A Distributed Algorithm for Ear Decomposition

S. Hannenhalli, K. Perumalla, N. Chandrasekharan *

Department of Computer Science
University of Central Florida
Orlando, FL 32816

R. Sridhar‡

‡School of Computer Science
University of Oklahoma
Norman, OK 73019

Abstract

A distributed algorithm for finding an ear decomposition of an asynchronous communication network with n nodes and m links is presented in this paper. At the completion of the algorithm either the ears are correctly labeled or the nodes are informed that there exists no ear decomposition. First we present a novel algorithm to check the existence of an ear decomposition which uses $O(m)$ messages. We also present two other algorithms, one which is time-optimal and the other which is message-optimal to determine the actual ears and their corresponding numbers after determining the existence of an ear decomposition.

1 Introduction

A distributed system consists of a set of computers (nodes) connected by a communication network. Each node communicates with their neighbors by sending messages to them. A distributed algorithm is a collection of automata, where one automata is associated with each node in the system. Given an instance I of input size n of a problem, and an algorithm A which solves the problem, the message complexity, C, of A on I is the total number of messages generated by all nodes during that execution of A; the worst-case message complexity of A on instances of size n is the maximum among the message complexities of A on all instances I of size n.

Our interest in distributed algorithms is motivated by the number of papers which have appeared in the literature related networks and distributed algorithms. These recent distributed algorithms fall into two classes; the class of algorithms for performing computations using networks and the class of algorithms for performing computations on networks. Among the papers related to computations using networks include Rajanarayanan and Iyenger [11] on set intersection and Zaks [17] on sorting and ranking. Among the papers related to computations on networks include Gallager [6] on minimum weight spanning trees, Korach, et. al. [8] on finding centers and medians in networks, Awerbuch, et. al. [3], on breadth first search, Ramarao [14] on network recognition, Sharma et. al. [15] on depth first search, and Awerbuch [2] on Max-Flow.

Ear decomposition is a technique of decomposing a given network into simpler parts such that computation on the simpler parts corresponds to the computation on the entire network. This technique of ear decompositon was first introducted in the parallel environment by Lovász [9]. This technique has been successfully used for connectivity testing [13, 10, 12, 16], planarity testing and finding planar embeddings [7]. We feel that a distributed algorithm for ear decomposition is an important step towards solving many problems on distributed networks. In the following we present a formal definition for the *ear decomposition* of graph.

Definition: An (open) ear decomposition of a graph $G = (V, E)$ is a sequence $P_0, P_1, ..., P_r$ of simple, edge-disjoint paths, with P_0 a cycle and only the endpoints of P_i, $i > 0$, are on earlier paths. In an open ear decomposition, the endpoints of each P_i, $i \geq 1$, have to be distinct.

2 Distributed Algorithm

We assume the following model of computation on the class of connected and undirected graphs which is essentially same as that in [5]. The model assumes that:

*Research supported in part by a National Science Foundation Grant No. NSF-CCR-9110159

0-8186-4212-2/93 $03.00 © 1993 IEEE

1. The nodes of the graphs correspond to the processing nodes and the undirected edges correspond to the duplex communication links between the processors.

2. Every node only knows its neighbors.

3. Nodes and the communication links are fully reliable.

4. Along a link the messages arrive at the destination in the order they are sent.

5. The only mode of communication between the processors is message-passing via the communication links.

6. It takes unit step for the house-keeping and sending the messages to the neighbors.

7. The processors communicate asynchronously, that is, the sender always hands over the message to the communication subsystem and proceeds with other communications or local computations if any.

Further, we assume that the algorithm is triggered at a single node, called the *trigger-node* or the root. The algorithm essentially consists of the following three phases:

1. A rooted spanning tree is built for the graph using a depth first search. The root of the tree is the trigger-node. While building the spanning tree the existence of an ear decomposition or an open ear decomposition is checked. The existence can be checked using $O(m)$ messages in $O(m)$ time. We omit the details due to space limitations. This determines the feasibility-status of the graph.

2. The feasibility-status of the graph regarding the absence of an ear decomposition or an open ear decomposition is broadcast to all the nodes from the root via the tree edges. This step is not executed if the graph is biconnected, *i.e.*, if an open ear decomposition exists. This can be done using $O(n)$ messages in $O(n)$ time by traversing the spanning tree constructed above.

3. Each edge in the graph is assigned a label denoting the ear it belongs to. The labeling starts with 0. The labels are stored at the adjacent nodes. Every node records the labels of all the edges incident on it.

