:             add $v$ to $SUCC^*(u)$;
14:         else if $spare(v) + \alpha + w(v, u) \leq t$ then
15:             add $v$ to $PRED^*(u)$;
16:         end if
17:     end if
18:     remove $v$ from $T'$;
19: until $|V(T')|$ is empty;
20: return 1 + $|E^-|$.
```

Implement details of the algorithm is introduced in the following sections. The details include finding $SUCC(v)$, $b\_time(v)$, and $spare(v)$. Finding $pred(v)$ is easy. Let p be a vertex in $PRED^*(v)$ such that the value $spare(p) + w(p, v)$ is minimized. If $spare(p) + \alpha + w(p, v) + b\_time(v) \leq t$, then p is the predecessor of v; otherwise, v has no predecessor.

3.2 Finding $SUCC(v)$ and $b\_time(v)$

The algorithm reorders $SUCC^*(v)$ first, which is described in Section 3.4. Then, the algorithm choose successor from the rear of $SUCC^*(v)$ to the front of $SUCC^*(v)$. The algorithm observes the change of the broadcast time (b_time') after a vertex joins $SUCC(v)$. If the broadcast time does not exceed the time constraint, this vertex can be a successor of v; otherwize, the number of successors of v reaches its maximum. We will show the correctness of the algorithm in Section 4.1. The algorithm below determines $SUCC(v)$ and $b\_time(v)$ with the given successor candidates $SUCC^*(v)$.

Algorithm 2 Finding $SUCC(v)$ and $b\_time(v)$

Input:

The successor candidates $SUCC^*(v)$.

Output:

The successors $SUCC(v)$ and the broadcasting time $b\_time(v)$.

```
1:  $SUCC^*(v) \leftarrow reorder(SUCC^*(v))$;
2:  /* Suppose $SUCC^*(v)$ is $(u_1, u_2, \cdots, u_k)$ now. */
3:  $b\_time(v) \leftarrow 0$;
4:  $SUCC(v) \leftarrow \emptyset$;
5:  for $i = k$ to $1$ do
6:      $b\_time' \leftarrow max(\{\alpha + b\_time(v), \alpha + w(v, u_i) + b\_time(u_i)\})$;
7:      if $b\_time' \leq t$ then
8:          add $u_i$ to the front of $SUCC(v)$;
9:          $b\_time(v) \leftarrow b\_time'$;
10:     end if
11: end for
12: return $SUCC(v)$ and $b\_time(v)$.
```

Here is an example the algorithm determines $SUCC(v)$ and $b\_time(v)$. We use the tree T_{89535}, which is shown in 3.1. Let $\alpha = 2$ and $t = 17$. The successor candidates of v are $(u_1, u_2, u_3, u_4, u_5)$, which are already reordered.

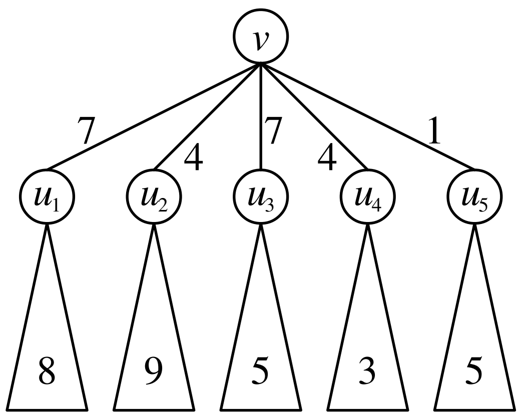

Figure 3.1: Tree T_{89535}.

Initially $b\_time(v) = 0$. The algorithm determines if u_5 is a successor of v. Because $\alpha + b\_time(v) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(v, u_5) + b\_time(u_5) = 2 + 1 + 5 = 8 \leq 17 = t$, the algorithm tells us u_5 is a successor of v and $b\_time(v)$ becomes 8. Now the algorithm determines if u_4 is a successor of v. Because $\alpha + b\_time(v) = 2 + 8 = 10 \leq 17 = t$ and $\alpha + w(v, u_4) + b\_time(u_4) = 2 + 4 + 3 = 9 \leq 17 = t$, the algorithm tells us u_4 is a successor of v and $b\_time(v)$ becomes 10.

Now the algorithm determines if u_3 is a successor of v. Because $\alpha + b\_time(v) = 2 + 10 = 12 \leq 17 = t$ and $\alpha + w(v, u_3) + b\_time(u_3) = 2 + 7 + 5 = 14 \leq 17 = t$, the algorithm tells us u_3 is a successor of v and $b\_time(v)$ becomes 14. Now the algorithm determines if u_2 is a successor of v. Because $\alpha + b\_time(v) = 2 + 14 = 16 \leq 17 = t$ and $\alpha + w(v, u_2) + b\_time(u_2) = 2 + 4 + 9 = 15 \leq 17 = t$, the algorithm tells us u_2 is a successor of v and $b\_time(v)$ becomes 16. Now the algorithm determines if u_1 is a successor of v. Because $\alpha + b\_time(v) = 2 + 16 = 18 > 17 = t$, the algorithm tells us u_1 cannot be a successor of v.
The algorithm concludes that
$SUCC(v) = (u_2, u_3, u_4, u_5)$ and $b\_time(v) = 16$.

3.3 Evaluatng $spare(v)$

We have evaluated $SUCC(v)$, but u, the unprocessed neighbor of v, may be also a possible successor of v. How early v can set up a connection to u depends on how many accepted successor of v can delay α time. The term "can delay α time" is defined as below. Let u_i be the i-th successor of $pred(u_i)$. A successor u_i is said can delay α time iff $(i+1)\alpha + w(u_i, v) + b\_time(u_i) \leq t$.

The algorithm checks which successors of v can delay α time from the rear of $SUCC(v)$ to the front of $SUCC(v)$. The earliest spare time of v is equal to the earliest time v knows the message + time number of successors which cannot delay α time $*\alpha$. We will show the correctness of this algorithm in Section 4.2. The algorithm below evaluates $spare(v)$ when $SUCC(v)$ is given.

Algorithm 3 Finding $spare(v)$

Input:
The successors $SUCC(v) = (u_1, u_2, \cdots, u_k)$.

Output:
The spare time $spare(v)$.

```
1: $h \leftarrow k + 1$;
2: for $i = k$ to 1 do
3:     if $arrive(v) + (i+1)\alpha + w(v, u_i) + b\_time(u_i) \leq t$ then
4:         $h \leftarrow i$;
5:     else
6:         break;
7:     end if
8: end for
9: return $arrive(v) + (h-1)\alpha$.
```

We continue from the example on T_{89535} to demostrate the algorithm for finding $spare(v)$. Initially $spare(v) = arrive(v) + |SUCC(v)|\alpha = 0 + 4 * 2 = 8$. Because $(4 + 1)\alpha + w(v, u_5) + b\_time(u_5) = 10 + 1 + 5 = 16 \leq 17 = t$, the algorithm tells us u_5 can delay α time. Because $(3+1)\alpha+w(v, u_4)+b\_time(u_4) = 8+4+3 = 15 \leq 17 = t$, the algorithm tells us u_4 can delay α time. Because $(2+1)\alpha+w(v, u_3)+b\_time(u_3) = 6 + 7 + 5 = 18 > 17 = t$, the algorithm tells us u_3 cannot delay α time. There are two successors, u_2 and u_3, cannot delay α time, so the algorithm concludes that $spare(v) = arrive(v) + 2\alpha = 0 + 2 * 2 = 4$.

3.4 A Non-Sorting Method

The order of successors is important, so the algorithm reorders the successor candidates. The comparison key is $w(v, u_i) + b\_time(u_i)$, where u_i is a successor of v. For convenience, we use "the length of u_i" to represent $w(v, u_i) + b\_time(u_i)$ in this section. Since sorting k elements by comparion runs in $\Omega(klogk)$ time, we need an $O(k)$-time alternative to keep Algorithm Broadcast can be done in $O(n)$ time. This method was introduced in [17].

Before introducing this non-sorting method, we explain why the comparison key is $w(v, u_i) + b\_time(u_i)$ first. We start from a simple example, the tree T_{368}, which is shown in Figure 3.2.

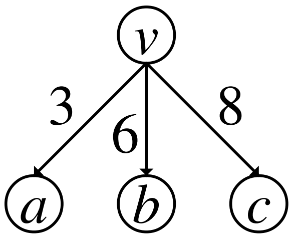

Figure 3.2: Tree T_{368}.

Let $\alpha = 2$, $SUCC(v) = \{a, b, c\}$, $w(v, a) = 3$, $w(v, b) = 6$, $w(v, c) = 8$ and $SUCC(a) = SUCC(b) = SUCC(c) = \emptyset$ (so $b\_time(a) = b\_time(b) = b\_time(c) = 0$). There are 6 possible permutation of $SUCC(v)$. The value of $b\_time(v)$ is shown in Table 3.1.

Table 3.1: Different broadcast sequences may lead to different broadcast time

$SUCC(v)$		$b\_time(v)$	
(a, b, c)	$max(\{\alpha + 3, 2\alpha + 6, 3\alpha + 8\})$	$= max(\{5, 10, 14\})$	$= 14$
(a, c, b)	$max(\{\alpha + 3, 2\alpha + 8, 3\alpha + 6\})$	$= max(\{5, 12, 12\})$	$= 12$
(b, a, c)	$max(\{\alpha + 6, 2\alpha + 3, 3\alpha + 8\})$	$= max(\{8, 7, 14\})$	$= 14$
(b, c, a)	$max(\{\alpha + 6, 2\alpha + 8, 3\alpha + 3\})$	$= max(\{8, 12, 9\})$	$= 12$
(c, a, b)	$max(\{\alpha + 8, 2\alpha + 3, 3\alpha + 6\})$	$= max(\{10, 7, 12\})$	$= 12$
(c, b, a)	$max(\{\alpha + 8, 2\alpha + 6, 3\alpha + 3\})$	$= max(\{10, 10, 9\})$	$= 10$

Observe that the minimal value of $b\_time(v)$, 10, occurs when the successors of v are sorted by $w(v, u_i) + b\_time(u_i)$ descendly. The following lemma shows that we can obtain the minimum broadcast time if we sort $SUCC(v)$ by $w(v, u_i) + b\_time(u_i)$ descendly.

