Graph-Theoretic Proof of a Network Theorem and Some Consequences

K. THULASIRAMAN, R. JAYAKUMAR, AND M.N.S. SWAMY

Abstract—A graph-theoretic proof of a network theorem is given. Some consequences of this theorem in relation to the analysis of an algorithm are discussed.

Manuscript received January 6, 1983; revised February 28, 1983.

The authors are with the Faculty of Engineering, Concordia University, Montreal, P.Q. H3G 1M8, Canada.

In this letter, we give a graph-theoretic proof of a network theorem (Theorem 1, which follows), which was recently brought to our attention [1] and discuss some of its consequences. For graph theory terms not defined here, [2] may be referred to.

Consider an *RLC* network N without mutual inductances. Let N have n nodes and m elements $e_1, e_2, \cdots, e_m$, with each element being a resistance or a capacitance or an inductance. The impedance and admittance of each e_i will be denoted by z_i and y_i, respectively. Also, the two nodes of each e_i will be denoted by i_1 and i_2. If $Z(i_1, i_2)$ denotes the driving point impedance of N across the pair of nodes i_1 and i_2, then we have the following theorem:

Theorem 1

$$\sum_{i=1}^{m} y_i Z(i_1, i_2) = n - 1. \tag{1}$$

Proof: Let T denote the set of all spanning trees of N and, for each i, let T_i denote the set of all spanning 2-trees of N separating the nodes i_1 and i_2. Note that adding e_i to a spanning 2-tree separating i_1 and i_2 will generate a spanning tree. Further, let $w(t)$ denote the admittance product of spanning tree t and $w(t_i)$ denote the admittance product of spanning 2-tree t_i separating i_1 and i_2. It is easy to see that if $t = t_i \cup e_i$, then

$$w(t) = y_i w(t_i).$$

If

$$W(T) = \sum_{t \in T} w(t)$$

and

$$W(T_i) = \sum_{t_i \in T_i} w(t_i)$$

then it is known [2] that

$$Z(i_1, i_2) = \frac{W(T_i)}{W(T)}.$$

Thus to prove the theorem, we need to show that

$$\sum_{i=1}^{m} y_i W(T_i) = (n - 1)\, W(T) \tag{2}$$

or

$$\sum_{i=1}^{m} y_i \sum_{t_i \in T_i} w(t_i) = (n - 1) \sum_{t \in T} w(t).$$

Consider any tree admittance product $w(t)$. We may assume, without loss of generality, that the spanning tree t contains the elements $e_1, e_2, \cdots, e_{n-1}$. Then for every $i = 1, 2, \cdots, n - 1$, $t - e_i$ is a spanning 2-tree t_i separating the nodes i_1 and i_2. So for every $i = 1, 2, \cdots, n - 1$

$$w(t) = y_i w(t_i)$$

for some spanning 2-tree t_i. Thus the admittance product $w(t)$ appears exactly once in each $y_i W(T_i), i = 1, 2, \cdots, n - 1$. In other words, each $w(t)$ appears $(n - 1)$ times in the sum on the left-hand side of (2). The theorem follows since all the terms on the left-hand side of (2) are admittance products of spanning trees. □

Suppose all the elements of the network N are 1-Ω resistances. Then (1) reduces to

$$\sum_{i=1}^{m} Z(i_1, i_2) = n - 1. \tag{3}$$

0018-9219/83/0600-0771$01.00

This means that

$$\min_i \{Z(i_1, i_2)\} \leq \frac{n-1}{m}$$

or

$$\max_i \{Y(i_1, i_2)\} \geq \frac{m}{n-1} \tag{4}$$

where

$$Y(i_1, i_2) = \frac{1}{Z(i_1, i_2)}.$$

Let $Y_{max} = \max_i \{Y(i_1, i_2)\}$. In the study of the computational complexity of a spanning tree enumeration algorithm [3], the following two questions, relating to a resistance network N in which all the elements are 1-Ω resistances, arise:

1) Determine Y_{max}.
2) Determine the elements e_i of N such that $Y(i_1, i_2) = Y_{max}$.

