MST 재구성 분산 알고리즘

박 정 호† 민 준 영††

요 약

본 논문은 최소목(Munimum-weight Spanning Tree,MST)에 있어서 네트워크의 링크중 몇개가 삭제(또는 파괴) 또는 추가(또는 회복)되었을때, MST를 재구성하는 분산 알고리즘을 제안한다. 본 논문에서 제안한 알고리즘의 메세지 복잡도는 $O(m+n\ log(t+f))$이고, 이상시간 복잡도는 $O(n+n\ log(t+f))$가 되며, 여기서 n은 네트워크의 프로세서의 수이고 $t(resp.\ f)$는 추가되는 링크의 수(resp. 이전 MST의 삭제된 링크의 수)이다. 그래서 네트워크의 형태가 변형이 된 다음에 $f=0$이고 $m=e$일 경우에는 $m=t+n$이 된다. 또한 본 논문의 마지막 부분에서는 링크의 추가, 삭제와 마찬가지로 프로세서의 추가, 삭제되었을 경우의 알고리즘도 제안한다.

Distributed Algorithm for Updating Minimum-Weight Spanning Tree Problem

Jung Ho Park† and Joon Young Min††

ABSTRACT

This paper considers the Updating Minimum-weight Spanning Tree Problem(UMP), that is, the problem to update the Minimum-weight Spanning Tree(MST) in response to topology change of the network. This paper proposes the algorithm which reconstructs the MST after several links deleted and added. Its message complexity and its ideal-time complexity are $O(m+n\ log(t+f))$ and $O(n+n\ log(t+f))$ respectively, where n is the number of processors in the network, $t(resp.\ f)$ is the number of added links (resp. the number of deleted links of the old MST), and $m=t+n$ if $f=0$, $m=e$ (i.e. the number of links in the network after the topology change) otherwise. Moreover the last part of this paper touches on the algorithm which deals with deletion and addition of processors as well as links.

1. Introduction

Consider an asynchronous network where a cost or weight (representing, for example, usage fee or delay) is associated with every link. For the purpose of disseminating information in the network, it is advantageous to broadcast it over the Minimum-weight Spanning Tree(MST), since information will be deliverd to every processor with small communication cost. Many distributed algorithms have been proposed for constructing the MST. Most of them consider the initial problem to construct the MST, that is, construction of the MST starts from scratch. In a real network, topology of a network often changes because of processor (or link) deletion (e.g. failure) and addition(i.e. recovery). When network N changes to N', the MST T of N may not be the MST of N'. For example, when a link of T is deleted by the topology change, T is devided into two trees, and the new MST must be reconstructed to

† 종신회원 : 선문대학교 전자계산학과 교수
†† 정 회 원 : 상지대학교 병설전문대학 전자계산과 교수
논문접수 : 1994년 1월 15일, 심사완료 : 1994년 5월 14일

broadcast or collect messages efficiently. This paper considers the Updating MST problem (UMP), that is, the problem to update the MST in response to topology change. In UMP, it is assumed that topology change does not occur during the execution of an algorithm. In the following, N(resp. N') represents a network before topology change(resp. a network after topology change), and T (resp. T') represents the MST of N(resp. N')

It is obvious that UMP can be solved by the known algorithms which construct the MST from scratch. For the problem to construct the MST from scartch, Gallager et al. presented a distributed algorithm and showed that its message complexity is optimal within a constant factor[3]. Moreover, Awerbuch improved the algorithm and obtained an optimal algorithm in both the message complexity and the ideal-time complexity[1]. By applying Awerbuch's algorithm to UMP, UMP can solved in $O(e+n \log n)$ messages and $O(n)$ (ideal) time, where n (resp. e) is the number of processors (resp. links) in the network N' However, most part of the new MST T' may coincide with the old MST T, and it is natural to assume that each processor knows the old solution, that is, each processor knows which incident links are those of the old MST T. This raises a question: *How efficiently UMP can be solved by utilizing the old solution?* This is an interesting subject of study.

