On the optimal synchronizer for asynchronous distributed networks

Yoji KAJITANI* Hiroshi MIYANO* Shuichi UENO* K.THULASIRAMAN**

* DEPT. OF ELECTRICAL AND ELECTRONIC ENGINEERING, TOKYO INSTITUTE OF THCHNOLOGY

**Concordia University

ABSTRACT: In this paper, we consider the networks with acyclic k-spanners which are closely releted to the optimal synchronizer for asynchronous distributed networks. We characterize the graphs with acyclic 2-spanners, and give a polynomial-time algorithm to decide whether a given outerplanner graph has an acyclic k-spanner.

1. INTRODUCTION

The asynchronous network is a communication network without global clock, described by an undirected simple graph $N=(V,E)$, where the set of vertices V represents processors of the network and the set of edges E represents bidirectional communication channels between them. Each vertex has a distinct identity. The synchronizer is a distributed algorithm that enables any synchronous distributed algorithm to run in any asynchronous network by generating sequences of clock pulses at each vertex of the network.

The communication and time complexities are used to evaluate performances of algorithms. The complexities of the asynchronous algorithm A resulting from combining a synchronous algorithm S with a synchronizer ν are

$C(A) = C(S) + T(S)\cdot C(\nu)$, and

$T(A) = T(S)\cdot T(\nu)$,

where C(X) and T(X) are the communication and time complexities of the algorithm X, respectively.

Because $O(|V|)$ and $O(1)$ are lower bounds on $C(\nu)$ and $T(\nu)$, respectively, a synchronizer ν is said to be optimal if $C(\nu)=O(|V|)$ and $T(\nu)=O(1)$. Unfortunately, it is known that there exists a trade-off between the communication and time complexities of the synchronizer, and no optimal synchronizer can be constructed in general.

More precisely, it is proved in [1] that for any positive integer i there exists a network N such that if $T(\nu)<i-1$ then $C(\nu)>\frac{1}{4}n^{1+1/i}$ for any synchronizer ν. This leads us to a fundamental problem of characterizing the networks with optimal synchronizers. [2] shows a sufficient condition for a network to have an optimal synchronizer. A subgraph $N'=(V,E')$ is called a k-spanner of N if for every $(u,v)\in E$, the distance between u and v in N' is at most k. It is proved in [2] that if a network N has a k-spanner with $O(|V|)$ edges, then N has an optimal synchronizer. [2] also shows that the hypercube has a 3-spanner with $O(|V|)$ edges and thus has an optimal synchronizer.

In this paper, we consider the networks with acyclic k-spanners (k-AS's for short) as the first step to characterize the networks with optimal synchronizers. Note that a k-AS is a spanning tree of N and thus has $O(|V|)$ edges. We characterize the graphs with 2-AS's, and give a polynomial-time algorithm to decide whether an outerplanar graph has a k-AS for a given integer k.

2. ACYCLIC 2-SPANNER

In this section, we characterize the graphs with 2-AS's. It is easy to see that a graph has a 2-AS if and only if each 2-connected component of the graph has a 2-AS. Thus we may assume without loss of generality that the graph is 2-connected.

LEMMA 1:

Let $G=(V,E)$ be a 2-connected graph with a 2-AS. If a set of two vertices $\{u,v\}$ is a separator of G, then $(u,v)\in E$ and every 2-AS contains (u,v).

PROOF:

Suppose that the graph obtained from G by removing u and v consists of the connected components $G_1,\ldots,$ and G_k. We prove the lemma by showing that any acyclic spanner without (u,v) is not a 2-AS of G.

Let G' be an acyclic spanner without (u,v). The unique path between u and v in G' is contained in exactly one component, say G_1, and the length of the path is at least 2. Then there exists a cotree edge e in G_2 such that the length of the fundamental circuit determined by e with respect to G' is more than 3, for G is 2-connected. Thus G' is not a 2-AS.

A graph G is said to be propped if there exists an edge (u,v) for any minimal separator $\{u,v\}$ of G. We call such an edge (u,v) a prop of G.

Now Lemma 1 is restated as follows.

LEMMA 1':

A 2-connected graph with 2-AS is propped. Moreover, every prop is contained in any 2-AS.

We characterize 3-connected graphs with 2-AS's.