Lemma 1 *Under the constraint $SUCC(v) = \{s_1, s_2, \cdots, s_k\}$, $b\_time(v)$ is minimal if $SUCC(v) = (s_1, s_2, \cdots, s_k)$ and $w(v, s_1) + b\_time(s_1) \geq w(v, s_j) + b\_time(s_j) \forall i < j$.*

Proof.

$$\begin{aligned}
& \neg(w(v,s_i) + b\_time(s_i) \geq w(v,s_j) + b\_time(s_j) \forall i < j) \\
\equiv\ & \exists i < j \text{ such that } w(v,s_i) + b\_time(s_i) < w(v,s_j) + b\_time(s_j) \\
& max(i\alpha + w(v,s_i) + b\_time(s_i), j\alpha + w(v,s_j) + b\_time(s_j)) \\
=\ & j\alpha + w(v,s_j) + b\_time(s_j) \\
>\ & j\alpha + w(v,s_i) + b\_time(s_i) \\
& j\alpha + w(v,s_j) + b\_time(s_j) \\
>\ & i\alpha + w(v,s_j) + b\_time(s_j) \\
\therefore\ & max(i\alpha + w(v,s_i) + b\_time(s_i), j\alpha + w(v,s_j) + b\_time(s_j)) \\
& max(j\alpha + w(v,s_i) + b\_time(s_i), i\alpha + w(v,s_j) + b\_time(s_j)) \\
\Rightarrow\ & \text{swapping } s_i \text{ and } s_j \text{ in } SUCC(v) \text{ improves } b\_time(v)
\end{aligned}$$

For any initial sequence of $SUCC(v)$, we can repeat swapping s_i and s_j until $\nexists i < j$ such that $w(v,s_i) + b\_time(s_i) < w(v,s_j) + b\_time(s_j)$ to improve $b\_time(v)$. This means $b\_time(v)$ is minimal if $SUCC(v) = (s_1, s_2, \cdots, s_k)$ and $w(v,s_i)+b\_time(s_i) \geq w(v,s_j)+b\_time(s_j) \forall i < j$.

□

After explaining why the comparison key is $w(v,u_i) + b\_time(u_i)$, we introduce the non-sorting method. The algorithm calssifies k vertices (S) to $k+1$ lists. The algorithm firstly finds the longest one (u_1) and remembers its length. Then, for every vertex u_i, the algorithm computes the difference between the length of u_i and length of u_1. The algorithm computes the quotient this difference divided by α (j). If $j \leq k$, it means the length of u_i is short enough and the algorithm puts u_i into the last list ($list_k$).

Otherwize, the algorithm puts u_i into $list_j$. For each nonempty list, the algorithm moves the element with the shortest length to the end of the list. Finally, the algorithm concats these $k+1$ lists by the lsit number. We will show this non-sorting method does not break the optimalness of the algorithm in the next chapter. The algorithm below describes this non-sorting method.

Algorithm 4 reorder

```
Input:
    A vertices set $S = \{u_1, u_2, \cdots, u_k\}$.
Output:
    A permutation of $\{u_1, u_2, \cdots, u_k\}$.

1: Let $u_1$ be the vertex in S such that
   $w(v, u_1) + b\_time(u_1) = max(\{w(v, u) + b\_time(u) | u \in S\})$ ;
2: Create $k+1$ linked lists, $list_0, list_1, \cdots, list_k$; $list_j$ contains vertices $u_i$ such that
3:     $j\alpha \leq (w(v, u_1) + b\_time(u_1)) - (w(v, u_i) + b\_time(u_i)) < (j+1)\alpha$ iff $0 \leq j < k$, and
4:     $k\alpha \leq (w(v, u_1) + b\_time(u_1)) - (w(v, u_i) + b\_time(u_i))$ iff $j = k$;
5: Let $u_j^*$ be a vertex in $list_j$ such that
   $w(v, u_j^*) + b\_time(u_j^*) = min(\{w(u, v) + b\_time(u) | u \in list_j\})$ ;
6: Move $u_j^*$ to the end of $list_j$;
```

Figure 3.3 shows an example of reordering. In this case, $k = 10$, the length values are 23, 60, 44, 21, 43, 7, 24, 8, 41, 9 and $\alpha = 5$.

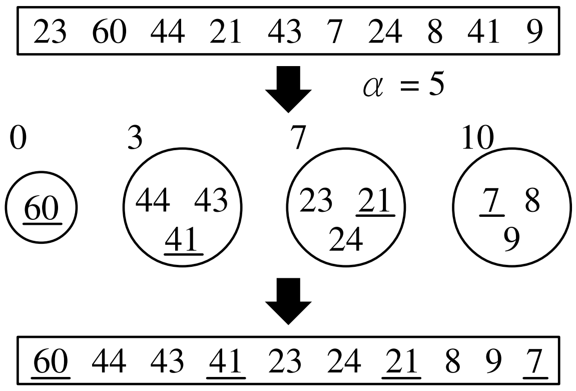

Figure 3.3: An Example of Reordering.

According to the algorithm, we have $u_1 = 60$, $list_0 = \{60\}$ with $u_0^* = 60$, $list_3 = \{44, 43, 41\}$ with $u_3^* = 41$, $list_7 = \{23, 21, 24\}$ with $u_7^* = 21$, $list_{10} = \{7, 8, 9\}$ with $u_{10}^* = 7$ and $list_1$, $list_2$, $list_4$, $list_5$, $list_6$, $list_8$, $list_9$ are empty. The reordred sequence is $(60, 44, 43, 41, 23, 24, 21, 8, 9, 7)$.

To make our proof easier, we keep u_1 the first place in the bucketed sequence. Also, we define several terms related to this non-sorting method. A sequence $(u_1, u_2, \cdots, u_k)$ is bucketed iff it is a possible output of $reorder(\{u_1, u_2, \cdots, u_k\}, false)$. A sequence is deheadedly bucketed iff it is a subsequence $(u_i, u_{i+1}, \cdots, u_k)$ of a bucketed sequence and $u_i \notin list_k$. A sequence is reversed bucketed iff it can be obtained from reversing a bucketed sequence.

3.5 Time Complexity

In this section, we discuss the complexity of the algorithm. We analyze the detailed ones before the mainly ones. Complicated analysis is not required.

Lemma 2 *Algorithm 4 runs in $O(k)$ time.*

Proof. Line 1 takes $O(k)$ time. Line 2 takes $O(k)$ time. The determination from Line 3 to Line 4 takes constant time for each u_i and $O(k)$ time for all vertices in S. Line 5 and Line 6 takes $O(|list_j|)$ time for $list_j$ and $O(k)$ time for $\{list_0, list_1, \cdots, list_k\}$. $\square$

Lemma 3 *Algorithm 2 runs in $O(k)$ time.*

Proof. Line 1 takes $O(k)$ time. Line 6 to line 10 takes constant time, so Line 5 to line 11 takes $O(k)$ time. Other statements takes constant time. $\square$

Lemma 4 *Algorithm 3 runs in $O(k)$ time.*

Proof. Line 3 to line 7 takes constant time, so Line 2 to line 8 takes $O(k)$ time. Other statements takes constant time. $\square$

Theorem 5 *Algorithm Broadcast runs in $O(n)$ time.*

Proof. Line 2 takes constant time, so the for loop from line 1 to line 3 takes $O(n)$ time. Line 4 takes $O(n)$ time and line 5 takes constant time. Finding $pred(v), SUCC(v), b\_time(v)$ and $spare(v)$ takes $O(|N(v)|)$ time and other statements from line 7 to line 17 takes constant time, so the repeat-until loop from line 6 to line 19 takes $O(\sum_{v \in V(T)} |N(v)|) = O(n)$ time. $\square$

Chapter 4

Correctness

The goal of this chapter is showing that the proposed algorithm indeed determines the minimal number of centers needed. The optimalness to the algorithm is based on the mutual trust between nodes. That is, every node believes its processed neighbors perform their best, and it can also perform the best to its unprocessed neighbor. The performance of a node v is defined as below:

$$perf(v, SUCC(v), pred(v)) = (|E^-(v)|, b\_time(v), spare(v)).$$

We say $(|E^-(v)|, b\_time(v), spare(v)) \leq (|E^{-\prime}(v)|, b\_time'(v), spare'(v))$ iff

(1) $|E^-(v)| < |E^{-\prime}(v)|$, or

(2) $|E^-(v)| = |E^{-\prime}(v)|$, $b\_time(v) \leq b\_time'(v)$ and

$\alpha + w(u,v) + b\_time(v) \leq t$, or

(3) $|E^-(v)| = |E^{-\prime}(v)|$, $\alpha + w(u,v) + b\_time(v) > t$,

$\alpha + w(u,v) + b\_time'(v) > t$,

$spare(v) \leq spare'(v)$ and

$spare(v) + \alpha + w(v,u) \leq t$, or

(4) $|E^-(v)| = |E^{-\prime}(v)|$, $\alpha + w(u,v) + b\_time(v) > t$,

$\alpha + w(u, v) + b\_time'(v) > t$,

$spare(v) + \alpha + w(v, u) > t$ and

$spare'(v) + \alpha + w(v, u) > t$.

Smaller is better.

The performance of a node is made up of three parts. The first one is unused edges; it depends on the number of successors and whether the node has a predecessor. The second one is its broadcast time; it may affect whether this node can be a successor of its unprocessed neighbor. The third one is its earliest spare time; it may affect whether this node can be the predecessor of its unprocessed neighbor. In this chapter, we will show the minimalness of $|E^-(v)|$, $b\_time(v)$ and $spare(v)$ respectively and then combining them for the correctness of the algorithm.

4.1 Minimum Unused Edges and Broadcast Time

The number of unused edges of v, $|E^-(v)|$, depends on $SUCC(v)$ and $pred(v)$. In this section, we show the algorithm can determine the maximum of $|SUCC(v)|$ and then discuss the difference between finding $pred(v)$ before and after finding $SUCC(v)$.