In this paper, we propose the algorithms which reconstructs MST T' after several links are deleted ans added. Its message complexity and its ideal-time complexity are $O(m+n \log (t+f))$ and $O(n+n \log (t+f))$ respectively, where t (resp. f) is the number of added links (resp. the number of deleted links of the old MST T) and where $m=n+t$ if $f=0$, $m=e$ otherwise. Thus, our algorithm is superior to Awerbush's algorithm in the message complexity. In the last part of this paper, we will touch on the algorithm which deals with deletion and addition of processors as well as links.

Up to date, a number of algorithms have been proposed for UMP. However, they consider either link deletion or link addition, and no algorithm considers the situation where both deletion and addition of links occur. Our algorithm can be applied to

(a) the case that only link addition occurs, and

(b) the case that only link deletion occurs.

In the rest of this section, we compare our algorithm with the previous results for these two cases (see Table 1).

(a) The case that only link addition occurs

For UMP after t links addition, Tsin pre

〈Table 1〉 Complexities of algorithms for UMP

	j Links Deletion		t Links Addition	
	Paper [2]	Our result	Paper [5]	Our result
M.C	$O(f^2p)$	$O(e+n \log f)$	$O(nt+t^2)$	$O(n+t+n \log t)$
B.C.	$O(f^2pe \log n)$	$O(e \log n+n \log f \log n)$	$O(nt \log n+t^2 \log n)$	$O(n \log t \log n+(n+t) \log n)$
I.C.	$O(f2p)$	$O(n+n \log f)$	$O(nt+t)$	$O(n+n \log t)$
S.C.	$O(ne \log n)$	$O(e)$	$O(e)$	$O(e)$

M.C. : Message Complexity.
B.C. : Bit Complexity.
I.C. : Ideal-Time Complexity.
S.C. : Space Complexity. (Total storage in the whole network)
n : the number of processors.
e : the number of links in the network after topology change.
f : the number of deleted links of the old MST T.
p : the length of the longest cycle in the network before topology change.

sented an algorithm[5]. Its message complexity and its ideal−time complexity are both $O(n+t+t^2)$. For this case, the message complexity and the ideal−time complexity of our algorithm are $O(n+t+n \log t)$ and $O(n+n \log t)$, respectively. Therefore, our algorithm are $O(n+t+n \log t)$ and $O(n+n \log t)$, respectively.

Therefore, our algorithm is superior to Tsin's algorithm in both the message complexity and ideal−time complexity.

(b) The case that only link deletion occurs.

For the case of j links deletion, Cimet *et al.* presented an algorithm (hereinafter denoted as CK)[2]. Its message complexity and its ideal−time complexity are both $O(f^2p)$, where p is the length of the longest cycle in a network N. For this case, the message complexity and the ideal−time complexity of our algorithm are respectively $O(e+n \log f)$ and $O(n+n \log f)$. (Recall that f represents the number of deleted links of the old MST T, while j represents the number deleted links containing the non−tree links. This implies that $f \leq j$ holds.) It depends on the parameters n, e, f, p and j whether or not our algorithm is better than CK in the message complexity and the ideal−time complexity. The bit complexity of CK is $O(f^2pe \log n)$ since the length of each message used in CK is $O(e \log n)$ bits. On the other hand, the bit complexity of our algorithm is $O(e \log n+n \log f \log n)$ since the length of each message used in our algorithm is $O(\log n)$ bits. Therefore, our algorithm is better, in the bit complexity, than CK. Furthermore, CK utilizes in the auxiliary information, called a replacement set, in addition to the old solution. To store the replacement set, CK needs the storage of $O(ne \log n)$ bits in the whole network. On the other hand, the space cpmplexity (the total storage in the whole network) of our algorithm is $O(e)$ since our algorithm needs no auxiliary information except for the old solution. Therefore, our algorithm is superior to CK in the space complexity.

2. Model

Our model is standard one, that is, (A1) through (A5) are assumed.

(A1) The processors are connected by bidirectional communication links and the processors communicate only by passing messages along the links. A nework N is denoted as N =(P,L) where P is the set of processors ans L is the set of links, with $|P| = n$ and $|L| = e$.

(A2) The network is asynchronous, that is, the time to transmit a message along a link is finite but unpredictable.