LEMMA 2:

Suppose G is a 3-connected graph. G has a 2-AS if and only if G has a star-tree, that is, G has a vertex adjacent to every other vertex of G.

PROOF:

If G has a star-tree, it is easy to see that the star-tree is a 2-AS of G.

Conversely, let T be a spanning nonstar-tree of G. Then T contains a path P of length 3. Suppose the vertices u, v, w, and x appear on P in this order. Since G is 3-connected, there exists in G a path between u and x vertex-disjoint from P. This means that there exists a cotree edge e such that the length of the fundamental circuit determined by e with respect to T is more than 3. Thus T is not a 2-AS.

For propped graph G and a prop (u,v) of G, cutting G with respect to (u,v) is to delete the vertices u and v from G and add edge (u,v) to every connected component of the resulting graph.

Given a propped graph G, we obtain $\tilde{G}$ by successively cutting the current graph with respect to a prop until every connected component becomes 3-connected. Note that $\tilde{G}$ is unique independent of the order of cuttings.

From these lemmas, we have the following result.

THEOREM 1:

A 2-connected graph G has a 2-AS if and only if each connected component C of $\tilde{G}$ has a star-tree with all props of G contained in C.

PROOF:

The union T of star-trees described in the theorem is a spanning tree of G because every prop of G is contained in T. Thus T is a 2-AS of G.

The necessity is trivial by the above lemmas.

From Theorem 1, we obtain the following polynomial-time algorithm to decide whether a given graph has a 2-AS.

ALGORITHM 1:

1. For every separator $\{u,v\}$,
   if $(u,v)\in E$ then goto 2
   else output 'No' and halt.
2. If props contain a circuit
   then output 'No' and halt
   else accept all of props for edges of 2-AS.

3. Repeat cuttint operation until each connected component has no prop.
   {Then each connected component is 3-connected}
4. For each connected component P,
   find a vertex
     which is adjacent to all vertices of V(P)
     and incident to all props of G contained in P,
   then accept the incident set of the vertex
     as edges of 2-AS.
   If there exists no such vertex
     then output 'No' and halt.
5. Output 'Yes' and halt.

3. THE ACYCLIC k-SPANNER OF OUTER PLANAR GRAPHS

In this section we give a polynomial-time algorithm to decide whether a given outerplanar graph has a k-AS.

3.1 ALGORITHM

Without loss of generality, we may assume that the graph is 2-connected. The edges on the boundary of the infinite region are called outer edges, and others are called inner edges.

ALGORITHM 2:

INPUT: k: integer G: outerplanar graph
OUTPUT: ANSWER: {'Yes','No'} P: k-AS

1: If G is a simple circuit then
     if |V| > k+1 then output 'No' and halt
       else P=E-{e}, output 'Yes' and halt.
         /* e is an arbitary edge */
2: Embed G into plane such that every vertex is on the boundary of the infinite region.
   /* This step is not essential but convenient for description of the algorithm */
3: P ← ϕ, Q ← ϕ.
   /* Q is defined for convenient as well */
4: Let label(e)=1 for each outer edge.
5: Find a face F which has exactly one non-labeled edge.
   /* Such a face must exist (See 3.2) */
6: Let e be the edge whose label(e) is minimum in F, and f be the non-labeled edge.
7: If label(e)+|F|-2 > k then goto 12.
   /* |F| means the number of elements of set F */
8: Q ← Qu{e}, P ← Pu(F-{e,f}).
9: Remove F-{f} from the graph.
10: label(f) ← label(e)+|F|-2.
11: Goto 13.
12: Q ← Qu{f}, and remove f from the graph.
13: If there exists a non-labeled edge then goto 5.
14: /* Now the graph is a simple circuit */
   Let F be the circuit.
   If label(e)+|F|-3 < k for some edge e,
   then
     Q ← Qu{e}, P ← Pu(F-{e}),
     output 'Yes', and list P
   else
     output 'No'.
15: Halt.

3.2 Validity of the algorithm

Notice that there always exists a face satisfying the condition in step 5. Because, in each stage of the algorithm, an edge is labeled if and only if the edge is an outer edge of the current outerplanar graph, and an outerplanar graph has a face with exactly one inner edge.