Lemma 6 *Let* $SUCC^*(v) = \{u_1, u_2, \cdots, u_k\}$ *and* $w(v, u_1) + b\_time(u_1) \leq w(v, u_2) + b\_time(u_2) \leq \cdots \leq w(v, u_k) + b\_time(u_k)$. *If we want to choose* f *nodes from* $SUCC^*(v)$ *to be the successors, then choosing* $(u_f, u_{f-1}, \cdots, u_1)$ *is* $b\_time(v)$*-optimal.*

Proof. Suppose we choose $(u'_f, u'_{f-1}, \cdots, u'_1)$. By Lemma 1, we may assume $w(v, u'_f) + b\_time(u'_f) \geq w(v, u'_{f-1}) + b\_time(u'_{f-1}) \geq \cdots \geq$

$w(v, u'_1) + b\_time(u'_1)$.
Since $w(v, u_f) + b\_time(u_f) \geq w(v, u'_f) + b\_time(u'_f)$,
$w(v, u_{f-1}) + b\_time(u_{f-1}) \geq w(v, u'_{f-1}) + b\_time(u'_{f-1}), \cdots,$
$w(v, u_1) + b\_time(u_1) \geq w(v, u'_1) + b\_time(u'_1)$,
we have $b\_time(v, (u_f, u_{f-1}, \cdots, u_1)) \geq b\_time(v, (u'_f, u'_{f-1}, \cdots, u'_1))$.
□

Lemma 7 *Using the method in Lemma 6, we can obtain the maximal number of* $|SUCC(v)|$.

Proof. We can simply choose f such that $b\_time(v, (u_f, u_{f-1}, \cdots, u_1))$ is maximized under the constraint $b\_time(v, (u_f, u_{f-1}, \cdots, u_1)) \leq t$. □

Lemma 8 *Let* $(u_1, u_2, \cdots, u_k)$ *be a bucketed sequence and* $(u'_1, u'_2, \cdots, u'_k)$ *be its sorted permutation; that is,* $w(v, u'_1) + b\_time(u'_1) \geq w(v, u'_2) + b\_time(u'_2) \geq \cdots \geq w(v, u'_k) + b\_time(u'_k)$.
Then, $b\_time(v, (u_1, u_2, \cdots, u_k)) = b\_time(v, (u'_1, u'_2, \cdots, u'_k))$.

Proof. For each $u_x \in list_k$, we have $arrive(u_x) = arrive(v) + x\alpha + w(v, u_x) + b\_time(u_x) \leq arrive(v) + x\alpha + w(v, u'_1) + b\_time(u'_1) - k\alpha \leq arrive(v) + w(v, u_1) + b\_time(u_1) = arrive(u_1)$. Let $u_y = u^*_j$ for some $j \in \{0, 1, \cdots, k-1\}$ and $u_z \in list_j - \{u^*_j\}$. We have $arrive(u_y) = arrive(v) + y\alpha + w(v, u_y) + b\_time(u_y) = arrive(v) + y\alpha + w(v, u_z) + b\_time(u_z) - \epsilon\alpha \geq arrive(v) + z\alpha + w(v, u_z) + b\_time(u_z) = arrive(u_z)$ for some ϵ such that $0 \leq \epsilon < 1$. This means only $u^*_0, u^*_1, \cdots, u^*_{k-1}$ may dominate $b\_time(v)$. Since $u_y = u'_y$ and the sorted permutation $(u'_1, u'_2, \cdots, u'_k)$ is also a bucketed sequence, we have $arrive(u_y) = arrive(u'_y)$ and therefore this lemma holds. □

Lemma 9 *Let $SUCC^*(v) = \{u_1, u_2, \cdots, u_k\}$.*

If $(u_1, u_2, \cdots, u_k)$ is reversed bucketed sequence,

then $b\_time(v) = b\_time(v, (u_f, u_{f-1}, \cdots, u_1))$

where f is obtained from Lemma 7.

Proof. If $list_k$ dominates $b\_time(v)$, since we can run Procedure 4 on $list_k$ to make $list_k$ bucketed without breaking $O(n)$-time, by Lemma 8, $b\_time(v)$ is optimal. If $list_k$ does not dominate $b\_time(v)$, by the same argument in Lemma 8, a vertex in $\{u_0^*, u_1^*, \cdots, u_{k-1}^*\} \cap \{u_f, u_{f-1}, \cdots, u_1\}$ dominates $b\_time(v)$, so this lemma holds. $\square$

Lemma 10 *If we find $pred(v)$ before finding $SUCC(v)$, the performance of v cannot be better.*

Proof. If assigning a predecessor causes $|SUCC(v)|$ decreased by two or more, $|E^-(v)|$ is increased by at least one. If the $|SUCC(v)|$ does not change, then there are no differences between finding $pred(v)$ before and after finding $SUCC(v)$. If $|SUCC(v)|$ is decreased by one, $|E^-(v)|$ does not change; however, it becomes impossible that $v \in SUCC(u)$, and $spare(v)$ is increased by at least α due to $pred(v)$ and decreased by at most α thanks to the removed successor of v. $\square$

Lemma 11 *If $v \in SUCC^*(u)$, then $v \notin PRED^*(u)$.*

Proof. If $v \notin SUCC(u)$, then $t - b\_time(u) < \alpha$, so $\alpha + w(v,u) + b\_time(u) > t + w(v,u) \geq t$, v cannot be a predecessor candidate of u. If $v \in SUCC(u)$, since Lemma 10 tells us the performance of v cannot be better if $pred(u) = v$, we do not need to include v to $PRED^*(u)$. $\square$

4.2 Earliest Spare Time

We show that the earliest spare time is minimal with a sorting method first, and then we can also obtain the minimum with the non-sorting method. We use notations in Procedure 4.

Lemma 12 $u_i \in list_k \Rightarrow u_i$ *can delay* α *time.*

Proof. $u_i \in list_k \Rightarrow w(u_i, v) + b\_time(u_i) \leq w(u_1, v) + b\_time(u_1) - k\alpha$, so $(i+1)\alpha + w(u_i, v) + b\_time(u_i) \leq (i+1)\alpha + w(u_1, v) + b\_time(u_1) - k\alpha \leq \alpha + w(u_1, v) + b\_time(u_1) \leq t$. □

Lemma 13 *Let* $u_i \in list_j - \{u_j^*\}$ *for some* $j \in \{0, 1, \cdots, k-1\}$. *We have* u_j^* *can delay* α *time* $\Rightarrow u_i$ *can delay* α *time.*

Proof. Let $u_j^* = u_{i'}$. We have $(i+1)\alpha + w(u_i, v) + b\_time(u_i) \leq (i'-1+1)\alpha + w(u_i, v) + b\_time(u_i) < (i'+1)\alpha + w(u_{i'}, v) + b\_time(u_{i'}) \leq t$. □

Lemma 14 *Let* $SUCC(v) = (u_1, u_2, \cdots, u_k)$. *If* $w(v, u_1) + b\_time(u_1) \geq w(v, u_2) + b\_time(u_2) \geq \cdots \geq w(v, u_k) + b\_time(u_k)$*, our algorithm can determine the earliest spare time* $spare(v)$.

Proof. If $spare(v) = arrive(v) + (i-1)\alpha$, we need to choose $k-i$ successors of v, denoted by $SUCC'(v)$, such that $b\_time(v, SUCC'(v)) \leq t - arrive(v) - (i-1)\alpha$. By Lemma 6, choosing $u_{i+1}, u_{i+2}, \cdots, u_k$ can obtain minimal $b\_time(v, SUCC'(v))$ under $|SUCC'(v)|$ is fixed by i. Since $b\_time(v, (u_i, u_{i+1}, \cdots, u_k)) \geq b\_time(v, (u_{i+1}, u_{i+2}, \cdots, u_k)) + \alpha$, we have $t - arrive(v) - (i-1)\alpha - b\_time(v, (u_{i+1}, u_{i+2}, \cdots, u_k)) \leq t - arrive(v) - i\alpha - b\_time(v, (u_i, u_{i+1}, \cdots, u_k))$. This implies $\exists f \in$

$\{1, 2, \cdots, k+1\}$ such that $b\_time(v, SUCC'(v)) \leq t - arrive(v) - (i-1)\alpha$ if $i \geq f$ and $b\_time(v, SUCC'(v)) > t - arrive(v) - (i-1)\alpha$ if $i < f$ and our algorithm determines f and the earliest spare time $spare(v)$ is exactly $arrive(v) + (f-1)\alpha$. □

Lemma 15 *Continued from the previous lemma. If $SUCC(v)$ is deheadedly bucketed, our algorithm can still determine the earliest spare time $spare(v)$.*

Proof. Lemma 13 implies the number of successors which can delay α time is equal for any two different deheadedly bucketed sequence of $SUCC(v)$. Since the sorted sequence is also deheadedly bucketed, this number is equal to $k - f + 1$ where f is introduced in the proof of the previous lemma. □

4.3 Correctness of the Algorithm

Combining Lemma 7, Lemma 9, Lemma 10 and Lemma 15, we have the result:

Lemma 16 *Given $SUCC^*(v)$ including $b\_time(x) \forall x \in SUCC^*(v)$ and $PERD^*(v)$ including $spare(p) \forall p \in PRED^*(v)$, the algorithm determines $SUCC(v)$ and $pred(v)$ such that $perf(v, SUCC(v), pred(v))$ is the best.*

Before showing the correctness of the algorithm, we still need a lemma:

Lemma 17 *Let $SUCC^*(v) = \{u_1, u_2, \cdots, u_k\}$, $SUCC^*(v') = \{u'_1, u'_2, \cdots, u'_k\}$, $w(v, u_j) = w(v', u'_j) \forall j \in \{1, 2, \cdots, k\}$*

and $arrive(v) = arrive(v')$. *If* $\exists! i \in \{1, 2, \cdots, k\}$ *such that* $b\_time(u'_i) < b\_time(u_i)$ *and* $b\_time(u_j) = b\_time(u'_j) \forall j \in \{1, 2, \cdots, k\} - \{i\}$, *then* $|SUCC(v')| \leq |SUCC(v)| + 1$.

