AUGMENTATION PROBLEM

by

Kapali P. Eswaran and R. Endre Tarjan

Memorandum No. ERL-M441

January 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AUGMENTATION PROBLEMS [*]

Kapali P. Eswaran
IBM Research Laboratories
San Jose, California.

R. Endre Tarjan
Computer Science Division
University of California
Berkeley, California.

January, 1974.

ABSTRACT

This paper considers problems in which the object is to add a minimum-weight set of edges to a graph so as to satisfy a given connectivity condition. In particular, simple characterizations of the minimum number of edges necessary to make a directed graph strongly connected and to make an undirected graph bridge-connected are given. Efficient algorithms for finding such minimum sets of edges are presented. It is shown that the weighted versions of these problems are NP-complete.

Keywords: algorithm, augmentation, bridge connectivity, connectivity, graph, strong connectivity, polynomial completeness.

* This work was partially supported by the Naval Electronic Systems Command, Contract N000 39-71-C-0255, by the National Science Foundation, Contract No. NSF-GJ-35604X, and by a Miller Research Fellowship.

AUGMENTATION PROBLEMS

Kapali P. Eswaran
and
R. Endre Tarjan

Introduction:

A common computational problem in graph theory is that of determining how many vertices or edges must be removed from a graph in order to satisfy some connectivity property. For instance, we might ask how many vertices must be removed from a graph to disconnect it. If the answer is zero, the graph itself is disconnected; if the answer is one or more, the graph is connected; if the answer is two or more, the graph is biconnected, and so on. Many good algorithms have been developed for solving such problems [1,2,3].

We can turn this idea around and ask questions about how many edges must be added to a graph to satisfy a given connectivity property. Several well-known problems, and also several heretofore unstudied ones, may be stated in this form. This paper proposes a theoretical framework for studying such augmentation problems, gives well-known examples of such problems, and analyses in detail the strong connectivity and bridge-connectivity augmentation problems.

Definitions:

A (finite) graph $G = (V,E)$ is a finite set of <u>vertices</u> V and a finite set of <u>edges</u> E. The edges of a graph are either ordered pairs (v,w) of distinct vertices (the graph is directed) or unordered pairs (v,w) of distinct vertices (the graph is undirected). A directed edge (v,w) has head w and tail v, enters w, and leaves v. V is the number of vertices and E is the number of edges in G. If $\mathcal{E}$ contains all possible edges, then G is complete. A subgraph $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ of G is a graph such that $\mathcal{V}_1 \subseteq \mathcal{V}$, $\mathcal{E}_1 \subseteq \mathcal{E}$. G_1 is spanning if $\mathcal{V}_1 = \mathcal{V}$.

A path p from v_1 to v_n is a sequence of edges

$(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$.

A path is closed if $v_1 = v_n$.

A cycle is a closed path such that $v_1, v_2, \ldots, v_{n-1}$ are all distinct. A path may contain no vertices, but a cycle must contain at least two vertices. A cycle is Hamiltonian if it is spanning. An undirected graph is connected if there is a path between every pair of vertices. If x is a vertex of G such that there are vertices v, w $\neq$ x for which every path from v to w contains x, and there is a path from v to w, then x is a cutnode. If G is connected and contains no cutnodes, it is biconnected. If (x,y) is an edge such that there are vertices v, w for which every path from v to w passes through (x,y) and there is a path from v to w, then (x,y) is a bridge. (An edge is a bridge if and only if it is contained in no cycles.) If G is connected and contains no bridges, it is bridge-connected. A directed graph is strongly connected if, for all pairs of vertices v and w, there is a path from v to w. The connected (biconnected, bridge-connected, strongly connected) components of a graph are its maximal connected (biconnected, bridge-connected, strongly connected) subgraphs.

A tree is an undirected, connected, acyclic graph. A spanning tree of a graph G is a spanning subgraph which is a tree. An arborescence is an acyclic directed graph with one vertex, the root, having no entering edges, and all other vertices having exactly one entering edge. A spanning arborescence of a directed graph G is a spanning subgraph which is an arborescence.