(A3) Each processor u has a unique *identity number* (i.e. processor number) $ID(u)$, and every identity number is represented in $O(\log n)$ bits. Each link (u, v) has a unique weight $W(u, v)$, and every weight is represented in $O(\log n)$ bits.

(A4) The processors all perform the same program. The program executed in each processor includes (a) its internal operations, (b) send operations to send messages via its ports, and (c) receive operations to receive messages from its ports. (Each processor can distinguish its ports each other.)

(A5) Any non−empty subset of processors may start the algorithm sponteneously, and each of other processors starts the algorithm

when it receives a message.

MST Updating Problem (MUP)

Let N be an arbitrary network and assume that the MST T of N is already constructed, that is, each processor knows which incident links are those of T. The *MST Updating Problem*(MUP) is the problem to reconstruct the MST after N changes to N' by adding several links to N and deleting several links from N. In order to solve UMP efficiently, some auxiliary information (e.g. the replacement set [2]) may be utilized. In this case, the auxiliary information needs to be updated so as to correspond to N'.

Note that UMP defined above dose not cover the topology change where deletion and addition of processors occur. This paper mainly considers only deletion and addition and links, and touches on deletion and addition of processors in the last part.

In this paper, MUP is considerd under the following assumption.

(A6) The topology change does not occur during the execution of an algorithm.

Throughout this paper, N=(P, L) (resp. N' = (P, L')) represents a network before topology change (resp. a network after topology change), and T=(P, L_T) (resp. T'=(P, L_T') represents the MST of N (resp. N'). Moreover, L_a (resp. L_d) represents the set of added links (resp. the set of deleted links of T), and let t= $|L_a|$ and f= $|L_d|$. Note that L_d does not contain the deleted links which are not the links of T. We will pay only a little attention to the deleted links not included in L_T.

Measures of efficiency of algorithms

In this paper, to measure the efficiency of an algorithm, we use the following measures.

(A) *Message complexity* : The (worst case) message complexity is the maximum total number of messages transmitted during any execution of the algorithm.

(B) *Bit complexity* : The (worst case) bit complexity is the maximum total number of bir transmitted during any execution of the algorithm.

(C) *Ideal time complexity* : The (worst case) ideal time complexity is the maximum number of time units from start to the completion of the algorithm, assuming that the propagation delay of every link is one time unit of some global clock. This assumption is used only for purpose of evaluating the ideal time complexity, since the network is asynchronous.

(D) *Space complexity* : The (worst case) space complexity is the total amount of storage of all processors in the whole network.

3. The idea of our algorithm

In this section, we describe the idea of our algorithm to update MST after several links are deleted ans added.

3.1 Application of Gallager's method

Our algorithm is partly based on the technique Gallager et al. proposed[3]. At first, we shortly illustrate the Gallager's method. Gallager's Method

Any connected subgraph of the MST is referred to as a *fragment*. Define a link as an *outgoing link* of a fragment if one adjacent processor is in the fragment and the other is not. Gallager's algorithm, denoted as GHS

hereinafter, starts with each individual processor as a fragment, that is, there exist n fragments each of which consist of one processor in the beginning of GHS. GHS enlarges each fragment by finding its minimum−weight outgoing link and combining the fragment with the fragment at the other end of link, then enlarges each of the new fragments again in the same way, and so forth, GHS repeats the above procedure until there exists only enforces a balanced growth of fragments by utilizing the level of each fragment, which reflects the size(i.e. the number of processors) of the fragment.

Applying Gallager's method to UMP

When topology change occurs, most part of the new MST T' may coincide with the old MST T. For example, consider the topology change where *several links of T are deleted and no link is added.* By the topology change, the old MST T is divided into several connected components, and then each connected component is a fragment of the new MST T'. This fact implies that an algorithm for UMP can start with enlarged fragments, that is, each of the fragments may contain more than one processor. By the algorithm exactly similar to GHS, we can efficiently construct MST from the initial configulation. We denote the algorithm as PMHT (see reference [4], initial character of author).

[Property 1][4] If the algorithm PMHT starts in the situation that there are r fragments, then it constructs the MST with the message complexity $O(n \log r+e)$ and the ideal−time complexity $O(n \log r+n)$.