Proof. Without loss of generality,
we may assume $w(v, u_a) + b\_time(u_a) \leq w(v, u_b) + b\_time(u_b) \forall a < b$.
Let $f = |SUCC(v)|$. If $|SUCC(v')| \geq f + 2$, by Lemma 6,
$b\_time(v', SUCC(v')) \geq b\_time(v', \{u'_{f+2}, u'_{f+1}, \cdots, u'_1\}) \geq$
$b\_time(v, \{u_{f+1}, u_f, \cdots, u_1\})$, which means broadcasting cannot be done within the time constraint. Thus, $|SUCC(v')| \geq f + 2$ is impossible. □

Theorem 18 *Algorithm Broadcast indeed determines the minimal number of centers needed.*

Proof. We prove by induction. We want to proof every time a vertex is processed, $|E^-| = OPT(T - T', t) - 1$; under this condition, for each node v satisfying v has an unprocessed neighbor,
$perf(v, SUCC(v), pred(v))$ is minimized.

Let v_1 be a leaf of T. Observe that v_1 has no choices; that is,
$SUCC^*(v_1) = \emptyset$ and $PRED^*(v_1) = \emptyset$
$\Rightarrow SUCC(v_1) = \emptyset$ and $pred(v_1) = nil$
$\Rightarrow E^-(v_1) = 0 = 1 - 1 = OPT(\{v_1\}, t) - 1$.
$\because SUCC(v_1) = \emptyset, \therefore b\_time(v) = 0$ and $spare(v) = 0$.
Therefore, $perf(v_1, \emptyset, nil) = (0, 0, 0)$ is obviously minimal.

Suppose before line 2, $|E^-| = OPT(T - T' - \{v\}, t) - 1$; under this condition, for each node v satisfying v has an unprocessed neighbor,

$perf(v, SUCC(v), pred(v))$ is minimized. We prove $|E^-| = OPT(T - T', t) - 1$; under this condition, for each node v satisfying v has an unprocessed neighbor,
$perf(v, SUCC(v), pred(v))$ is minimized.

By Lemma 16, it is impossible to find a strategy to make $perf(v, SUCC(v), pred(v))$ smaller without modifying $SUCC(x)$ for some $x \in T - T'$. To make $perf(v, SUCC(v), pred(v))$ smaller, we have only two choices. The first choice is $\exists s \in N(v) - \{u\}$ such that $b\_time(s)$ is decreased, and the other choice is $\exists p \in N(v) - SUCC^*(v)$ such that $spare(p)$ is decreased.

For each time the challenger makes a node $\exists s \in N(v) - \{u\}$ such that $b\_time(s)$ is decreased, by the induction hypothesis, $|E^-(s)|$ must be decreased by at least one. After $b\_time(s)$ is decreased, we observe the change of $SUCC(v)$. Lemma 17 tells us v can accept only one more successor.

If there is a node s' joining $SUCC(v)$, since $b\_time(v, SUCC(v) \cup \{s'\}) \geq b\_time(v, SUCC(v) \cup \{s'\} - \{s\}) \geq b\_time(v, SUCC(v))$ and $spare(v, SUCC(v) \cup \{s'\}) \geq spare(v, SUCC(v) \cup \{s'\} - \{s\}) \geq spare(v, SUCC(v))$, the challenger fails to make
$perf(v, SUCC(v), pred(v))$ smaller.

If $SUCC(v)$ is not increased by 1, it is possible v can have a predecessor. From $pred(v) = nil$ to $pred(v) \neq nil$, $spare(v)$ is increased by at least α. However, making $b\_time(s)$ decreased can only make $spare(v)$ decreased by at most α. The challenger still cannot make

$perf(v, SUCC(v), pred(v))$ smaller.

It is impossible both a node s' joins $SUCC(v)$ and from $pred(v) = nil$ to $pred(v) \neq nil$ happen because $\forall s^* \in SUCC(v) \cup \{s'\}$, $b\_time(v, SUCC(v) \cup \{s'\} - \{s^*\}) \geq b\_time(v, SUCC(v))$, so $t - arrive(v) - b\_time(v, SUCC(v) \cup \{s'\} - \{s^*\}) \leq t - arrive(v) - b\_time(v, SUCC(v))$, which means v cannot become have ability to have a predecessor after this change by the challenger.

If $\exists p \in N(v) - SUCC^*(v)$ such that $spare(p)$ is decreased, by the induction hypothesis, $|E^-(p)|$ is decreased by at least one; it can be only made up if $pred(v) = nil$ is changed to $pred(v) = p$, which makes $spare(v)$ increased by at least α, so $perf(v, SUCC(v), pred(v))$ cannot be improved. $\square$

Chapter 5

Two Illustrative Examples

In this chapter, we illustrate the algorithm with two arbitrary trees. Both trees are spanning trees of G_0, which is shown in Figure 5.1. Also, we suppose $\alpha = 2$ and $t = 17$ in both examples.

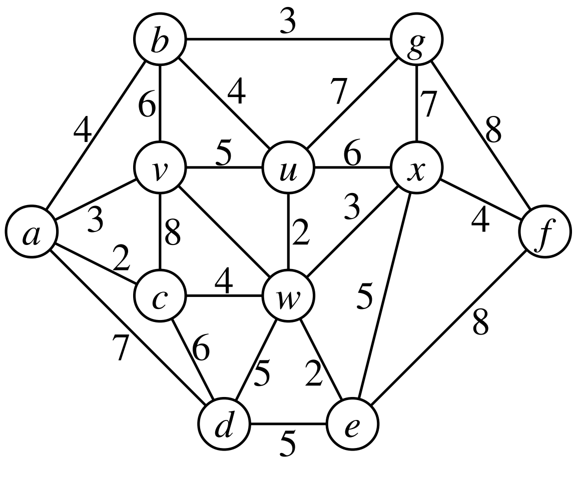

Figure 5.1: Graph G_0.

5.1 The First Example

The tree T_1 is shown in Figure 5.2. We demostrate the algorithm twice with two processing sequence,

$(a, b, c, d, e, f, g, v, w, x, u)$ and $(g, f, x, e, d, w, u, b, c, v, a)$.

We show the case $(a, b, c, d, e, f, g, v, w, x, u)$ first.

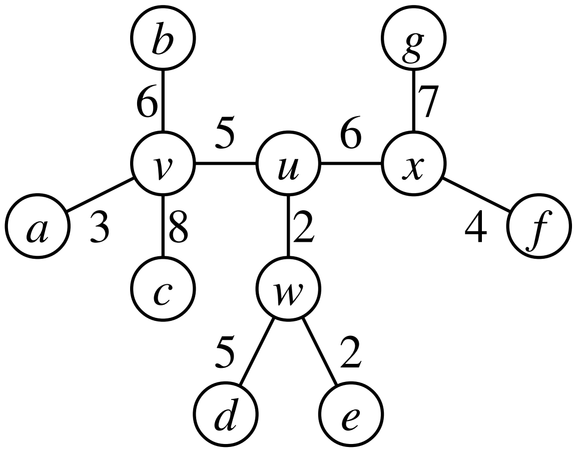

Figure 5.2: Tree T_1.

The 1st processed node is a. Its unprocessed neighbor is v. Because $SUCC^*(a) = \emptyset$ and $PRED^*(a) = \emptyset$, it is obviously $SUCC(a) = \emptyset$, $b\_time(a) = 0$, $pred(a) = \emptyset$ and $spare(a) = 0$. Because $pred(a) = \emptyset$ and $\alpha + w(v, a) + b\_time(a) = 2 + 3 + 0 = 5 \leq 17 = t$, the algorithm adds a to $SUCC^*(v)$.

The 2nd processed node is b. Its unprocessed neighbor is v. Because $SUCC^*(b) = \emptyset$ and $PRED^*(b) = \emptyset$, it is obviously $SUCC(b) = \emptyset$, $b\_time(b) = 0$, $pred(b) = \emptyset$ and $spare(b) = 0$. Because $pred(b) = \emptyset$ and $\alpha + w(v, b) + b\_time(b) = 2 + 6 + 0 = 8 \leq 17 = t$, the algorithm adds b to $SUCC^*(v)$.

The 3rd processed node is c. Its unprocessed neighbor is v. Because $SUCC^*(c) = \emptyset$ and $PRED^*(c) = \emptyset$, it is obviously $SUCC(c) = \emptyset$, $b\_time(c) = 0$, $pred(c) = \emptyset$ and $spare(c) = 0$. Because $pred(c) = \emptyset$ and $\alpha + w(v, c) + b\_time(c) = 2 + 8 + 0 = 10 \leq 17 = t$, the algorithm adds c to $SUCC^*(v)$.

The 4th processed node is d. Its unprocessed neighbor is w. Because $SUCC^*(d) = \emptyset$ and $PRED^*(d) = \emptyset$, it is obviously $SUCC(d) = \emptyset$,

$b\_time(d) = 0$, $pred(d) = \emptyset$ and $spare(d) = 0$. Because $pred(d) = \emptyset$ and $\alpha + w(w, d) + b\_time(d) = 2 + 5 + 0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC^*(w)$.

The 5th processed node is e. Its unprocessed neighbor is w. Because $SUCC^*(e) = \emptyset$ and $PRED^*(e) = \emptyset$, it is obviously $SUCC(e) = \emptyset$, $b\_time(e) = 0$, $pred(e) = \emptyset$ and $spare(e) = 0$. Because $pred(e) = \emptyset$ and $\alpha + w(w, e) + b\_time(e) = 2 + 2 + 0 = 4 \leq 17 = t$, the algorithm adds e to $SUCC^*(w)$.

The 6th processed node is f. Its unprocessed neighbor is x. Because $SUCC^*(f) = \emptyset$ and $PRED^*(f) = \emptyset$, it is obviously $SUCC(f) = \emptyset$, $b\_time(f) = 0$, $pred(f) = \emptyset$ and $spare(f) = 0$. Because $pred(f) = \emptyset$ and $\alpha + w(x, f) + b\_time(f) = 2 + 4 + 0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC^*(x)$.

