SLAC-PUB-517
CGTM 55
OCTOBER 1968
(MISC)

AN ALGORITHM FOR FINDING BRIDGES — AND ITS EXTENSION*

Ira Pohl

Stanford Linear Accelerator Center
Stanford University, Stanford, California

ABSTRACT

In graph models, the problem of partitioning arises naturally in many areas. This requires finding a set of edges which disconnects the graph. In this note we give an efficient computational method for finding these edges. Especially of interest is the case when only one edge is needed.

(Submitted to the Journal of the ACM)

*Work supported by the U.S. Atomic Energy Commission.

INTRODUCTION

In communication networks [4], [5], segmented programs [3], and other structural models [6], [1], and important problem is decomposing the representation and identifying especially crucial linkages. When these problems are described as graphs, a set of edges that upon removal disconnects the graph is a cut-set. If we are interested in finding a natural point in a program for segmentation, a place of interest would be where a minimum number of disconnections would be necessary. Obviously if the flow is through a single link, this link would be ideal. The problem of finding minimum cut-sets is therefore of interest in a large variety of problems. Especially interesting is the case of a single edge, called a bridge, which disconnects the graph (see Fig. 1). The approach is similar to that of other computer scientists [7], [8], in emphasizing computational ease and simplicity in describing an efficient method for finding bridges and other cut-sets.

TERMINOLOGY

We will be concerned with undirected graphs*. A graph $G = (X, E)$ is a collection of nodes X, and edges E, where the edges are unordered pairs of elements from X. For notational convenience, the node set is mapped one-one onto the integers 1, 2, ..., n where $n = |G|$ is the cardinality of the graph. The edges may now be written as pairs (i, j). A path from node i to node k in G is some sequence of nodes, $\mu = (i, j, \ldots, k)$ such that any consecutive pair of nodes in μ is an edge in E. A connected graph is a graph which has a path between all pairs of nodes in X. A circuit is a path from a node to itself which uses no edge more than once.

*To be called graphs throughout the remainder of this paper.

A _tree_ is a graph which is connected and has no circuits; this implies that its node set has one more node than its edge set cardinality.

The _degree_ of a node is the number of edges incident to it, i.e., the number of edges that contain it as an endpoint. A _rooted_ tree is a tree with one node designated the root, which has no predecessors. A _spanning_ tree of a graph G, is a tree T, which is a subgraph of G, with all nodes of G contained in T, written $X(T) = X(G)$.

A _cut-set_ of a graph is a set of edges which, when removed from the graph, leaves the graph unconnected. A _proper_ cut-set is a cut-set which has no proper subset which in turn is a cut-set. A _bridge_ is a cut-set of one edge, and is therefore identically a proper cut-set. A graph is called _h edge-connected_*, when h is the cardinality of its smallest (proper) cut-set.

PROBLEM AND SOLUTION

Find all the bridges in a graph. One can do this simply by removing each edge in turn and checking the remaining graph for connectedness. There are up to $n(n-1)/2$ edges in a loop-free undirected graph and this approach is obviously too tedious.

At this point let us note a simple theorem [4, p. 18].

> "Every spanning tree has at least one edge in common with every cut-set of a graph."

In particular, we note that any spanning tree must contain all bridges of the graph. Generating a spanning tree is a simple computation, and is on the average only twice the work of generating a path. Now in a dense graph there are many

* Berge [2] calls this _h-coherent_, but we will from now on refer to graphs as h-connected, meaning edge-connected.

spanning trees possible, and by suitably generating successive spanning trees and intersecting their edge sets, one should be left with only a smaller number of edges ($< n$) to check as bridges. This then is the method we outline below in detail.

SPANNING TREE ALGORITHM

1. Mark all nodes as unreached and unused.
2. Choose some node $i \epsilon X$ as the root node and mark it reached.
3. Select any node n that is reached but unused and mark it used.
4. Mark all nodes n_k, which are connected by an edge to n and not previously reached as reached. Include the edges (n, n_k) in the spanning tree.
5. If all the nodes in X are reached then halt, else go to step 3.

By selecting different root nodes and by suitably varying the order in which nodes are examined in step 3, a reasonably different sampling of spanning trees will be constructed, if possible. One simple possibility is to use reached nodes in ascending value and varying this by next choosing them in descending value. Also this algorithm is a test for connectedness, for if no reached but unused nodes exist at some stage before the computation halts, the graph must be unconnected.

BRIDGE FINDING ALGORITHM

1. Compute two spanning trees in different (as possible) ways.
2. Find the set of edges in the intersection of these two trees — set I.
3. If I is empty halt.
4. Take the first edge in I and delete it from the graph and from I.
5. Generate a new spanning tree (again try to make it different from the previous ones).

6. If the tree does not have all the nodes of the graph, then list the removed edge as a bridge. Otherwise, intersect the new tree with I to obtain the new I. Return to step 3.