When f links of the old MST T are deleted, T is divided into $f+1$ connected component. Therefore from Property 1, the algorithm PMHT reconstructs MST with the message complexity $O(n \log f+e)$ and the ideal−time complexity $O(n \log f+n)$, after f links of the old MST T are deleted (and no like is added).

Consider the case where *both link addition and link deletion occur.* In this case, the old MST T is divided into several connected by the deleted links. But all connected components of $T-L_d(P, L_T-E_d)$)may not be the new MST T' since there exist added links. Therefore, the algorithm PMHT can not be simply applied to this case.

How can we solve UMP after several links are added and deleted? We can pick up an idea in the algorithm Tsin proposed[5].

3.2 Tsin's method

Tsin[5] proprsed an algorithm to reconstruct MST after *several links are added and no link is deleted.* We briefly describe the algorithm.

Removal of the maximum−weight link in a cycle

When a link (u, v) is added, (u, v) and the $u-v$ path in T from a cycle. In this case, the new MST T' is obtained by removing the maximum−weight link in the cycle from $T+\{(u, v)\}$ $(=(P, L_T \cup \{(u, v)\}))$. When several links are added, $T+L_a(=(P, L_T \cup L_a))$ has several cycles. Tsin's algorithm reconstructs the new MST T' as follows.

(1) Let T_{work} be $T+L_a$.

(2) Repeat the followings until there exists no cycle in T_{work}.

(i) Find a cycle of T_{work}, and find the maximum−weight link l in the cycle.

(ii) Let T_{work} be $T_{work}-\{l\}$.

(3) T_{work} is the new MST T'. (Note that T_{work} has no cycle in (3).)

Tsin's algorithm cannot efficiently reconstruct the new MST T', because it sequentially processes the cycles in $T+L_a$. Therefore the authors tried to process the cycles concurrently, but it seems to be difficult to do so in the reasonable cost (the number of messages). In order to solve this difficulty, we introduce new idea.

3.3 Partition links (Our new idea)

The algorithm PMHT is an efficient algorithm to reconstruct T' from the initial configulation where there exist several enlarged fragments. In order to apply PMHT to the case where *both link addition and link deletion occur* (the set of partition links is denoted by L_p throughout this paper). The definition of the partition links is described later. The key point is that the partition links satisfy the following properties.

[Property 2] The partition links are links of the old MST T, that is, $L_T \supseteq L_p$ holds.

[Property 3] Let l be any link of the old MST T. If l is deleted by the topology change, l is a partition link, that is $L_p \supseteq L_d$ holds.

[Property 4] Each connected component of $T-L_p$ is a fragment of the new MST T', that is, L_p contains the maximum weight link in each cycle of $T+L_a$.

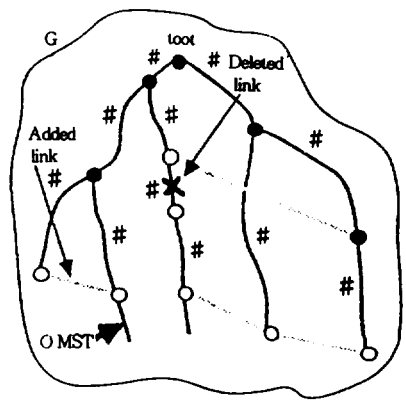
[Property 5] The number of connected components in $T-L_p$ is $O(t+f)$.

[Property 6] The partition links can be found efficiently.

Our algorithm first finds the partition links and changes them to non-tree links, then all tree links (except for partition links) of the old MST T form several connected components of the old MST T. Property 4 implies that each connected component is a fragment of the new MST T'. Therefore in the latter half of our new algorithm, PMHT is applied to construct the new MST T'. Property 5 is needed to bound the message complexity of our algorithm, because the message complexity of PMHT depends on the number of fragments in the initial configulation. In order to utilize the partition links efficiently, Property 6 is needed.

The partition links are defined as follows. For simplicity, the MST is regarded as a rooted tree in what follows.

[Definition] An *adjacent processor* is a processor incident to an added link or a deleted link. Define processor u as a branch processor, if there exist two processors in u's sons such that each of them has an adjacent processor in its descendents. In other words, u is a branch processor if u is the nearest common ancestor of some two adjacent processors. A marked processor is a processor that is an adjacent processor or a branch proces-



(Fig. 1) Adjacent processors, branch processors and primary paths.