The 7th processed node is g. Its unprocessed neighbor is x. Because $SUCC^*(g) = \emptyset$ and $PRED^*(g) = \emptyset$, it is obviously $SUCC(g) = \emptyset$, $b\_time(g) = 0$, $pred(g) = \emptyset$ and $spare(g) = 0$. Because $pred(g) = \emptyset$ and $\alpha + w(x, g) + b\_time(g) = 2 + 7 + 0 = 9 \leq 17 = t$, the algorithm adds g to $SUCC^*(x)$.

The 8th processed node is v. Its unprocessed neighbor is u. The algorithm reorders $SUCC^*(v) = \{a, b, c\}$ first. Because $w(v, a) + b\_time(a) = 3 + 0 = 3 = 8 - 2.5\alpha$, $w(v, b) + b\_time(b) = 6 + 0 = 6 = 8 - 1.0\alpha$ and $w(v, c) + b\_time(c) = 8 + 0 = 8$, the algorithm gives $u_1 = c$, $list_0 = \{c\}$, $list_1 = \{b\}$, $list_2 = \{a\}$, $list_3 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (c, b, a).

Now the algorithm starts determining $SUCC(v)$. Because $\alpha + b\_time(v) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(v, a) + b\_time(a) = 2+3+0 = 5 \leq 17 = t$, the algorithm adds a to $SUCC(v)$ and $b\_time(v)$ is 5 now. Because $\alpha+b\_time(v) = 2+5 = 7 \leq 17 = t$ and $\alpha+w(v, b)+ b\_time(b) = 2 + 6 + 0 = 8 \leq 17 = t$, the algorithm adds b to $SUCC(v)$ and $b\_time(v)$ is 8 now.

Because $\alpha + b\_time(v) = 2 + 8 = 10 \leq 17 = t$ and $\alpha + w(v, c) + b\_time(c) = 2+8+0 = 10 \leq 17 = t$, the algorithm adds c to $SUCC(v)$ and $b\_time(v)$ is 10 now. All successor candidates of v are successors of v. Because $PRED^*(v) = \emptyset$, it is obviously $pred(v) = \emptyset$. Because $pred(v) = \emptyset$ and $\alpha + w(u, v) + b\_time(v) = 2 + 5 + 10 = 17 \leq 17 = t$, the algorithm adds v to $SUCC^*(u)$.

The 9th processed node is w. Its unprocessed neighbor is u. The algorithm reorders $SUCC^*(w) = \{d, e\}$ first. Because $w(w, d)+b\_time(d) = 5 + 0 = 5$ and $w(w, e) + b\_time(e) = 2 + 0 = 2 = 5 - 1.5\alpha$, the algorithm gives $u_1 = d$, $list_0 = \{d\}$, $list_1 = \{e\}$, $list_2 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (d, e).

Now the algorithm starts determining $SUCC(w)$. Because $\alpha + b\_time(w) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(w, e) + b\_time(e) = 2+2+0 = 4 \leq 17 = t$, the algorithm adds e to $SUCC(w)$ and $b\_time(w)$ is 4 now. Because $\alpha+b\_time(w) = 2+2 = 4 \leq 17 = t$ and $\alpha+w(w, d)+ b\_time(d) = 2 + 5 + 0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC(w)$ and $b\_time(w)$ is 7 now. All successor candidates of w are successors of w. Because $PRED^*(w) = \emptyset$, it is obviously $pred(w) = \emptyset$. Because

$pred(w) = \emptyset$ and $\alpha + w(u,w) + b\_time(w) = 2+2+7 = 11 \leq 17 = t$, the algorithm adds w to $SUCC^*(u)$.

The 10th processed node is x. Its unprocessed neighbor is u. The algorithm reorders $SUCC^*(x) = \{f, g\}$ first. Because $w(x,f)+b\_time(f) = 4+0 = 4 = 7-1.5\alpha$ and $w(x,g)+b\_time(e) = 7+0 = 7$, the algorithm gives $u_1 = g$, $list_0 = \{g\}$, $list_1 = \{f\}$, $list_2 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (g, f). Now the algorithm starts determining $SUCC(x)$. Because $\alpha + b\_time(x) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(x,f) + b\_time(f) = 2+4+0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC(x)$ and $b\_time(x)$ is 6 now. Because $\alpha + b\_time(x) = 2+6 = 8 \leq 17 = t$ and $\alpha + w(x,g) + b\_time(g) = 2+7+0 = 9 \leq 17 = t$, the algorithm adds g to $SUCC(x)$ and $b\_time(x)$ is 9 now. All successor candidates of x are successors of x. Because $PRED^*(x) = \emptyset$, it is obviously $pred(x) = \emptyset$. Because $pred(x) = \emptyset$ and $\alpha + w(u,x) + b\_time(x) = 2+6+9 = 17 \leq 17 = t$, the algorithm adds x to $SUCC^*(u)$.

The last processed node is u. The algorithm reorders $SUCC^*(u) = \{v, w, x\}$ first. Because $w(u,v) + b\_time(v) = 5+10 = 15 = 8$, $w(u,w)+b\_time(w) = 2+7 = 9 = 15-3.0\alpha$ and $w(u,x)+b\_time(x) = 6+9 = 15$, the algorithm gives $u_1 = v$, $list_0 = \{v, x\}$, $list_1 = \emptyset$, $list_2 = \emptyset$, $list_3 = \{w\}$. Therefore, the reordered sequence of $SUCC^*(u)$ is (v, x, w).

Now the algorithm starts determining $SUCC(u)$. Because
$\alpha + b\_time(u) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(u,w) + b\_time(w) = 2+2+7 = 11 \leq 17 = t$, the algorithm adds w to $SUCC(u)$ and

$b\_time(u)$ is 11 now. Because $\alpha + b\_time(u) = 2 + 11 = 13 \leq 17 = t$ and $\alpha + w(u, x) + b\_time(x) = 2 + 6 + 9 = 17 \leq 17 = t$, the algorithm adds x to $SUCC(u)$ and $b\_time(u)$ is 17 now. Because $\alpha + b\_time(u) = 2 + 17 = 19 > 17 = t$, the algorithm gives v cannot be a successor of u. Because $PRED^*(u) = \emptyset$, it is obviously $pred(u) = \emptyset$. The result is there are two broadcast centers, v and u and there is one unused edge, (v, u).

Now we run the algorithm again, but with an another processing sequence, $(g, f, x, e, d, w, u, b, c, v, a)$. The 1st processed node is g. Its unprocessed neighbor is x. Because $SUCC^*(g) = \emptyset$ and $PRED^*(g) = \emptyset$, it is obviously $SUCC(g) = \emptyset$, $b\_time(g) = 0$, $pred(g) = \emptyset$ and $spare(g) = 0$. Because $pred(g) = \emptyset$ and $\alpha + w(x, g) + b\_time(g) = 2 + 7 + 0 = 9 \leq 17 = t$, the algorithm adds g to $SUCC^*(x)$.

The 2nd processed node is f. Its unprocessed neighbor is x. Because $SUCC^*(f) = \emptyset$ and $PRED^*(f) = \emptyset$, it is obviously $SUCC(f) = \emptyset$, $b\_time(f) = 0$, $pred(f) = \emptyset$ and $spare(f) = 0$. Because $pred(f) = \emptyset$ and $\alpha + w(x, f) + b\_time(f) = 2 + 4 + 0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC^*(x)$.

The 3rd processed node is x. Its unprocessed neighbor is u. The algorithm reorders $SUCC^*(x) = \{f, g\}$ first. Because $w(x, f) + b\_time(f) = 4 + 0 = 4 = 7 - 1.5\alpha$ and $w(x, g) + b\_time(e) = 7 + 0 = 7$, the algorithm gives $u_1 = g$, $list_0 = \{g\}$, $list_1 = \{f\}$, $list_2 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (g, f).

Now the algorithm starts determining $SUCC(x)$. Because

$\alpha + b\_time(x) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(x, f) + b\_time(f) = 2+4+0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC(x)$ and $b\_time(x)$ is 6 now. Because $\alpha+b\_time(x) = 2+6 = 8 \leq 17 = t$ and $\alpha+w(x, g)+ b\_time(g) = 2 + 7 + 0 = 9 \leq 17 = t$, the algorithm adds g to $SUCC(x)$ and $b\_time(x)$ is 9 now. All successor candidates of x are successors of x. Because $PRED^*(x) = \emptyset$, it is obviously $pred(x) = \emptyset$. Because $pred(x) = \emptyset$ and $\alpha + w(u, x) + b\_time(x) = 2 + 6 + 9 = 17 \leq 17 = t$, the algorithm adds x to $SUCC^*(u)$.

The 4th processed node is e. Its unprocessed neighbor is w. Because $SUCC^*(e) = \emptyset$ and $PRED^*(e) = \emptyset$, it is obviously $SUCC(e) = \emptyset$, $b\_time(e) = 0$, $pred(e) = \emptyset$ and $spare(e) = 0$. Because $pred(e) = \emptyset$ and $\alpha + w(w, e) + b\_time(e) = 2 + 2 + 0 = 4 \leq 17 = t$, the algorithm adds e to $SUCC^*(w)$.

The 5th processed node is d. Its unprocessed neighbor is w. Because $SUCC^*(d) = \emptyset$ and $PRED^*(d) = \emptyset$, it is obviously $SUCC(d) = \emptyset$, $b\_time(d) = 0$, $pred(d) = \emptyset$ and $spare(d) = 0$. Because $pred(d) = \emptyset$ and $\alpha + w(w, d) + b\_time(d) = 2 + 5 + 0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC^*(w)$.

The 6th processed node is w. Its unprocessed neighbor is u. The algorithm reorders $SUCC^*(w) = \{d, e\}$ first. Because $w(w, d)+b\_time(d) = 5 + 0 = 5$ and $w(w, e) + b\_time(e) = 2 + 0 = 2 = 5 - 1.5\alpha$, the algorithm gives $u_1 = d$, $list_0 = \{d\}$, $list_1 = \{e\}$, $list_2 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (d, e).