First, we prove that if this algorithm outputs 'Yes' then P is k-AS. At the beginning of this algorithm, two end vertices of any labeled edge are not spaned by P because P is empty. When an edge is labeled, two end vertices of the edge is not spaned by P. And there is a path in P from any vertex which has already removed in step 9 to some vertex which has not removed yet. This means that the union of a spanning tree of the current graph and P becomes a spanning tree of the original graph. Thus the output P is a spanning tree of G. We must prove that each fundamental circuit of the tree is of length k+1 or less. Assume that C is a fundamental circuit of P of length more than k+1. Let ℓ be the length of C, and

V(C) be the vertices of C. Also let $e_1, e_2, \ldots$, and e_k be cotree edges each of which connects two vertices of V(C), sorted by increasing order of labels. Note that C is the fundamental circuit defined by e_1 with respect to P. It is clear that label(e_1)=1. When e_i is labeled in step 10, the edge e, which appeared in the statement of step 10, is some e_j (i>j). Let ℓ_i and ℓ_j be the lengths of fundamental circuits defined by e_i and e_j respectively. Then it is easy to see that label(e_i)=label(e_j)+|F|-2 and $\ell_i=\ell_j+|F|-2$. So the length of fundamental circuit defined by e_i is $\ell+1-$label(e_i). Since this circuit satisfies the condition of step 7, there is a tree edge which is not got into P until step 14 executed. However, this circuit does not satisfy the condition of step 14, that contradicts to our assumption that P is the output. Therefore P is a k-AS.

Conversely, we prove that if the output is 'No', the graph G does not have a k-AS. Suppose that graph G has a k-AS and the algorithm outputs 'No'. In this algorithm, P and Q are updated one after another. Let the pairs of (P, Q) of each step be $(P_1, Q_1), (P_2, Q_2), \ldots, (P_m, Q_m)$. Note that $P_1=Q_1=\phi$, and $P_i \subseteq P_j, Q_i \subseteq Q_j$ (i<j). Then there is an integer t which satisfies the following conditions.

(CONDITIONS)

$\exists T$:k-AS such that $P_t \subseteq T, Q_t \cap T=\phi$

$\nexists T$:k-AS such that $P_{t+1} \subseteq T, Q_{t+1} \cap T=\phi$

If such a number t dose not exist then P_m must be k-AS and this algorithm outputs 'Yes'. The updating from (P_t, Q_t) to (P_{t+1}, Q_{t+1}) can occur at step 7, 11 or 13. We prove that this is impossible. It is trivial that the updating which satisfies the conditions does not occur at step 13. Therefore we disscuss steps 7 and 11.

Assume that such an updating occurs at step 7. Then,

$Q_{t+1}=Q_t \cup \{e\}$, $P_{t+1}=P_t \cup (F-\{e, f\})$.

Suppose that T is a k-AS which satisfies that $P_t \subseteq T$ and $Q_t \cap T=\phi$. By the definition of t, $e \in T$ or $F-\{e, f\} \not\subset T$. Also by the fact that a labeled edge is an outer edge, $E-(P_{t+1} \cup Q_{t+1})$ does not span two end vertices of any edge of $F-\{f\}$. So, $(F-\{f\})$ contains at most one cotree edge of the tree which contains all edges of P_t and none of Q_t. Now two cases remain: i) $F-\{f\} \subset T$ and ii) $\exists h \in F-\{f\}, h \neq e, h \notin T$.

i) $f \notin T$ holds, then it is clear that $T \cup \{f\}-\{e\}$ is k-AS. This contradicts the definition of t.

ii) Obviously $T \cup \{h\}-\{e\}$ is k-AS which contradicts the definition of t.

Next, we assume that such an updating occurs at step 11. Then,

$P_{t+1}=P_t, Q_{t+1}=Q_t \cup \{f\}$.

There exists a k-AS satisfying that $P_t \subseteq T$, $Q_t \cap T=\phi$, and $f \in T$, by the definition of t. But this means that T contains a fundamental circuit of length more than k+1. This is a contradiction.

It is easy to see that the time complexity of this algorithm is $O(n^2)$.

REFERENCES

[1] B. Awerbuch: Complexity of Network Synchronization, Journal of ACM, Vol. 32 No. 4 (1985) 804-823.

[2] David Peleg and J. D. Ullman: An optimal synchronizer for the hypercube, Symp. on Primciples of Distributed Computing (1987).