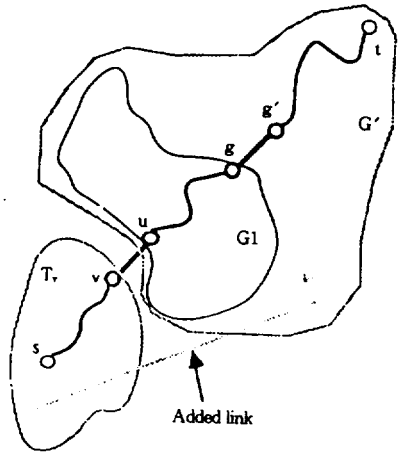
sor. (We use the term a marked processor when there is no need for distinguishing a branch processor from an adjacent processor.) For any marked processors u and v, the $u-v$ path in T is called a *primary path* if no other marked processor exists on the $u-v$ path. Define a link as a partition link, if it is the maximum−weight link in a primary path (Figure 1).

From the definition, it is clear that the partition links defined above satisfy Property 2 and Property 3. The following two lemmas show that Property 4 and Property 5 hold. Property 6 is examined later.

[Lemma 1] Each connected component of T $-L_p$ is a fragment of the new MST T'.

(proof) Assume that there is a connected component in $T-L_p$ which is not a fragment of the new MST T'. Then, it contains a link (u, v) such that it is not a link of T', that is, $(u, v) \notin L_T'$. Let v be a son u in the old MST T. By removing (u, v) from T, T is divided in to two trees, the subtree with the root v (denoted by T_v) and the other part T $-T_v$ (denoted by G').

(Case 1) There exists an added link between Tv and G', that is, an added link which connects a processor in T_v and a processor in G'(Figure 2).



(Fig. 2) Network in the proof of Lemma 1.

There is marked processor which is an ancestor of u. Let t be the nearest marked processor to u among ancestors of u. Similarly, let s be the nearest marked processor to v among descendants of v. From the definition of the partition link, the maximum weight link in the $t-s$ path is the partition link. Let (g, g') be the maximum weight link and g be a son of g'. Since (u, v) is a link in $T-L_p$, (u, v) is not a partition link, that is, $(u, v) \neq (g, g')$ holds. Without loss of generality, we can assume that (g, g') is a link on the $u-t$ path. When (u, v) and (g, g') are removed from T, T is devided into three connected components; the connected component containing u and g (denoted by G1), the connected component containing g' and subtree with a root v. From the definition of t, there exist no marked processor in G1, that is, there is no added link connecting to processor in G1. This fact implies that $W(z, z') > W(u, v)$ or $W(z, z') > W(g, g')$ holds for any link (z, z') between G1 and $T-G1$. Since (g, g') is the maximum weight link on the $t-s$ path, $W(g, g') > W(u, v)$ holds, and $W(z, z') > W(u, v)$ holds (Claim 1). When(u, v) is added to the new MST T', (u, v) and the $u-v$ path in T' from cycle. On the cycle, there exists a link (p, q) between G1 and T $-$G1 except for (u, v). Since $W(p, q) > W(u, v)$ holds from Claim 1, $(T'-\{(p-q)\})+\{(u, v)\}$ is a spanning tree and its total sum of weight is smaller than that of T'. This contradicts the fact that T' is the new MST of N'.

(Case 2) There exists no added link be-

tween T_v and G'.

Since (u, v) is a link of old MST of N, $W(p, q) > W(u, v)$ holds for any link (p, q) between Tv and G' (Claim 2). When (u, v) is added to T', (u, v) and the $u-v$ path in T' from a cycle. On the cycle, there exists a link (p, q) between T_v and G' except for (u, v). Since $W(p, q) > W(u, v)$ holds from 2 Claim 2. $(T'-\{(p, q)\})+\{(u, v)\}$ is a spanning tree and its total sum of weight is smaller than that of T'. That contradicts the fact that T' is the new MST of N'.

[Lemma 2] The number of connected components in $T-L_p$ is $O(t+f)$.