As a first step towards answering the above questions, we now show that the lower bound for Y_{max} as given in (4) cannot be improved. In other words, we show that for some network N, $Y_{max} = m/(n-1)$.

We shall call a network complete if there is an element connecting every pair of nodes of the network. In fact, the graph of a complete network is a complete graph. Consider a complete resistive network N in which all resistances are of 1-Ω value. Clearly for such a network

$$Z(i_1, i_2) = Z(j_1, j_2)$$

for all $i, j = 1, 2, \cdots, m$. Let Z be the common value of $Z(i_1, i_2)$. Then it follows from (3) that

$$Z = \frac{n-1}{m} = \frac{2}{n}$$

since for a complete network $m = n(n-1)/2$. So

$$Y = \frac{1}{Z} = \frac{n}{2}.$$

Thus we have the following theorem:

Theorem 2

For an n-node resistance network of m elements each of 1-Ω value

$$Y_{max} \geq \frac{m}{n-1}.$$

For a complete network containing only 1-Ω resistances, the lower bound in the above inequality is achieved. □

References

[1] L. Roytman, private communication.
[2] M.N.S. Swamy and K. Thulasiraman, *Graphs, Networks, and Algorithms*. New York: Wiley-Interscience, 1981.
[3] R. Jayakumar, K. Thulasiraman, and M.N.S. Swamy, "Complexity of computation of a spanning tree enumeration algorithm," submitted to *IEEE Trans. Circuits Syst.*

GRAPH-THEORETIC CONSIDERATIONS ON THE OPTIMAL SYNCHRONIZER FOR ASYNCHRONOUS DISTRIBUTED NETWORKS

Y. KAJITANI* H. MIYANO* S. UENO* K. THULASIRAMAN**

* DEPT. OF ELECTRICAL AND ELECTRONIC ENGRG., TOKYO INST. OF THCH., MEGURO-KU TOKYO

**DEPT. OF ELEC. ENGRG., CONCORDIA UNIV., MONTREAL CANADA

ABSTRACT

An acyclic k-spanner (k-AS) of a graph is a spanning tree such that every fundamental circuit is of length k+1 or less. The concept of the k-AS is not only graph theoretically interesting but closely related to the optimal synchronizer for better operation of asynchronous distributed networks. This paper contains two results. First, a compact characterization of the graph that has a 2-AS together with an $O(n^2)$ time algorithm to find one is given. Second, for any integer k and an outerplanar graph, an $O(n^2)$ time algorithm to find a k-spanner of the graph if one exists is given.

1. INTRODUCTION

The asynchronous network is a communication network without a global clock. We model it by an undirected simple graph $N=(V,E)$ where the set V of vertices represents the processors of the network and the set E of edges represents bidirectional communication channels between processors. Each vertex has a distinct identity.

An asynchronous network is easier to build. On the other hand, a synchronous network is easier to develop its softwares. A synchronizer of an asynchronous network is a distributed algorithm that enables any synchronous distributed algorithm to run in the network by generating sequences of trigger pulses at each vertex. Therefore, if we have a synchronizer ν, we can implement an algorithm S designed for synchronous networks on an asynchronous network. Thus, we can take both advantages unless additional complexity by ν is large.

The communication and time complexities are used to evaluate performances of distributed algorithms. The complexities of an 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 trivial lower bounds of $C(\nu)$ 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 complexity and time complexity of synchronizers on a network, 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 fact leads us to a fundamental problem of characterizing the networks for which we can construct optimal synchronizers. [2] shows a sufficient condition for such a network on which the idea of this paper is based.

A subgraph $N'=(V,E')$ of N is called a spanner of N if E' contains a spanning tree of N. A spanner N' is called a k-spanner of N if for every $(u,v)\in E$, the shortest path length 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 (i.e. cycle free) k-spanners (k-AS's for short) as a 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. Our results include polynomial time algorithms to solve the following two problems.

1. For any graph, find a 2-AS if one exists.
2. For any outer planar graph, find a k-AS for any k if one exists.

2. ACYCLIC 2-SPANNERS

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 assume without loss of generality that the graph is 2-connected.