After the algorithm terminates, every processing node knows the feasibility-status of the graph. If there exists an ear decomposition for the graph every node knows the labels of all the edges incident on it. Otherwise every node knows that there exists no ear decomposition. If $G = (V, E)$ is a connected graph, we write an edge $e = (u\ v)$ iff $e \in E$ and Preorder(u)<Preorder(v).

3 Ear labeling - Phase 3

For clarity of presentation, in the following we will give the sequential version of the distributed algorithm for the phase 3, *i.e.*, the ear-labeling, given later.

Sequential version

Seq_Ear_Label(r: Vertex)
For_all edge $(u\ v) \in E$
ear-label(u,v) $\leftarrow \infty$
cur_ear_num $\leftarrow$ 0
For_all $u \in$ V in *increasing* preorder numbers
Use_New_Ear(u)

Use_New_Ear(u: Vertex)
For_all v *suchthat* $(u\ v) \in$ nontree AND Preorder(u) < Preorder(v)
ear-label($u\ v$) $\leftarrow$ cur_ear_num
Label(v)
cur_ear_num $\leftarrow$ cur_ear_num + 1

Label(u: Vertex)
if (parent(u) $\neq u$) AND (ear-label(parent(u) u) > cur_ear_num)
ear-label(parent(u) u) $\leftarrow$ cur_ear_num
Label(parent(u))

We scan the vertices in the increasing order of their Preorder numbers and at every node u, look for an adjacent nontree edge connecting u to a descendant v. For each such nontree edge (in arbitrary order), we label that edge with the cur_ear_num. Let $TP_{(v,u)}$ be the unique path of tree edges from v to u. We follow $TP_{(v,u)}$ while labeling the edges with cur_ear_num until we encounter an edge with an ear-label lower than cur_ear_num. We then increment the cur_ear_num variable. For the distributed version, the **For** loop in the Seq_Ear_Label routine is simulated by sending one by one, a *Use_New_Ear* message to the children in the increasing Preorder number of the children synchronized by a *Used* message from the children. A *Used* message is sent to the parent if all the children

are exhausted. We will now present the distributed version.

Distributed version

We present two distributed algorithms for the ear-labeling, *viz. message-optimal* and *time-efficient*. We will present them in the order listed. We will add the suffix M after the message names to denote message-optimal and the suffix T to denote the time-optimal algorithm.

3.0.1 Message-Optimal algorithm

In the following we present the message-optimal distributed algorithm for the ear-labeling phase. This is essentially the simulation of the sequential algorithm stated above in a distributed setting. All nodes are scanned in the increasing order of their Preorder numbers. Every node being scanned looks for an adjacent nontree edge connecting to a descendant and sends a *Label_M* message on it. This *Label_M* message percolates up back towards the initiator via the unique path of tree edges. Every edge in the path is labeled by the same ear-label embedded in the *Label_M* message. The message retracts back in the form of *Labeled_M* message when it encounters an already labeled edge or reaches the initiator. When the initiator receives the *Labeled_M* message it looks for next adjacent nontree edge to send next *Label_M* message. If it does not find one it looks for next unscanned child to send a *Use_New_Ear_M* message. Again if it doesn't finds one, it sends a *Used_M* message to its parent. The algorithm terminates when the root receives a *Used_M* message from all of its children. To send the *Labeled_M* message which retraces the path created by the corresponding *Label_M* message, every node maintains a variable *back-node*.

Messages

Use_New_Ear_M(cur_ear_num)
Used_M(cur_ear_num)
Label_M(sender_id, initiator_id, ear_label)
Labeled_M(initiator_id, ear_label)

Process at node i on the reception of:

Use_New_Ear_M(e):
current_ear_number$_i \leftarrow e$
if $\exists\ c \in$ nontree$_i$ suchthat Preorder$_c >$ Preorder$_i$
Send to c *Label_M*(i,i,current_ear_number$_i$)
delete c from nontree
else if $\exists\ c$ *in* Children$_i$
delete c from Children$_i$
Send to c
Use_New_Ear_M(current_ear_number$_i$)
else if Parent$_i \neq i$ send to Parent$_i$
Used_M(current_ear_number$_i$) □

Label_M(j,initiator,e):
ear-label(j) $\leftarrow e$
back-node $\leftarrow j$
if (initiator $\neq i$) AND (ear-label(Parent$_i$) $> e$)
ear-label(Parent$_i$) $\leftarrow e$
Send to Parent$_i$ *Label_M*(i,initiator,e)
else send to back-node *Labeled_M*(initiator,e) □

Labeled_M(initiator,e):
if (initiator $\neq i$)
send to back-node *Labeled_M*(initiator,e)
else send to i *Use_New_Ear_M*(e+1)

Used_M(e):
if $\exists\ c$ *in* Children$_i$
delete c from Children$_i$
Send to c *Use_New_Ear_M*(e)
else if Parent$_i \neq i$ send to Parent$_i$ *Used_M*(e) □

We will discuss the correctness of the algorithm after presenting the time-optimal version of the algorithm.