Now the algorithm starts determining $SUCC(w)$. Because

$\alpha + b\_time(w) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(w, e) + b\_time(e) = 2+2+0 = 4 \leq 17 = t$, the algorithm adds e to $SUCC(w)$ and $b\_time(w)$ is 4 now. Because $\alpha+b\_time(w) = 2+2 = 4 \leq 17 = t$ and $\alpha+w(w, d)+ b\_time(d) = 2+5+0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC(w)$ and $b\_time(w)$ is 7 now. All successor candidates of w are successors of w. Because $PRED^*(w) = \emptyset$, it is obviously $pred(w) = \emptyset$. Because $pred(w) = \emptyset$ and $\alpha + w(u, w) + b\_time(w) = 2 + 2 + 7 = 11 \leq 17 = t$, the algorithm adds w to $SUCC^*(u)$.

The 7th processed node is u. Its unprocessed neighbor is v. The algorithm reorders $SUCC^*(u) = \{x, w\}$ first. Because $w(u, x)+b\_time(x) = 6 + 9 = 15$ and $w(u, w) + b\_time(w) = 2 + 7 = 9 = 15 - 3.0\alpha$, the algorithm gives $u_1 = x$, $list_0 = \{x\}$, $list_1 =$
$emptyset$, $list_2 = \{w\}$. Therefore, the reordered sequence of $SUCC^*(u)$ is (x, w).

Now the algorithm starts determining $SUCC(u)$. Because
$\alpha + b\_time(u) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(u, w) + b\_time(w) = 2 + 2 + 7 = 11 \leq 17 = t$, the algorithm adds w to $SUCC(u)$ and $b\_time(u)$ is 11 now. Because $\alpha+b\_time(u) = 2+11 = 13 \leq 17 = t$ and $\alpha+w(u, x)+b\_time(x) = 2+6+9 = 17 \leq 17 = t$, the algorithm adds x to $SUCC(u)$ and $b\_time(u)$ is 17 now. All successor candidates of u are successors of u. Because $PRED^*(u) = \emptyset$, it is obviously $pred(u) = \emptyset$. Because $pred(u) = \emptyset$ and $\alpha + w(v, u) + b\_time(u) = 2 + 5 + 17 = 24 > 17 = t$, the algorithm gives u cannot be a successor candidate of v.

Now the algorithm determines $spare(u)$. Because $arrive(u) + (2 +$

$1)\alpha + w(u,w) + b\_time(w) = 0+6+2+7 = 15 \le 17 = t$, the algorithm gives w can delay α time. Because $arrive(u) + (1+1)\alpha + w(u,x) + b\_time(x) = 0+4+6+9 = 19 > 17 = t$, the algorithm gives x cannot delay α time. The first one successor of u cannot delay α time, the algorithm tells us $spare(u) = arrive(u) + 1\alpha = 0+2 = 2$. Because $spare(u) + \alpha + w(u,v) = 2+2+5 = 9 \le 17 = t$, the algorithm adds u to $PRED^*(v)$.

The 8th processed node is c. Its unprocessed neighbor is v. Because $SUCC^*(c) = \emptyset$ and $PRED^*(c) = \emptyset$, it is obviously $SUCC(c) = \emptyset$, $b\_time(c) = 0$, $pred(c) = \emptyset$ and $spare(c) = 0$. Because $pred(c) = \emptyset$ and $\alpha + w(v,c) + b\_time(c) = 2+8+0 = 10 \le 17 = t$, the algorithm adds c to $SUCC^*(v)$.

The 9th processed node is b. Its unprocessed neighbor is v. Because $SUCC^*(b) = \emptyset$ and $PRED^*(b) = \emptyset$, it is obviously $SUCC(b) = \emptyset$, $b\_time(b) = 0$, $pred(b) = \emptyset$ and $spare(b) = 0$. Because $pred(b) = \emptyset$ and $\alpha + w(v,b) + b\_time(b) = 2+6+0 = 8 \le 17 = t$, the algorithm adds b to $SUCC^*(v)$.

The 10th processed node is v. Its unprocessed neighbor is a. The algorithm reorders $SUCC^*(v) = \{c, b\}$ first. Because $w(v,c) + b\_time(c) = 8+0 = 8$ and $w(v,b) + b\_time(b) = 6+0 = 6 = 8 - 1.0\alpha$, the algorithm gives $u_1 = c$, $list_0 = \{c\}$, $list_1 = \{b\}$, $list_2 = \emptyset$, $list_3 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (c, b).

Now the algorithm starts determining $SUCC(v)$. Because $\alpha + b\_time(v) = 2+0 = 2 \le 17 = t$ and $\alpha + w(v,b) + b\_time(b) =$

$2+6+0 = 8 \leq 17 = t$, the algorithm adds b to $SUCC(v)$ and $b\_time(v)$ is 8 now. Because $\alpha+b\_time(v) = 2+8 = 10 \leq 17 = t$ and $\alpha+w(v,c)+b\_time(c) = 2+8+0 = 10 \leq 17 = t$, the algorithm adds c to $SUCC(v)$ and $b\_time(v)$ is 10 now. All successor candidates of v are successors of v.

Now the algorithm starts determining $pred(v)$. There is only one predecessor candidate of v, u. Because $spare(u)+\alpha+w(u,v)+b\_time(v) = 2+2+5+10 = 19 > 17 = t$, the algorithm gives $pred(v) = \emptyset$. Because $pred(v) = \emptyset$ and $\alpha + w(a,v) + b\_time(v) = 2+3+10 = 15 \leq 17 = t$, the algorithm adds v to $SUCC^*(a)$.

The last processed node is a. Now the algorithm starts determining if $v \in SUCC(a)$. Because $\alpha + b\_time(a) = 2+0 = 2 \leq 17 = t$ and $\alpha+w(a,v)+b\_time(v) = 2+3+10 = 15 \leq 17 = t$, the algorithm adds v to $SUCC(a)$ and $b\_time(a)$ is 15 now. Because $PRED^*(a) = \emptyset$, it is obviously $pred(a) = \emptyset$. The result is there are two broadcast centers, u and a and there is one unused edge, (u,v). Both results tell us we need at least two broadcast centers on T_1, but the location of centers is not necessary unique.

5.2 The Second Example

The tree T_2 is shown in Figure 5.3. Note that the edges in T_2 are shorter edges in G_0. We demostrate the algorithm twice with two processing sequence, $(v, a, c, d, e, f, x, g, b, u, w)$ and $(f, x, d, e, g, b, u, w, c, a, v)$.

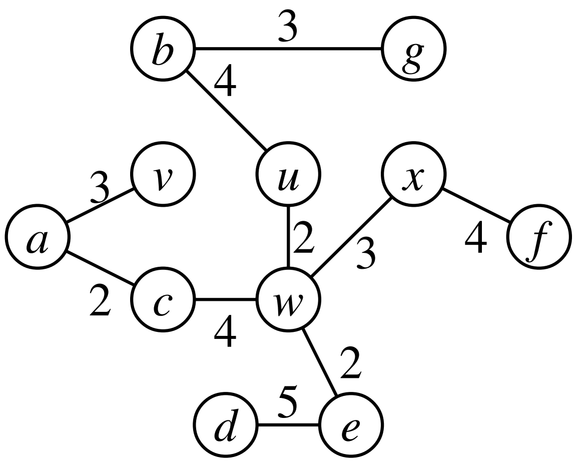

Figure 5.3: Tree T_2.

We show the case $(v, a, c, d, e, f, x, g, b, u, w)$ first. The 1st processed node is v. Its unprocessed neighbor is a. Because $SUCC^*(v) = \emptyset$ and $PRED^*(v) = \emptyset$, it is obviously $SUCC(v) = \emptyset$, $b\_time(v) = 0$, $pred(v) = \emptyset$ and $spare(v) = 0$. Because $pred(v) = \emptyset$ and $\alpha + w(a, v) + b\_time(v) = 2+3+0 = 5 \leq 17 = t$, the algorithm adds v to $SUCC^*(a)$.

The 2nd processed node is a. Its unprocessed neighbor is c. Now the algorithm starts determining if $v \in SUCC(a)$. Because $\alpha + b\_time(a) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(a, v) + b\_time(v) = 2+3+0 = 5 \leq 17 = t$, the algorithm adds v to $SUCC(a)$ and $b\_time(a)$ is 5 now. Because $PRED^*(a) = \emptyset$, it is obviously $pred(a) = \emptyset$. Because $pred(a) = \emptyset$ and $\alpha + w(c, a) + b\_time(a) = 2 + 2 + 5 = 9 \leq 17 = t$, the algorithm adds a to $SUCC^*(c)$.

The 3rd processed node is c. Its unprocessed neighbor is w. Now the algorithm starts determining if $a \in SUCC(c)$. Because $\alpha + b\_time(c) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(c,a) + b\_time(a) = 2+2+5 = 9 \leq 17 = t$, the algorithm adds a to $SUCC(c)$ and $b\_time(c)$ is 9 now. Because $PRED^*(c) = \emptyset$, it is obviously $pred(c) = \emptyset$. Because $pred(c) = \emptyset$ and $\alpha + w(w,a) + b\_time(a) = 2+4+9 = 15 \leq 17 = t$, the algorithm adds a to $SUCC^*(w)$.

The 4th processed node is d. Its unprocessed neighbor is e. Because $SUCC^*(d) = \emptyset$ and $PRED^*(d) = \emptyset$, it is obviously $SUCC(d) = \emptyset$, $b\_time(d) = 0$, $pred(d) = \emptyset$ and $spare(d) = 0$. Because $pred(d) = \emptyset$ and $\alpha + w(e,d) + b\_time(d) = 2+5+0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC^*(e)$.

The 5th processed node is e. Its unprocessed neighbor is w. Now the algorithm starts determining if $d \in SUCC(e)$. Because $\alpha + b\_time(e) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(e,d) + b\_time(d) = 2+5+0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC(e)$ and $b\_time(e)$ is 7 now. Because $PRED^*(e) = \emptyset$, it is obviously $pred(e) = \emptyset$. Because $pred(e) = \emptyset$ and $\alpha + w(w,e) + b\_time(e) = 2+2+7 = 11 \leq 17 = t$, the algorithm adds e to $SUCC^*(w)$.