We define a separator of a connected graph G. For a set of vertices S ($S\subset V$), G-S denotes the subgraph induced

by V-S. S is a separator of G if and only if G-S is not connected and for any S'⊆S, G-S' is connected. Especially, a separator S is called a t-vertex-separator if |S|=t.

LEMMA 1:

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

PROOF:

Suppose that the graph G-{u, v} consists of connected components $G_1, \ldots,$ and G_k. And let H_i be a subgraph of G induced by $V(G_i) \cup \{u, v\}$, where $V(G_i)$ is the vertex set of G_i.

We prove the lemma by showing that any acyclic spanner without edge (u, v) is not a 2-AS of G. Let T' be an acyclic spanner without edge (u, v). The unique path between u and v in T' is contained in exactly one of H_i's, say H_1, and the length of the path is at least 2. Then there exists a cotree edge e in H_2 such that the length of the fundamental circuit determined by e with respect to T' 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 (u, v)∈E for every 2-vertex-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 graph with a 2-AS is propped. Moreover, every prop is contained in any 2-AS. ∎

Next, we characterize the graphs without props which have 2-AS's. A star-tree, which appears in the following lemma, is a spanning tree without a path of length 3. Note that G has a star-tree if and only if G has a vertex adjacent to every other vertex of G.

LEMMA 2:

A graph G without props has a 2-AS if and only if G has a star-tree.

PROOF:

If G is a K_3 then the lemma is trivial. So we assume that |V|>3.

If G has a star-tree, it is trivially 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 that the vertices u, v, w, and x appear on P in this order. G is 3-connected since a 2-vertex-separator must have a prop. Therefore, there exists in G a path between u and x, say P', which contains neither v nor w. Let e be any edge that belongs to both P' and the fundamental cutset defined by edge (u, w) with respect to T. Then 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 construct a graph G' by the following process. Obtain the connected components G_1, G_2, ..., and G_k of G-{u, v}. And let H_i (i=1, ..., k) be a subgraph of G induced by $V(G_i) \cup \{u, v\}$. Then the disjoint union of these graphs H_i's is G'.

Given a propped graph G, we obtain $\tilde{G}$ by successively cutting the current graph with respect to a prop until every connected component has no props. Note that $\tilde{G}$ is unique independent of the order of cuttings.

From LEMMA 1 and 2, we have the following result.

THEOREM 1:

A 2-connected graph G has a 2-AS if and only if the set of all props contains no circuits and each connected component C of $\tilde{G}$ has a star-tree which contains all the props of G contained in C. Moreover if there is such a star-tree in each component, the union T of the edges of these star-trees is a 2-AS of G.

PROOF:

The necessity is trivial by the above lemmas. So we prove sufficiency.

An edge which is contained in more than one component is a prop, which is contained in T. In each component, T has no circuits. Therefore T has no circuits. It is trivial that T is a spanning tree and every fundamental circuit is of length 3. ∎

Based on Theorem 1, we propose the following polynomial-time algorithm to find a 2-AS T if one exists.

ALGORITHM 2-AS:

```
INPUT:   G: 2-connected graph
OUTPUT:  answer: {'Yes', 'No'}  P: edge set
  /* if answer='Yes' then P is a 2-AS */

1. P := φ.
2. For every 2-vertex-separator {u, v},
     if (u, v)∉E then answer:='No' and halt.
3. If the set of the props contains a circuit
     then answer:='No' and halt
     else P:=P∪{all props}.
4. Repeat the cutting operation until each connected
   component has no prop.
5. For each connected component C do
     if there exists such a vertex that
        is adjacent to all vertices of V(C)
        and incident to all props of G contained in C,
     then P:=P∪{edges incident to such a vertex}
     else answer:='No' and halt.
```

6. answer:='Yes' and halt.
/* End of Algorithm */

TIME COMPLEXITY OF ALGORITHM 2-AS:

The time complexity of this algorithm depends on what subroutines we use for each step. It will be as total $O(|V|\cdot|E|)$ if we use very familiar ways as follows. A 2-vertex-separator which includes a vertex v is found by finding all 2-connected components of G-{v}. All the 2-connected components of a graph can be found in $O(|E|)$ by constructing a depth-first-search tree. Hence all the 2-vertex separators can be found in $O(|V|\cdot|E|)$. Then, step 2 is executed in $O(|V|\cdot|E|)$. Step 3 and 4 follow with additional $O(|E|)$ time. Trivially, step 5 is executed in $O(|V|\cdot|E|)$. Therefore, as total, the time complexity of the algorithm can be $O(|V|\cdot|E|)$.