Communication and Time Complexity:

The communication and time complexity can be easily shown to be $O(m)$.

Now we present the *time-efficient* algorithm for the ear-labeling.

3.0.2 Time-Efficient Algorithm

We will first describe the algorithm and then state it formally. This phase of the ear decomposition algorithm starts with the reception of the message *Use_New_Ear_T* by the root. The message is sent by the root to itself in one of the previous phases depending on the feasibility-status of the graph. Node i receiving a *Use_New_Ear_T* message checks the Preorder numbers of all the neighbors connected to it by a nontree edge. It then compares them with its own Preorder and sends a *Label_T* message to all those neighbors whose Preorder is higher than its own Preorder (a descendant). A distinct ear-label is sent on each such edge starting from the current lowest ear-label available incremented by 1 for each such edge. For example if the current highest label used is e and node i has k adjacent nontree edges connecting it to k descendants, then those edges are labeled as $e+1$ through $e+k$ in arbitrary order. After the node i does

the above or it doesn't find any candidate to send the *Label_T* message to, it sends a *Use_New_Ear_T* message to one of its unscanned children along with the current highest ear-label used. When all the children are scanned , it sends back a *Used_T* message to its parent which in turns looks for one of its unscanned children and so on. On receiving a *Label_T*, message node i first records the ear-label of the edge on which the message came and then checks its own *id* against the *id* of the initiator of the *Label_T* message. It passes on the message to its parent if node i itself is not the initiator and ear-label of the parent edge is greater than the ear-label embedded in the *Label_T* message. In this entire process we basically scan the spanning tree of the graph following the preorder numbering. At every node we check for nontree edges going to a descendant, and send a label message on each of them which percolates up towards the initiator via the chain of parents and stops when it reaches the initiator or it encounters an edge labeled by a lower ear-label. In a distributed environment it is possible that a tree edge is labeled many times with a different ear-label until it finally gets the lowest ear-label among all the *Label_T* messages that pass through it. A higher ear-label is always overridden by a lower one and never the reverse. We will refer to the set of edges with ear-label e as P_e. There are two facts worth noting:

Observation 3.1: Exactly one *Label_T* message is sent on every nontree edge with a unique ear-label. This implies that for every e, P_e has exactly one nontree edge belonging to it.

Observation 3.2: After the algorithm terminates, any P_e is of the form:
$(x_0\ x_1)$, $(x_1\ x_2)$, $\cdots$, $(x_{k-1}\ x_k)$, $k \geq 1$
where $(x_0\ x_1)$ is the unique nontree edge in P_e, x_k is the parent of x_{k-1}, $1 < i \leq k$, and all nodes are distinct with the exception that when kk > 1, x_k may be same as x_0.

We will now state the distributed algorithm in a more formal manner:

We will give the prototypes of the messages and the action taken by node i on the reception of each message.

Messages

*Use_New_Ear_T*current_ear_num)
Label_T(sender_id, initiator_id, ear_label)
Used_T(current_ear_num)

Process at node i on the reception of:

Use_New_Ear_T(e):

current_ear_number$_i$ $\leftarrow$ e
For_all c where $c \in$ nontree$_i$ AND Preorder$_c$ > Preorder$_i$
Send to c *Label_T*(i,i,current_ear_number$_i$)
current_ear_number$_i$ $\leftarrow$ current_ear_number$_i$ + 1
if $\exists$ c *in* Children$_i$
delete c from Children$_i$
Send to c *Use_New_Ear_T*(current_ear_number$_i$)
else if Parent$_i$ $\neq$ i send to Parent$_i$ *Used_T*(current_ear_number$_i$) □

Label_T(j,initiator,e):

ear-label(j) $\leftarrow$ e
if (initiator $\neq$ i) AND (ear-label(Parent$_i$) > e)
ear-label(Parent$_i$) $\leftarrow$ e
Send to Parent$_i$ *Label_T*(i,initiator,e) □

Used_T(e):

if $\exists$ c *in* Children$_i$
delete c from Children$_i$
Send to c *Use_New_Ear_T*(e)
else if Parent$_i$ $\neq$ i send to Parent$_i$ *Used_T*(e) □

In the following, we will discuss the correctness of the proposed distributed algorithm. We will restrict our discussion to the third phase *i.e.*, the ear labeling phase. The first phase simply does feasibility checking which is a straightforward implementation of the algorithm in [1]. To show the correctness we will first discuss the relevant facts.