Let C be a connectivity condition on either directed or undirected graphs. We will assume: (A) if G_1 is a subgraph of G_2 and G_1 satisfies C, then so does G_2; that is, adding extra edges to G_1 cannot make C false if it is true in G_1. Consider a set of vertices V and a real-valued cost function f(v,w) defined on all possible edges between vertices of V. We will allow f to have value infinity on some edges. The (weighted) augmentation problem with respect to C is the problem of determining a subgraph G of the complete graph on vertex set V such that G satisfies C and the total cost of G's edges is minimum. Because of condition (A), we may assume without loss of generality that f is non-negative. Let G* be the graph with vertex set V and all edges having finite cost. If G* satisfies C we may restrict our attention to G*; if not, every subgraph satisfying C has infinite cost. If f is a 0 - 1 valued function, the augmentation problem is unweighted. In this case, if G_o is the graph whose edges are those of cost zero, then the problem is to determine a graph with a minimum number of edges which contains G_o and satisfies C.

One class of hard-to-solve problems deserves special attention. This is the class of NP-complete problems, as studied by Cook [4], Karp [5], and others. Let Σ^* be the set of all finite strings of 0's and 1's. A subset of Σ^* is a language. Let $\mathcal{P}$ ($\mathcal{NP}$) be the class of languages recognizable in polynomial time on one-tape deterministic (non-deterministic) Turing machines. Let Π be the class of functions from Σ^* into Σ^* computable in polynomial time by deterministic one-tape Turing machines. If L and M are languages, we say L is reducible to M if there is a function $f \in \Pi$ such that $f(x) \in M$ iff $x \in L$. A language L is NP-complete if (1) $L \in \mathcal{NP}$ and (2) every language in $\mathcal{NP}$ is reducible to L.

Reducibility is transitive, so in order to show (2) for a given language L we need only show that some known NP-complete problem is reducible to L. The NP-complete problems are computationally related in the sense that their time bounds are polynomial functions of each other; that is, either all these problems have polynomial-time algorithms, or none do. Such famous problems as the tautology problem, the traveling salesman problem, and the chromatic number problem are all NP-complete [5].

Examples of Augmentation Problems:

Several well-known problems are augmentation problems. For instance, suppose we consider undirected graphs, and C is "a given vertex x is connected to a given vertex y." The resulting augmentation problem asks for the minimum cost path in G* between x and y. Efficient algorithms for solving this problem are discussed in [6]. If C is "G is connected", the augmentation problem asks for the minimum-cost spanning tree in G*. Various researchers have developed algorithms to find minimum-cost spanning trees [7].

An analogous problem for directed graphs uses as C: "all vertices in G are reachable from some single vertex". The augmentation problem then asks for a minimum-cost spanning arborescence in the graph G*. Edmonds [8] has proposed an algorithm for this problem; the algorithm is quite efficient if implemented properly [9].

Eswaran [10] has considered the problem of adding a minimum-cost set of edges to a directed graph G_o so that there is a cycle which contains all the edges of G_o. He gives efficient algorithms for solving both the weighted and the unweighted version of this problem.

Not all augmentation problems have efficient algorithms. Consider undirected graphs: let $S \subseteq V$, and suppose C is " there is a path in G between any two vertices in S." This augmentation problem is the Steiner tree problem, it asks for the minimum-cost tree containing all vertices in S; other vertices may or may not be included. Karp [5] has shown that the Steiner tree problem is NP-complete.

A much more general kind of augmentation problem has been studied by Frank and Chou [12]. Given a $V \times V$ symmetric matrix $[r_{ij}]$ with $r_{ii} = 0$ for all i, they ask for a graph with fewest edges such that there are r_{ij} edge-disjoint paths between vertices i and j. They provide an efficient algorithm for solving this version of the augmentation problem, but they give no time bound. The weighted version of this problem is polynomial complete (as we shall see). They do not discuss what happens in the unweighted problem when some of the edges are predetermined; presumably the problem becomes much harder. Here we shall consider the special case when $r_{ij} = 2$ for all $i \neq j$ and some of the edges are predetermined.

Strong Connectivity

Let us now look at a few augmentation problems which have not been studied before. Consider directed graphs and suppose C is "G is strongly connected." This augmentation problem is intermediate in difficulty between the minimum spanning tree problem and the Steiner tree problem; its weighted version is polynomial complete, but its unweighted version has a nice solution.