The 6th processed node is f. Its unprocessed neighbor is x. Because $SUCC^*(f) = \emptyset$ and $PRED^*(f) = \emptyset$, it is obviously $SUCC(f) = \emptyset$, $b\_time(f) = 0$, $pred(f) = \emptyset$ and $spare(f) = 0$. Because $pred(f) = \emptyset$ and $\alpha + w(x,f) + b\_time(f) = 2+4+0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC^*(x)$.

The 7th processed node is x. Its unprocessed neighbor is w. Now the algorithm starts determining if $d \in SUCC(x)$. Because $\alpha + b\_time(x) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(x, f) + b\_time(f) = 2+4+0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC(x)$ and $b\_time(x)$ is 6 now. Because $PRED^*(x) = \emptyset$, it is obviously $pred(x) = \emptyset$. Because $pred(x) = \emptyset$ and $\alpha + w(w, x) + b\_time(x) = 2 + 3 + 6 = 11 \leq 17 = t$, the algorithm adds x to $SUCC^*(w)$.

The 8th processed node is g. Its unprocessed neighbor is b. Because $SUCC^*(g) = \emptyset$ and $PRED^*(g) = \emptyset$, it is obviously $SUCC(g) = \emptyset$, $b\_time(g) = 0$, $pred(g) = \emptyset$ and $spare(g) = 0$. Because $pred(g) = \emptyset$ and $\alpha + w(b, g) + b\_time(g) = 2 + 3 + 0 = 5 \leq 17 = t$, the algorithm adds g to $SUCC^*(b)$.

The 9th processed node is b. Its unprocessed neighbor is u. Now the algorithm starts determining if $v \in SUCC(b)$. Because $\alpha + b\_time(b) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(a, v) + b\_time(g) = 2+3+0 = 5 \leq 17 = t$, the algorithm adds g to $SUCC(b)$ and $b\_time(b)$ is 5 now. Because $PRED^*(b) = \emptyset$, it is obviously $pred(b) = \emptyset$. Because $pred(b) = \emptyset$ and $\alpha + w(u, b) + b\_time(b) = 2 + 4 + 5 = 11 \leq 17 = t$, the algorithm adds b to $SUCC^*(u)$.

The 10th processed node is u. Its unprocessed neighbor is w. Now the algorithm starts determining if $a \in SUCC(u)$. Because $\alpha + b\_time(u) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(c, a) + b\_time(b) = 2+4+5 = 11 \leq 17 = t$, the algorithm adds b to $SUCC(u)$ and $b\_time(u)$ is 9 now. Because $PRED^*(u) = \emptyset$, it is obviously $pred(u) = \emptyset$. Because $pred(u) = \emptyset$ and

$\alpha + w(w, u) + b\_time(b) = 2 + 2 + 11 = 15 \leq 17 = t$, the algorithm adds b to $SUCC^*(w)$.

The last processed node is w. The algorithm reorders $SUCC^*(w) = \{c, e, x, u\}$ first. Because $w(w, c) + b\_time(c) = 4 + 9 = 13$, $w(w, e) + b\_time(e) = 2 + 7 = 9 = 13 - 2.0\alpha$, $w(w, x) + b\_time(x) = 3 + 6 = 9 - 2.0\alpha$ and $w(w, u) + b\_time(u) = 2 + 11 = 13$, the algorithm gives $u_1 = c$, $list_0 = \{c, u\}$, $list_1 = \emptyset$, $list_2 = \{e, x\}$, $list_3 = \emptyset$, $list_4 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (c, u, e, x).

Now the algorithm starts determining $SUCC(w)$. Because $\alpha + b\_time(w) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(w, x) + b\_time(x) = 2 + 3 + 6 = 11 \leq 17 = t$, the algorithm adds x to $SUCC(w)$ and $b\_time(w)$ is 11 now. Because $\alpha + b\_time(w) = 2 + 11 = 13 \leq 17 = t$ and $\alpha + w(w, e) + b\_time(e) = 2 + 2 + 7 = 11 \leq 17 = t$, the algorithm adds e to $SUCC(w)$ and $b\_time(w)$ is 13 now. Because $\alpha + b\_time(w) = 2 + 13 = 15 \leq 17 = t$ and $\alpha + w(w, u) + b\_time(u) = 2 + 2 + 11 = 15 \leq 17 = t$, the algorithm adds u to $SUCC(w)$ and $b\_time(w)$ is 15 now. Because $\alpha + b\_time(c) = 2 + 15 = 17 \leq 17 = t$ and $\alpha + w(w, c) + b\_time(c) = 2 + 4 + 9 = 15 \leq 17 = t$, the algorithm adds c to $SUCC(w)$ and $b\_time(w)$ is 17 now. All successor candidates of w are successors of w. Because $PRED^*(w) = \emptyset$, it is obviously $pred(w) = \emptyset$. The result is that one broadcast center is enough. The location of the center given by the algorithm is w.

Now we run the algorithm again, but with an another processing sequence, $(f, x, d, e, g, b, u, w, c, a, v)$. The 1st processed node is f. Its unprocessed neighbor is x. Because $SUCC^*(f) = \emptyset$ and $PRED^*(f) = \emptyset$, it is obviously $SUCC(f) = \emptyset$, $b\_time(f) = 0$, $pred(f) = \emptyset$ and $spare(f) = 0$. Because $pred(f) = \emptyset$ and $\alpha + w(x, f) + b\_time(f) = 2 + 4 + 0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC^*(x)$.

The 2nd processed node is x. Its unprocessed neighbor is w. Now the algorithm starts determining if $d \in SUCC(x)$. Because $\alpha + b\_time(x) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(x, f) + b\_time(f) = 2+4+0 = 6 \leq 17 = t$, the algorithm adds f to $SUCC(x)$ and $b\_time(x)$ is 6 now. Because $PRED^*(x) = \emptyset$, it is obviously $pred(x) = \emptyset$. Because $pred(x) = \emptyset$ and $\alpha + w(w, x) + b\_time(x) = 2 + 3 + 6 = 11 \leq 17 = t$, the algorithm adds x to $SUCC^*(w)$.

The 3rd processed node is d. Its unprocessed neighbor is e. Because $SUCC^*(d) = \emptyset$ and $PRED^*(d) = \emptyset$, it is obviously $SUCC(d) = \emptyset$, $b\_time(d) = 0$, $pred(d) = \emptyset$ and $spare(d) = 0$. Because $pred(d) = \emptyset$ and $\alpha + w(e, d) + b\_time(d) = 2 + 5 + 0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC^*(e)$.

The 4th processed node is e. Its unprocessed neighbor is w. Now the algorithm starts determining if $d \in SUCC(e)$. Because $\alpha + b\_time(e) = 2+0 = 2 \leq 17 = t$ and $\alpha + w(e, d) + b\_time(d) = 2+5+0 = 7 \leq 17 = t$, the algorithm adds d to $SUCC(e)$ and $b\_time(e)$ is 7 now. Because $PRED^*(e) = \emptyset$, it is obviously $pred(e) = \emptyset$. Because $pred(e) = \emptyset$ and $\alpha + w(w, e) + b\_time(e) = 2 + 2 + 7 = 11 \leq 17 = t$, the algorithm adds e to $SUCC^*(w)$.

The 5th processed node is g. Its unprocessed neighbor is b. Because

$SUCC^*(g) = \emptyset$ and $PRED^*(g) = \emptyset$, it is obviously $SUCC(g) = \emptyset$, $b\_time(g) = 0$, $pred(g) = \emptyset$ and $spare(g) = 0$. Because $pred(g) = \emptyset$ and $\alpha + w(b, g) + b\_time(g) = 2 + 3 + 0 = 5 \leq 17 = t$, the algorithm adds g to $SUCC^*(b)$.

The 6th processed node is b. Its unprocessed neighbor is u. Now the algorithm starts determining if $v \in SUCC(b)$. Because $\alpha + b\_time(b) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(a, v) + b\_time(g) = 2 + 3 + 0 = 5 \leq 17 = t$, the algorithm adds g to $SUCC(b)$ and $b\_time(b)$ is 5 now. Because $PRED^*(b) = \emptyset$, it is obviously $pred(b) = \emptyset$. Because $pred(b) = \emptyset$ and $\alpha + w(u, b) + b\_time(b) = 2 + 4 + 5 = 11 \leq 17 = t$, the algorithm adds b to $SUCC^*(u)$.

The 7th processed node is u. Its unprocessed neighbor is w. Now the algorithm starts determining if $a \in SUCC(u)$. Because $\alpha + b\_time(u) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(c, a) + b\_time(b) = 2 + 4 + 5 = 11 \leq 17 = t$, the algorithm adds b to $SUCC(u)$ and $b\_time(u)$ is 9 now. Because $PRED^*(u) = \emptyset$, it is obviously $pred(u) = \emptyset$. Because $pred(u) = \emptyset$ and $\alpha + w(w, u) + b\_time(b) = 2 + 2 + 11 = 15 \leq 17 = t$, the algorithm adds b to $SUCC^*(w)$.

The 8th processed node is w. Its unprocessed neighbor is c. The algorithm reorders $SUCC^*(w) = \{x, e, u\}$ first. Because $w(w, x) + b\_time(x) = 3 + 6 = 9 - 2.0\alpha$, $w(w, e) + b\_time(e) = 2 + 7 = 9 = 13 - 2.0\alpha$ and $w(w, u) + b\_time(u) = 2 + 11 = 13$, the algorithm gives $u_1 = u$, $list_0 = \{u\}$, $list_1 = \emptyset$, $list_2 = \{x, e\}$, $list_3 = \emptyset$. Therefore, the reordered sequence of $SUCC^*(v)$ is (u, x, e).