(proof) Consider the graph $G'=(V', E')$ where

$V'=\{u \mid u$ is a connected component in $T-L_p\}$, and

$E'=\{(u, v) \mid$ There exists a partition link (s, t) such that s(resp. t) bolongs to the connected component u (resp. v) in $T-L_p\}$.

Clearly, G' is a tree. From the definition of partition link, (a) each connected component corresponding to a leaf of G' contains one adjacent processor, and (b) each connected component corresponding to an internal node of G' has at least two sons (Claim 3). From Claim 3, we can show that the number of the connected components in $T-L_p$ (i.e. the number of vertices in G') is $O(t+f)$.

Consider the case that no link of the old MST T is deleted, that is, $L_d=\phi$ holds. Then, it is obvious that the following Lemma 3 holds. This observation saves messages in this case. It follows from Lemma 3 that we can ignore all links which belong to N but not belong to T. In other words, no message is sent along those links in our algorithm when no link of T is deleted.

[Lemma 3] The new MST T' of N' coincides with the MST of $T+L_a$, if there exists no deleted link of T.

4. The outline of our algorithm

In this section, we describe the outline of our algorithm. Our algorithm consists of four phases.

(Phase 1) Check where there exists a deleted link of T' or not. If there exists no deleted link of T (i.e. $L_d=\phi$), ignore all links in $L-L_T$ (links which belong to N but not belong to T) in the following three phases.

(Phase 2) Elect a leader in each connected component of $T-L_d$. In next phase, the connected component of $T-L_d$ is regarded as the rooted tree whose root is the leader elected in this phase.

(Phase 3) Find the partition links and change them into non-tree links.

In this phase, the partition links are found in each connected component of $T-L_d$ as follows.

(3.1) Find the marked processors (the adjacent processors and the branch processors) in the bottom-up fashion from leaves of the connected component of $T-L_d$. This step proceeds as follows.

(i) Each leaf of the connected component of $T-L_d$ decides where it is an adjacent processor or not, and then sends a message to its parent to inform whether the leaf is an adjacent processor or not.

(ii) When each internal-processor receives messages from all sons, it decides whether it is marked processor or not, that is, it decides to be a marked processor if and only if (a) it is an adjacent proces-

sor or (b) there exist two sons which inform that there exist adjacent processors in their descendants. Then, it sends a message to its parent to inform whether there exists an adjacent processors in its descendants or not.

(3.2) Every marked internal−processor is the upper end of a primary path, if it receives the message telling that its son has an adjacent processor in its descendants. In order to find the partition link in the primary path, the marked internal−processor sends a message to every son that informs there exists an adjacent processor in its descendants. The message is transfered to a marked processor, which is the other end of the primary path, and the message carries the maximum weight of the link which is has ever traversed. When the message reached the other end of the primary path, the processor finds the weight of the partition link of the primary path, and the message is forwarded upward to the processors incident to the partition link.

(Phase 4) Apply the algorithm PMHT to the network N' with the initial configuration where each connected component of $T-L_p$ is a fragment of the new MST T'.

5. Correctness and complexities of our algorithm

It is obvious that Phases 1,2 and 3 terminate wihtin a finite time, and the partition links are correctly found in Phase 3. From Lemma 1, $T-L_p$ is a fragment of the new MST T'. Thus, by applying PMHT, the new MST T' of N' can be reconstructed within a finite time in Phase 4.

[Theorem 1] The message complexity of our algorithm is $O(n\log(t+f)+ m)$ and the bit complexity is $O(n \log(t+f)\log n+m\log n)$, where $m=n+t$ if $f=0$, $m=e$ otherwise.

(Proof) The message complexity of Phase 1 is $O(n)$, and the message complexity of Phase 2 is $O(n)$ if $f=0$, $O(e)$ otherwise. In Phase 3, a constant number of messages are sent through each remaining link of T. Thus, the message complexity of Phase 3 is $O(n)$. From Lemma 2, there exist $O(t+f)$ fragments in the beginning of Phase 4. Therefore, it follows from Property 1 that the message complexity of Phase 4 is $O(n \log(t+f)+e)$, if $f\neq 0$. If $f=0$, from Lemma 3 we ignore all links in $L-L_T$, and no message is sent along these links. Thus the message complexity is $O(n \log t+ (n+t))$ if $f=0$..