Remark: At most n-1 spanning trees will be constructed, where this limit is attainable.

Pf. A spanning tree of a graph of size n has n-1 edges. Therefore set I can have at most n-1 edges initially. If the graph is just a simple circuit:

$$X = \{1, 2, 3, \ldots, n\}$$

$$E = \{(1,2), (2,3), \ldots, (n,1)\},$$

then the maximum number of intersections will be achieved.

If the graph of interest is dense, then there will be many possible different spanning trees. The intersection of two of these will leave but few candidate edges. Outside of an iteration required for each bridge found, the algorithm will normally need only a few intersections before all extraneous edges are discarded. In implementing the algorithm, the number of intersections stayed between three and five over a wide range of graph sizes and densities.

GENERALIZATION

The more general problem of finding the minimum proper cut-sets of a graph is a great deal more difficult. Methods based on the repeated use of the Ford-Fulkerson network flow algorithm [5] with edge capacities identically one, can be used. The fundamental result is that the maximum flow is equal to the minimum cut capacity and the Ford-Fulkerson algorithm may be programmed to find the cut-set. In the case of bridges, obviously the tree intersection algorithm requires substantially less work. While the Ford-Fulkerson algorithm is efficient, it is more complex than the simple tree spanning algorithm, and each iteration of it is about the same work as a complete spanning tree computation.

It is possible to generalize our method to cut-sets of higher order. Consider a cut-set of cardinality 2; name it C_2. By our theorem, each spanning tree must include one or the other edge of C_2. Therefore if k spanning trees are generated, some member of C_2 will appear more than k/2 times. If edges are investigated in order of number of occurrences (given that they appear $\geq$ k/2 times) the case of finding the other edge in C_2 is reduced to finding a bridge. This scheme seems more reasonable, especially in very dense graphs, than the more complex flow algorithm. The method, of course, is iteratively applicable to C_n with a criterion of k/n appearances. However, it is most reasonable for n small.

ACKNOWLEDGMENTS

The author wishes to thank C. Zahn of the Stanford Linear Accelerator Center, Z. Manna and W. F. Miller of Stanford University for their thoughtful reading and encouragement of this work.

REFERENCES

1. Amarel, S., "On Machine Representations of Problems of Reasoning About Actions — The Missionaries and Cannibals Problem," Carnegie Institute of Technology, (June 1966).

2. Berge, C., The Theory of Graphs and Its Applications, (Methuen and Co., Ltd., London, 1962).

3. Berztiss, A., "A Note on Segmentation of Computer Programs," Information and Control, Vol. 12, (January 1968) pp. 21-22.

4. Busacker, R. and T. Saaty, Finite Graphs and Networks: An Introduction with Applications, (McGraw-Hill Co., New York, 1965).

5. Ford, L. and D. Fulkerson, Flows in Networks, (Princeton University Press, Princeton, 1962).

6. Harary, F., R. Normal, and D. Cartwright, Structural Models: An Introduction to the Theory of Directed Graphs, (John Wiley and Sons, Inc., New York, 1965).

7. Ramamoorthy, C., "Analysis of Graphs by Connectivity Considerations," Journal of the ACM, Vol. 13, No. 2, (April 1966) pp. 211-222.

8. Warshall, S., "A Theorem on Boolean Matrices," Journal of the ACM, Vol. 9, No. 1, (January 1962) pp. 11-12.

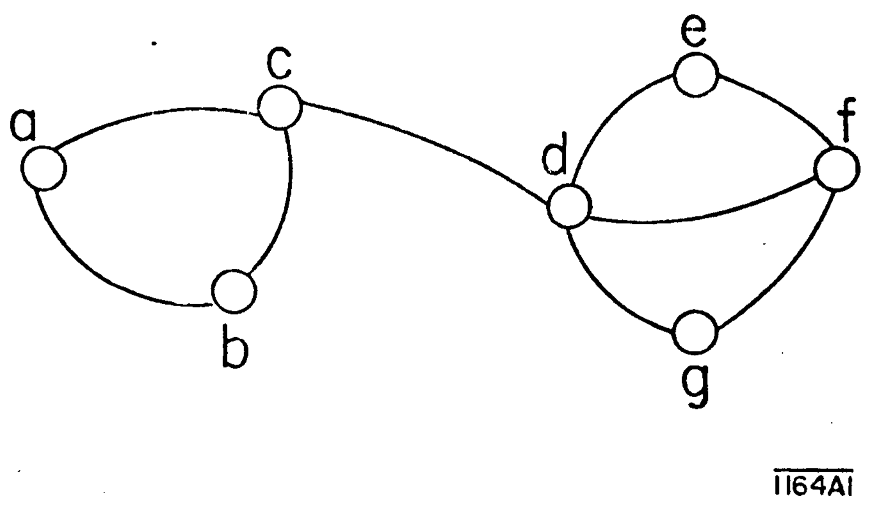

Fig. 1

Edge (c, d) is a bridge.