3. ACYCLIC k-SPANNERS OF OUTER PLANAR GRAPHS

It seems hard to find a k-AS for an arbitary graph and k. However, we can show that the problem is solvable in polynomial time for outerplanar graphs.

First, we propose an algorithm (in 3.1) to find a k-AS of a given outerplanar graph if one exists, and then, show its validity and time complexity (in 3.2).

3.1 ALGORITHM

Without loss of generality, we assume that the graph is a 2-connected outerplanar graph. An outerplanar graph is called outerplanarly embedded, if it is drawn on a plane such that no two edges cross each other and every vertex is on the same face called the infinite region. Edges on the boundary of the infinite region are called outer edges and the others are called inner edges.

ALGORITHM OUTERPLANAR k-AS:

```
INPUT:   k: integer  G: outerplanar graph
OUTPUT:  answer: {'Yes', 'No'}  P: edge set
  /* if answer='Yes' then P is a k-AS */

VARIABLE:
        Q: set of edges
        label[e:edge]: array of integer
 /* An edge e whose label[e] is not
        defined is called unlabeled */

1: If G is a simple circuit then
     if |V| > k+1
       then answer:='No' and halt
       else select an arbitary edge e,
            P:=E-{e}, answer:='Yes' and halt.
2: Identify each edge if it is an outer edge or an
   inner one.
3: P ← φ, Q ← φ.
   /* Q will be the complement of a k-AS P */
4: For each outer edge e do label[e]:=1.
5: Find a face F which has exactly one unlabeled
   edge.
   /* Such a face must exist (See 3.2) */
6: Let e be any one of the edges whose label[e] is
   minimum in F, and f be the unlabeled edge.
   /* f is an inner edge of the current graph */
7: If label[e]+|F|-2 > k then goto 12.
   /* If f is added into P,
      there causes a fundamental curcuit
      of length label[e]+|F|-1 or more,
      which must be k+1 or less          */
8: Q := Qu{e}, P := Pu{F-{e,f}).
9: label[f] ← label[e]+|F|-2.
10: Remove F-{f} and all isolated vertices from G.
  /* f becomes an outer edge in the resultant graph */
11: Goto 13.
12: Q ← Qu{f}, and remove f from the graph.
13: If there exists an unlabeled edge then goto 5.
14:  /* G is a simple circuit */
   Let F be the circuit,
   If there is an edge e which satisfies
   label[e]+|F|-3 < k
     then
        let e be any one of such edges,
        Q := Qu{e}, P := Pu(F-{e}),
        answer:='Yes'
     else
        answer:='No'.
15: Halt.
```

3.2 VALIDITY AND COMPLEXITY OF THE ALGORITHM

PROOF OF CORRECTNESS:

Suppose that step 5 of this algorithm is going to be executed. An edge is labeled if and only if the edge is an outer edge of the current graph. An outerplanar graph has a face with exactly one inner edge if it is not a simple circuit. If the current graph were a simple circuit, every edge would be labeled and so step 5 were not executed. (See step 1 and 13.) So there always exists a face satisfying the condition in step 5.

First, we prove that if answer='Yes' then P is a k-AS. It is enough if we show that

1) P is a spanning tree of G,

and that

2) each fundamental circuit of the tree P is of length at most k+1.

Proof of 1)

Before the first execution of step 5, two end vertices of any labeled edge are not connected by P because P is empty. When an edge f, say (u,v), is labeled

at step 9, u and v have not been connected by P yet. And there is a path in P from any vertex which has already been removed in step 10 to some vertex which has not been removed yet. This means that the union of any spanning tree of the current graph and P becomes a spanning tree of the original graph. And finally, step 14 is executed where a spanning tree of the current graph is added. Thus the output P is a spanning tree of G.