Lemma 3.3: The root of DFS spanning tree of a bridgeless graph has atleast 1 nontree edge adjacent to it. □

Lemma 3.4: Let u and v be distinct nodes initiating *Label_T* messages with ear-labels e_u and e_v respectively. $e_u < e_v$ iff Preorder$_u$ < Preorder$_v$. □

Theorem 3.5: Given a bridgeless graph along with a rooted DFS spanning tree, phase 3 of the algorithm gives an ear decomposition of the graph. □

Lemma 3.6: If the graph is biconnected then the algorithm gives an open ear decomposition. □

Communication and Time Complexity:

Since the *Use_New_Ear_T* and *Used_T* messages are sent exactly once along each tree edge, the number of these messages is in $O(n)$. Now, consider the node u with preorder number k. Let the number of adjacent nontree edges connecting u to its descendants be d_k. Any *Label_T* message initiated by u can go through $(n-k)$ edges at the most (It cannot involve a node with lower preorder than the initiator). So the max-

imum number of *Label_T* messages generated in the network due an initiation by u can be $(n-k)$. Hence the total number of *Label_T* message generated due to u can be $d_k * (n-k)$. This gives us the upper bound on the number of *Label_T* messages as:

$\sum_{k=0}^{n-1} d_k * (n-k)$
Also, $\sum_{k=0}^{n-1} d_k = m - n + 1$
The complexity simplifies to:
$m * n - n^2 + n - \sum_{k=0}^{n-1} k * d_k$
As evident, the message complexity will be $O(m * n)$ in the worst case but will be linear for more practical purposes.
The algorithm terminates as soon as all the edges are labeled. Consider the node u with preorder number k. It receives a *Use_New_Ear_T* message after k steps from the start of the phase 3. After it receives the message, it will take atmost $(n-k)$ time steps to label any ear starting at u following the same arguments as for message complexity. So the time for completion of any ear labeling is bounded by $(k + n - k)$, *i.e.*, exactly n.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[2] B. Awerbuch. Reducing complexities of the distributed max-flow and breadth-first-search algorithms by means of network synchronization. *Networks*, 15:425–437, 1985.

[3] B. Awerbuch and R. G. Gallager. A new distributed algorithm to find breadth-first-search trees. *IEEE Trans. Information Theory*, IT-33:315–322, May 1987.

[4] K. Bogart. *Introductory Combinatorics.* Harcourt, Brace and Jovanovich, 1990.

[5] T. Cheung. Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Trans. Software Eng.*, SE-9(4):504–512, July 1983.

[6] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM TOPLAS*, 5(1):66–77, 1983.

[7] P. N. Klein and J. H. Reif. An efficient parallel algorithm for planarity. In *Proc. 27th Annual IEEE FOCS*, pages 465–477, 1986.

[8] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM TOPLAS*, 6(3):380–401, 1984.

[9] L. Lovász. Computings ears and branchings in parallel. In *Proceedings of the 26th Annual IEEE Symp. on Foundations of Computer Science (Portland, OR)*, pages 464–467, 1985.

[10] G. L. Miller and V. Ramachandran. A new graph triconnectivity algorithm and its parallelization. In *Proc. 19th Annual ACM STOC*, pages 335–344, 1987.

[11] S. Rajanarayanan and S. Iyengar. A new optimal distributed algorithm for the set intersection problem. *Inform. Process. Lett.*, 38(3):143–148, May 1991.

[12] V. Ramachandran. Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. *to appear, Synthesis of Parallel Algorithms, J. H. Reif, ed.*, 1992.

[13] V. Ramachandran and U. Vishkin. Efficient parallel triconnectivity in logarithmic time. *VLSI Algorithms and Architecture*, pages 33–42, 1988.

[14] K. Ramarao. Distributed algorithms for network recognition problems. *IEEE Trans. on Computers*, 38(9), Sept 1989.

[15] M. Sharma, S. Iyengar, and N. Mandyam. An efficient distributed depth-first-search algorithm. *Inform. Process. Lett.*, 32(4):183–186, Sept. 1989.

[16] B. S. Y. Maon and U. Vishkin. Parallel ear decomposition search (eds) and st-numbering in graphs. *Theoretical Computer Science*, 47:277–298, 1986.

[17] S. Zaks. Optimal distributed algorithms for sorting and ranking. *IEEE Trans. Comput.*, C-34(4):376–379, 1985.