Theorem 1:

Let V be a set of vertices, f a cost function on ordered pairs of distinct vertices, and F a total cost. The problem of determining whether there exists a set of edges with cost F or less which strongly connect the vertices of V is polynomial complete.

Proof:

(1) It is easy to construct a non-deterministic Turing machine which will guess a set of edges of cost F or less and check whether it strongly connects the vertices of V in polynomial time. Thus, the strong connectivity augmentation problem is solvable on a non-deterministic polynomial-time-bounded Turing machine.

(2) We prove that the directed Hamiltonian cycle problem (which is polynomial complete [5]) is reducible to the strong connectivity augmentation problem. Let $G = (V,E)$ be a directed graph. Construct a strong connectivity augmentation problem on vertex set V with

costs $f(v,w) = 1$ if $(v,w) \in E$, $f(v,w) = 2$ otherwise, and $F = V$. This augmentation problem has a solution with cost F or less if and only if G contains a Hamiltonian cycle. This construction is obviously computable in polynomial time, so the weighted strong connectivity augmentation problem is complete. Q.E.D.

It should be noted that only the most general version of this problem is complete. For instance, if G_o has the property that some vertex x can be reached from every vertex, then the strong connectivity augmentation problem reduces to that of finding a minimum weight spanning arborescence with root x, which is efficiently solvable using Edmond's algorithm [8,9].

Let's restrict our attention to the unweighted version of the strong connectivity augmentation problem. We are given a graph G_o to which we want to add a minimum number of edges to form a graph G which is strongly connected. We may reduce this problem by converting G_o into a simplified directed graph G_o' which contains one vertex for each strongly connected component of G_o. Two vertices representing components are joined by an edge in G_o' if there is an edge from one component to the other in G_o. The reduced graph G_o' is well-defined, acyclic, and may be constructed in $O(V+E)$ time by using depth-first search [1] to find the strongly connected components of G_o.

<u>Lemma 2</u>:

If A is a set of edges which when added to G_o strongly connects G_o, Let $A' = \{(X,Y) | (v,w) \in A$, X is the strong component of G_o containing v, and Y is the strong component of G_o containing w$\}$. Then A' strongly connects G_o'. Conversely, if A' is a set of edges which strongly connect G_o', form a set A containing one edge (v,w) such that $v \in X$ and $w \in Y$ for each edge (X,Y) in A', where X and Y are strong components of G_o. Then A strongly connects G_o.

Proof: Obvious.

Because of Lemma 2 we can restrict our attention to the acyclic graph G_o' . This graph contains zero or more sources (vertices with no entering edges and one or more exiting edges), zero or more sinks (vertices with no exiting edges and one or more entering edges), and zero or more isolated vertices (no entering or exiting edges). The following theorem gives a lower bound on the number of augmenting edges needed to make G_o' strongly connected.

Theorem 3:

Let G_o' be an acyclic directed graph with s sources, t sinks, and q isolated vertices. Then at least $\max(s,t) + q$ edges are needed to make G_o' strongly connected.

Proof:

After G_o' is made strongly connected, each source and each isolated vertex must have at least one entering edge. Thus, at least $s + q$ edges must be added. Similarly, each sink and each isolated vertex must have at least one exiting edge. Thus, at least $t + q$ edges must be added. Combining these two results gives the theorem.

The bound in Theorem 3 is attainable; the rest of this section gives an efficient algorithm for finding a set of max(s,t) + q edges which will strongly connect G'_o. First, we simplify the graph still further. We construct a graph G''_o which contains only the sources, sinks, and isolated vertices of G'_o and which only has edges from the sources to the sinks. To construct G''_o, we explore G'_o, moving out from the sources, and label each vertex with the name of one source from which it is reachable. (The isolated vertices never get labeled.) Then we explore G'_o moving backward along edges from the sinks, labelling each vertex with the name of one sink reachable from it. G''_o contains as edges all (v ,w) such that source v is labeled with sink w or sink w is labeled with source v. The total time required to construct G''_o is O(V+E) using any search method.

Lemma 4:

Let A be an augmenting set of edges which strongly connects G''_o. Then A strongly connects G'_o.