Proof of 2)

Assume that C is the longest fundamental circuit of P of length $\ell > k+1$ which is defined by e_1. Then, e_1 is an outer edge in the original graph, and hence label[e_1]=1. Let V(C) be the vertices of C and G'=(V(C),E') be a subgraph induced by V(C). Also let T=P∩E'(=C-{e_1}) and S=E'-T. It is clear that T is a spanning tree of G'. Then let $e_1, e_2, \ldots, e_t$ be the labeled edges of S, sorted such as label[e_i] > label[e_j] if i>j. Let C_i be the fundamental circuit defined by e_i with respect to P and ℓ_i be its length. Note that $C=C_1$. When e_i is labeled in step 9 (i.e. $f=e_i$), e is some e_j (i>j). Then label[e_i]=label[e_j]+|F|-2 and $\ell_i=\ell_j+|F|-2$, because $F=C_i \oplus C_j$ ($\oplus$ denotes exclusive-or of two sets). Then it is clear that $\ell_i=\ell+1-$label[e_i] by induction. So ℓ_t+label[e_t]$-2 = \ell-1 > k$. Then there are two cases that

a) C_t is a face which has exactly one unlabeled edge (Step 5),

and that

b) the graph is a simple circuit C_t (Step 14).

a) C_t satisfies the inequality of step 7, which contradicts the fact that $C_t-\{e_t\} \subseteq P$. b) C_t does not satisfy the inequality of step 14, which contradicts answer='Yes'. Therefore P is a k-AS.

Second, we prove the converse, that is the graph G does not have a k-AS if answer='No'. Suppose that graph G has a k-AS and answer='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 may be an integer t which satisfies the following conditions.

(CONDITIONS)

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

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

The updating from (P_t,Q_t) to (P_{t+1},Q_{t+1}) can occur at step 8 (case 1) or step 12 (case 2). If such a number t does not exist then there is a k-AS T such that $P_m \subseteq T$ and $Q_m \cap T=\phi$ (case 3). We prove that thay are all impossible.

(Case 1) Assume that such an updating from (P_t,Q_t) to (P_{t+1},Q_{t+1}) occurs at step 8. 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 a spanning 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, and so, clearly $T \cup \{f\}-\{e\}$ is also a k-AS. This contradicts the definition of t.

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

(Case 2) Assume that such an updating occurs at step 12. 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 causes T to contain a fundamental circuit of length more than k+1. This is a contradiction.

(Case 3) P does not connect two end vertices of a labeled edge of current graph. Hence $P_m \neq T$ because of the assumption of answer='No'. So $P_m \subseteq T$ and $Q_m \cap T=\phi$. Anyway, Step 14 must be executed. Then there is an edge f such that $f \in F$, $f \neq e$ and $T=P_m \cup (F-\{f\})$. But label[f]+|F|-3 $\geq$ k holds, and there is a fundamental circuit of length label[f]+|F|-1 which contradicts that T is a k-AS. ∎

TIME COMPLEXITY:

It is O(|E|) to identify $e \in E$ if it is an outer edge since it is equivalent to check if G/e is 2-connected which needs O(|E|). Therefore identification of all edges in step 2 needs $O(|E|^2)=O(|V|^2)$. Note that an outerplanar graph has O(n) edges.

Clearly step 1, 3, 4 and 14 have complexities of O(n). Now we consider the loop from step 5 to step 13.

Each time this loop is excuted, either step 10 or 12 must be executed and at least one edge is removed from the graph as a result. So this loop is repeated at most |E| times which is O(n).

We show that the time complexity of this loop is O(n). A face in step 5 is found by tracing outer edges within O(n) steps. Clearly, the complexities of the other steps (from 6 to 13) are O(n).

Thus the total complexity is $O(n^2)$.

ACKNOWLEDGEMENT

We are grateful to Dr. Makoto Imase of NTT for suggesting this field of research.

REFERENCES

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

[2] D. Peleg and J. D. Ullman: An Optimal Synchronizer for the Hypercube, Symp. on Principles of Distributed Computing pp. 77-85 (Aug. 1987).