Each message of our algorithm is $O(\log n)$ bits long, hence, the bit complexity of our algorithm is $O(n \log(t+f)$ log $n+m \log n)$, where $m=n+t$ if $f=0$, $m=e$ otherwise.

[Theorem2] The ideal−time complexity of our algorithm is $O(n \log(t+f)+ n)$.

(Proof) The ideal−time complexity of Phases 1, 2 and 3 is $O(n)$. From Property 1, the ideal−time complexity of Phase 4 is $O(n \log(t+f)+n)$.

6. Addition and deletion of processors and links

We can easily modify our algorithm so that it can reconstruct the new MST after addition and deletion of processors as well as links occur. We have only to modify the definition of an *adjacent processor*. In the modified algorithm, a processor u is defined as an adjacent processor, if u is incident to an added or deleted link, or if u is adjacent to an added or deleted processor. When a pro-

cessor v is added or deleted, it causes at most d adjacent processors where d is the degree of v (i.e.the number of links incident to u). Therefore there exist $O(g+t+f)$ adjacent processors, where g is the sum of degree over all added or deleted processors. From this observation, the following theorem can be proved.

[Theorem 3] After processors and links are deleted and added, the new MST T' can be reconstructed with the message complexity $O(n'log(g+t+f)+r)$ and the ideal-time complexity $O(n'log(g+t+f)+n')$. Here, n' is the number of processors in the network after topology change, g is the sum of degree over all added or deleted processors, and $r=n+g+t$ if $f=0$ and no processor is deleted, $r=e'$ (i.e. the number of links in the network after topology change) otherwise.

References

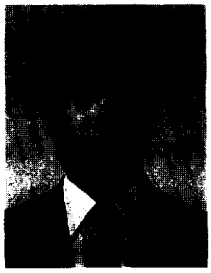
[1] B.Awerbuch : "Optimal Distributed Algorithm for Minimum Weight Spanning Tree, Counting Leader Election and related problems", Proceedings 19th Annual ACM Symposium on Theory of Computing., pp.230-240 (1987).

[2] I.Cimet and S.P.Kumar : "A Resilient Distributed Algorithms for Minimum Weight Spanning Trees", Proceedings of the 1987 International Conference on Parallel Processing., pp.196-203(1987).

[3] R.Gallager, P.Humblet and P.Spira : "A Disrtributed Algorithm for Minimum Weight Spanning Trees", ACM TOPLAS, 5, 1, pp. 66-77(1983).

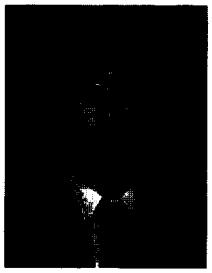
[4] J.Park, T.Masuzawa, K.Hagihara and N. Tokura : "Distributed Algorithms for Reconstructing Minumum Spanning Tree-The Case of Link Deletions-", Tech. Rep. IECEJ, COMP-89-25(in Japanes) (1989).

[5] Y.H.Tsin : "An Asynchronous Distributed MST Updating Algorithm for Handling Vertex Insertion in Networks", Proc. of the International Conference on Parallel Processing and Applications, pp.221-226(1987).



박 정 호

1980년 성균관대학교 사범대학 졸업(문학사)
1980년~1982년 성균관대학교 경영대학원 정보처리학과 졸업(경영학석사)
1985년~1987년 일본 오사까대학교 대학원 정보공학전공(공학석사)
1987년~1990년 일본 오사까대학교 대학원 정보공학전공(공학박사)
1991년~현재 선문대학교 전자계산학과 조교수
관심분야 : 분산알고리즘, 소프트웨어 공학.



민 준 영

1982년 아주대학교 산업공학과 졸업(공학사)
1986년~1989년 성균관대학교 경영대학원 정보처리학과 졸업(경영학석사)
1990년~1993년 성균관대학교 대학원 전산통계전공(박사과정수료)
1993년 9월~현재 상지대학교병설전문대학 전자계산과 교수
관심분야 : 분산알고리즘, 컴퓨터그래픽스, Neural network