Proof:

Let v and w be any vertices in G'_o. If neither v nor w is isolated, there is a path in G'_o from v to some sink x, and a path in G'_o from some source y to w. Every edge in G''_o corresponds to a path in G'_o. Hence, since A strongly connects G''_o, there is a path from x to y consisting of edges in A and edges in G''_o. Thus, there is a path from v to w when the edges A are added to G'_o. Similar arguments work if either v or w is an isolated vertex. Thus, A strongly connects G'_o.

Since G_o'' has the same number of sources, sinks, and isolated vertices as G_o', we may concentrate on G_o''. To strongly connect G_o'', we apply a reduction rule:

(R) Pick any source v such that any vertex x with (v,x) an edge has two or more entering edges. Pick any sink w such that any vertex y with (y,w) an edge has two or more exiting edges. Add (w,v) as an augmenting edge and discard v, w, and all incident edges from G_o'' to form a smaller graph G_o''.

Graph G_o''' has one less sink, one less source, and the same isolated vertices as G_o'''. Furthermore:

<u>Lemma 5</u>:

If A is a set of edges which strongly connects G_o''', then $A \cup \{(w,v)\}$ strongly connects G_o''.

<u>Proof</u>:

Let the edges $A \cup \{(w,v)\}$ be added to G_o''. Since v has two or more outgoing edges in G_o'', one of them must lead to a vertex other than w. Thus, there is a path from v to some vertex in G_o''' and hence to any vertex in G_o''' when set A is added. Similarly there is a path from any vertex in G_o''' to w when set A is added. Finally, the edge (w,v) gives a path from w to v. It follows that $A \cup \{(w,v)\}$ strongly connects G_o''.

We continue applying (R) to smaller and smaller graphs, building up a set of augmenting edges, until (R) is no longer applicable. Then either each source v has an edge (v,x) to a sink with only one entering edge or each sink w has an edge (y,w) from a source with only one exiting edge. The two situations are symmetric; let's consider the latter. In this case the number of sinks must be no greater than the number of sources, and we can find a set of edges $(s_1,t_1),\ldots,(s_n,t_n)$ in the remaining graph such that $t_1 \ldots t_n$ are all the remaining sinks and $s_1 \ldots s_n$ are all distinct. Suppose $m - n + q \geq 1$, and $v_{n+1} \cdots v_{m-n+q}$ are the remaining sources and isolated vertices. We can strongly connect the remaining graph by adding the $m + q$ edges (t_i,s_{i+1}) for $1 \leq i < n$, (t_n, v_{n+1}), (v_i,v_{i+1}) for $n + 1 \leq i < m - n + q$, and (v_{m-n+q}, t_1). If $m - n + q = 0$, we can strongly connect the graph by adding the m edges (t_i, s_{i+1}) for $1 \leq i < n$ and (t_n, s_1). Combining the appropriate one of these sets of edges with the set of edges found using (R), we get a minimum augmenting set of edges for G''_o, which by Theorem 3 and Lemma 4 also augments G'_o minimally. If $n > m$, a corresponding construction works. Using Lemma 2 we can transform this set into a set of edges which minimally augments G_o, and we are done. The entire augmentation algorithm is presented below in Algolic notation.

```
Procedure SCONNECT begin
  S1: Use depth-first search to find strong components of graph $G_o$.
          Construct acyclic graph $G'_o$ with one vertex representing
          each component of $G_o$. Label each vertex of $G'_o$ with the
          name of some vertex in $G_o$ contained in the corresponding
          component. Find sources, sinks, and isolated vertices in $G'_o$ ;
  S2:     Search graph $G'_o$ forward from sources, labeling each vertex
          reached with the name of one source from which it is reachable.
          Search $G'_o$ backward from sinks, labeling each vertex reached
          with the name of one sink reachable from it. Construct graph
          $G''_o$ with sources, sinks, and isolated vertices of $G'_o$ as its
          vertices, having as edges all $(v,w)$ such that $v$ is labeled
          with sink $w$ or $w$ is labeled with source $v$;
```

```
Count number of edges DI(v) entering and DO(v) leaving each vertex
   v in $G_o''$ ;
Put all sources v such that (v,x) an edge implies DI(x) $\geq$ 2 on list LS ;
Put all sources w such that (y,w) an edge implies DO(y) $\geq$ 2 on list LT ;
A:= $\emptyset$ ;
while LS $\neq \emptyset$ and LT $\neq \emptyset$ do begin
   delete a vertex v from LS and a vertex w from LT;
   add (w,v) to A;
   for each edge (v,x) with DI(x) = 2 do begin
      let (Z,x) be an edge with z $\neq$ v;
      delete z from LS if it is there;
   end;
   for each edge (y,w) with DO(y) = 2 do begin
      let (y,z) be an edge with z $\neq$ w;
      delete z from LT if it is there;
   end;
   delete v,w, and all incident edges from $G_o''$, updating DO and DI;
end;
if $G_o''$ contains an isolated vertex then
let x be some isolated vertex in $G_o''$ else x:=0;
y:=x ;
if LS = $\emptyset$ then while some source s exists in $G_o''$ do begin
   find some edge (s,t) in $G_o''$ with DI(t) = 1;
   add (y,s) to A;
   delete s, t, and all incident edges from $G_o''$;
   y:=t;
end else while some sink t exists in $G_o''$ do begin
   find some edge (s,t) in $G_o''$ with DO(s) = 1;
```

```
          add (y,s) to A;
          delete s, t, and all incident edges from $G''_o$;
          y:=t;
       end;
       for each remaining vertex v in $G''_o$ do begin
          add (y,v) to A;
          delete v from $G''_o$;
          y:=v;
       end;
       comment if x = 0 we must delete a fictitious edge in A;
       if x ≠ 0 then add (y,x) to A else begin
           find the (unique) edge $(0,z) \in A$;
           delete (0,z) from A;
           add (y,z) to A;
       end;comment set A is a minimum set of augmenting edges;
end;
```