Now the algorithm starts determining $SUCC(w)$. Because $\alpha + b\_time(w) = 2 + 0 = 2 \leq 17 = t$ and $\alpha + w(w, e) + b\_time(e) = 2 + 2 + 7 = 11 \leq 17 = t$, the algorithm adds e to $SUCC(w)$ and $b\_time(w)$ is 11 now. Because $\alpha + b\_time(w) = 2 + 11 = 13 \leq 17 = t$ and $\alpha + w(w, x) + b\_time(x) = 2 + 3 + 6 = 11 \leq 17 = t$, the algorithm adds x to $SUCC(w)$ and $b\_time(w)$ is 13 now. Because $\alpha + b\_time(w) = 2 + 13 = 15 \leq 17 = t$ and $\alpha + w(w, u) + b\_time(u) = 2 + 2 + 11 = 15 \leq 17 = t$, the algorithm adds u to $SUCC(w)$ and $b\_time(w)$ is 15 now. All successor candidates of w are successors of w.

Because $PRED^*(w) = \emptyset$, it is obviously $pred(w) = \emptyset$. Because $\alpha + w(c, w) + b\_time(w) = 2 + 4 + 15 = 21 > 17 = t$, the algorithm gives w cannot be a successor candidate of c. Now the algorithm determines $spare(w)$. Because $arrive(w) + (3 + 1)\alpha + w(w, e) + b\_time(e) = 0 + 8 + 2 + 7 = 15 \leq 17 = t$, the algorithm gives e can delay α time. Because $arrive(w) + (2 + 1)\alpha + w(w, x) + b\_time(x) = 0 + 6 + 3 + 6 = 13 \leq 17 = t$, the algorithm gives x can delay α time. Because $arrive(w) + (1 + 1)\alpha + w(w, u) + b\_time(u) = 0 + 4 + 2 + 11 = 17 \leq 17 = t$, the algorithm gives u can delay α time. All successors of w can delay α time, the algorithm tells us $spare(v) = 0$. Because $spare(w) + \alpha + w(w, c) = 0 + 2 + 4 = 6 \leq 17 = t$, the algorithm adds w to $PRED^*(c)$.

The 9th processed node is c. Its unprocessed neighbor is a. Because $SUCC^*(c) = \emptyset$, it is obviously $SUCC(c) = \emptyset$ and $b\_time(c) = 0$. There is only one predecessor candidate of c, w. Because $spare(w) + \alpha + w(w, c) + b\_time(c) = 0 + 2 + 5 + 0 = 7 \leq 17 = t$, the algorithm gives $pred(c) = w$, $spare(c) = arrive(c) = 6$ and that c cannot apply for being a successor candidate of a. Because $spare(c) + \alpha + w(c, a) = 6 + 2 + 2 = 10 \leq 17 = t$, the algorithm adds c to $PRED^*(a)$.

The 10th processed node is a. Its unprocessed neighbor is v. Because $SUCC^*(a) = \emptyset$, it is obviously $SUCC(a) = \emptyset$ and $b\_time(a) = 0$. There is only one predecessor candidate of a, c. Because $spare(c) + \alpha + w(c, a) + b\_time(a) = 6 + 2 + 2 + 0 = 10 \leq 17 = t$, the algorithm gives $pred(a) = c$, $spare(a) = arrive(a) = 10$ and that a cannot apply for being a successor candidate of v. Because $spare(a) + \alpha + w(a, v) = 10 + 2 + 3 = 15 \leq 17 = t$, the algorithm adds a to $PRED^*(v)$.

The last processed node is v. Because $SUCC^*(v) = \emptyset$, it is obviously $SUCC(v) = \emptyset$ and $b\_time(v) = 0$. There is only one predecessor candidate of v, a. Because $spare(a)+\alpha+w(a, v)+b\_time(v) = 10+2+3+0 = 15 \leq 17 = t$, the algorithm gives $pred(v) = a$.

The results with the processing sequence $(f, x, d, e, g, b, u, w, c, a, v)$ and $(f, x, d, e, g, b, u, w, c, a, v)$ are the same. Both of them say one broadcast center is enough and the location of the center is w.

Chapter 6

Conclusion

We made an improvement for finding the location of broadcast centers and the broadcast sequences on any tree T, $OPT(T, t)$, such that broadcasting can be done within the time constraint t, which is introduced in [20]. We extended this problem from uniform telephone model to heterogeneous postal model and improved the time complexity to $O(n)$.

Although broadcast problem has several decades of history, There are still many open problems; many of them are shown in Table 6.1, where "?" means open problems and "$\times$" means not defined.

Table 6.1: Broadcast problems in uniform telephone model and heterogeneous postal model

	Uniform Telephone	Heterogeneous Postal
1-center	$O(n)$ [5]	$O(n)$ [17]
1-median	$O(n)$	$O(n)$ [22]
p-center	?	?
p-median	?	?
k-broadcasting	$O(n)$ [27]	?
uncertainty	$\times$	$O(nlog(n)loglog(n))$ [28]
Centers with Time Constraints	$O(n)$	$O(n)$
approximation	ratio : $O(\frac{log(n)}{loglog(n)})$	?
	time : $O(\|V\|\|E\|)$ [8]	
heuristic	$O(\|E\|)$ [10]	?

How to extend the results from uniform telephone model to heterogeneous postal model or more complicated environment, from trees to more general structures, and so on, are researchable topics.

Bibliography

[1] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, "Message multicasting in heterogeneous networks," *SIAM Journal on Computing*, vol. 30, no. 2, pp. 347–358, 2000.

[2] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the postal model for message-passing systems," *Mathematical Systems Theory*, vol. 27, no. 5, pp. 431–452, 1994.

[3] A. Bar-Noy and S. Kipnis, "Multiple message broadcasting in the postal model," *Networks*, vol. 29, no. 1, pp. 1–10, 1997.

[4] J.-m. Koh and D.-w. Tcha, "Information dissemination in trees with nonuniform edge transmission times," *IEEE Transactions on Computers*, vol. 40, no. 10, pp. 1174–1177, 1991.

[5] P. J. Slater, E. J. Cockayne, and S. T. Hedetniemi, "Information dissemination in trees," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 692–701, 1981.

[6] R. Ravi, "Rapid rumor ramification: approximating the minimum broadcast time," in *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, 1994, pp. 202–213.

[7] G. Kortsarz and D. Peleg, "Approximation algorithms for minimum-time broadcast," *SIAM Journal on Discrete Mathematics*, vol. 8, no. 3, pp. 401–427, 1995.

[8] M. Elkin and G. Kortsarz, "Sublogarithmic approximation for telephone multicast: path out of jungle," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete algorithms*, 2003, pp. 76–85.

[9] R. Beier and J. F. Sibeyn, *A Powerful Heuristic for Telephone Gossiping*. Citeseer, 2000.

[10] H. A. Harutyunyan and B. Shao, "An efficient heuristic for broadcasting in networks," *Journal of Parallel and Distributed Computing*, vol. 66, no. 1, pp. 68–76, 2006.

[11] P. Scheuermann and G. Wu, "Heuristic algorithms for broadcasting in point-to-point computer networks," *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 804–811, 1984.

[12] H. A. Harutyunyan and E. Maraachlian, "On broadcasting in unicyclic graphs," *Journal of Combinatorial Optimization*, vol. 16, no. 3, pp. 307–322, 2008.

[13] H. A. Harutyunyan, G. Laza, and E. Maraachlian, "Broadcasting in necklace graphs," in *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*, 2009, pp. 253–256.

[14] H. A. Harutyunyan and E. Maraachlian, "Broadcasting in fully connected trees," in *Proceedings of the 15th IEEE International Conference on Parallel and Distributed Systems*, 2009, pp. 740–745.

[15] P. Bhabak and H. A. Harutyunyan, "Broadcast problem in hypercube of trees," *Lecture Notes in Computer Science : Frontiers in Algorithmics*, vol. 8497, pp. 1–12, 2014.

[16] E. Maraachlian, "Optimal broadcasting in treelike graphs," PhD thesis, Concordia University, 2010.

[17] Y.-H. Su, C.-C. Lin, and D. T. Lee, “Broadcasting in heterogeneous tree networks,” *Lecture Notes in Computer Science : Computing and Combinatorics*, vol. 6196, pp. 368–377, 2010.

[18] A. Farley, S. Hedetniemi, S. Mitchell, and A. Proskurowski, “Minimum broadcast graphs,” *Discrete Mathematics*, vol. 25, no. 2, pp. 189–193, 1979.

[19] A. M. Farley, “Broadcast time in communication networks,” *SIAM Journal on Applied Mathematics*, vol. 39, no. 2, pp. 385–390, 1980.

[20] A. M. Farley and A. Proskurowski, “Broadcasting in trees with multiple originators,” *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 4, pp. 381–386, 1981.

[21] A. L. Liestman and J. G. Peters, “Broadcast networks of bounded degree,” *SIAM Journal on Discrete Mathematics*, vol. 1, no. 4, pp. 531–540, 1988.

[22] C.-H. Tsou, G.-H. Chen, H.-I. Yu, and C.-C. Lin, “The broadcast median problem in heterogeneous postal model,” *Journal of Combinatorial Optimization*, vol. 25, no. 4, pp. 602–616, 2013.

[23] P. Fraigniaud and E. Lazard, “Methods and problems of communication in usual networks,” *Discrete Applied Mathematics*, vol. 53, no. 1, pp. 79–133, 1994.

[24] H. Grigoryan, “Problems related to broadcasting in graphs,” PhD thesis, Concordia University, 2013.

[25] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, “A survey of gossiping and broadcasting in communication networks,” *Networks*, vol. 18, no. 4, pp. 319–349, 1988.

[26] J. Hromkovič, R. Klasing, B. Monien, and R. Peine, “Dissemination of information in interconnection networks (broadcasting & gossiping),” *Combinatorial Network Theory*, pp. 125–212, 1996.

[27] H. A. Harutyunyan, A. L. Liestman, and B. Shao, “A linear algorithm for finding the k-broadcast center of a tree,” *Networks*, vol. 53, no. 3, pp. 287–292, 2009.

[28] C.-H. Tsou, G.-H. Chen, and C.-C. Lin, “Broadcasting in heterogeneous tree networks with uncertainty,” *Lecture Notes in Computer Science : Algorithms and Computation*, vol. 7074, pp. 200–209, 2011.