Theorem 6:

SCONNECT requires O(V+E) time and storage space (if G_o has V vertices and E edges), to correctly find a minimum set of augmenting edges.

Proof:

Theorems 1 and 3, and Lemmas 2, 4, and 5 imply that SCONNECT correctly finds a minimum set of edges. To represent the problem graph, we use two lists A(v), and B(v), for each vertex v. List A(v) contains all vertices w such that (v,w) is an edge, and list B(v) contains all vertices u such that (u,v) is an edge. These representations require O(V+E) storage space. Storage space for DI, DO, A, LS, LT, and other variables is also O(V+E). Step S1 may be carried out in O(V+E) time using depth-first search applied to A(v) [1]. Step S2 may also be carried out in O(V+E) time using any kind of search applied to A(v) and B(v).

The remaining steps, which actually construct A, require time proportional to the number of vertices and edges in G''_o, since each vertex and edge in G''_o is examined a fixed number of times and each examination causes the program to perform a fixed number of steps, independent of G''_o. Since G''_o contains no more than V edges, the time to construct A is O(V). The total time required by SCONNECT is thus O(V+E).

Figures 1 - 3 show the application of this algorithm to a directed graph.

Bridge Connectivity

We might ask whether there are undirected graph problems similar to the strong connectivity augmentation problem. Indeed there are. Let C_1 be "G is bridge connected" and let C_2 be "G is biconnected". Both of the corresponding weighted augmentation problems are NP-complete, while their unweighted version have O(V+E) algorithms.

Theorem 7:

Let $\mathcal{V}$ be a set of vertices, f a cost function on unordered pairs of distinct vertices ,and let F be a total cost. The problem of determining whether there exists a set of edges with cost F or less which bridge connect the vertices of $\mathcal{V}$ is NP-complete.

Proof:

(1) It is easy to construct a non-deterministic Turing machine which solves the bridge connectivity augmentation problem in polynomial time.

(2) We prove that the undirected Hamiltonian cycle problem, which is NP-complete [5], is reducible to the bridge connectivity augmentation problem. Let $G = (\mathcal{V},\mathcal{E})$ be an undirected graph. Construct a bridge connectivity augmentation problem on vertex set $\mathcal{V}$ with costs $f(v,w) = 1$ if $(v,w) \in \mathcal{E}$, $f(v,w) = 2$ if $(v,w) \notin \mathcal{E}$, and $F = V$. This augmentation problem has a solution with cost F or less if and only if G has a Hamiltonian cycle. This construction is obviously polynomial-time, so the weighted bridge connectivity problem is NP-complete.

Theorem 8:

Let V be a set of vertices, f a cost function on unordered pairs of distinct vertices and let F be a total cost. The problem of determining whether there exists a set of edges with cost F or less which biconnect the vertices of V is NP-complete.

Proof:

The undirected Hamiltonian cycle problem is reducible to the weighted biconnectivity augmentation problem using the same transformation as in the Proof of Theorem 7.

These constructions can be improved to show that even if the graph G_o is connected, both the bridge connectivity and biconnectivity augmenation problems are NP-complete. Thus, these weighted augmentation problems are different from the weighted strong connectivity augmentation problem, for which an interesting special case has an efficient algorithm. However, the unweighted bridge connectivity and biconnectivity problems both have good algorithms. We consider the bridge connectivity problem here. Pecherer and Rosenthal have developed an algorithm for the biconnectivity problem [12].

Consider the bridge connectivity augmentation problem. A graph G of exactly two vertices cannot be made bridge connected unless we allow multiple edges in the augmented graph. Thus, for the moment, let's allow multiple edges in the augmented graph. We shall see later that we can eliminate multiple edges in all but the case when G_o has exactly two vertices.

To solve the bridge connectivity problem, we first find the bridge-connected components of G_o and shrink each to a single vertex to form a graph G_o'. This may be done in O(V+E) time using depth-first search to identify the bridges of G_o. G_o' is a set of trees; each edge in G_o' corresponds to a bridge of G_o and each vertex corresponds to a bridge component of G_o. Any set of edges which bridge connects G_o corresponds to a set of edges which bridge connects G_o' in an obvious way; thus we have:

Lemma 9:

If A is a set of edges which bridge connect G_o, let $A' = \{(X,Y) \mid (v,w) \in A$, X is the bridge component containing v, and Y is the bridge component containing w$\}$. Then set A' bridge connects G_o' . Conversely, if set A' bridge connects G_o', form a set A containing one edge (v,w) such that $v \in X$ and $w \in Y$ for each edge (X,Y) in G_o', where X and Y are bridge components of G_o. Then A bridge connects G_o'.

Proof:

Obvious.

Because of Lemma 9 we can concentrate on G_o'. G_o' contains zero or more vertices with exactly one incident edge, called pendants, and zero or more vertices with no incident edges, called isolated vertices. The following theorem gives a lower bound on the number of edges needed to make G_o' bridge connected.

Theorem 10:

Let G_o' contain p pendants and q isolated vertices. Then at least $\lceil p/2 \rceil + q$ edges are needed to make G_o' bridge connected, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

Proof:

After G_o' is bridge connected, each pendant will have at least one new incident edge, and each isolated vertex will have at least two new incident edges. Each new edge can satisfy at most two of these requirements. Thus, at least $\lceil p/2 \rceil + q$ edges are needed to bridge connect G_o'.

The bound in Theorem 10 is attainable. There are in fact several different ways to efficiently find a set of $\lceil p/2 \rceil + q$ edges to bridge connect G_o'. One is a direct method of pairing pendants analogous to our solution of the strong connectivity augmentation problem. Another more elegant approach is to convert the problem into a strong connectivity augmentation problem and use our already-formulated algorithm.

We need to add edges to G_o' so that every edge is in a cycle. Suppose we direct the edges of G_o' so that each pendant becomes either a source or a sink, every other non-isolated vertex is neither a source or a sink, and the number of sinks is $\lceil p/2 \rceil$. Let A be a minimum set of augmenting edges which strongly connects the resulting directed graph G_o''. Then A will have $\lceil p/2 \rceil + q$ edges, and every edge in G_o'' will be in a directed cycle. If we now ignore the directions on all the edges, A gives a set of edges which when added to G_o' cause every edge to be in an undirected cycle, so A is a minimum augmenting set for G_o'. Now all we need is a method to direct the edges of G_o' in a suitable way. We use the following algorithm:

```
Procedure DIRECT begin
    Choose a set P of $\lceil p/2 \rceil$ pendants of $G_o'$, with at least one
    pendant from each (non-trivial) connected component of $G_o'$ and so
    that $V$ - P contains at least one pendant from each (non-trivial)
    connected component of $G_o'$;
```

```
Direct the edges incident to P into P;
while some edge is undirected do begin
  if some non-pendant vertex v has one undirected incident edge e
    and all other incident edges are directed out of v, then begin
    direct e into v ;
        comment this is called a forced move;
  end else begin if some non-pendant vertex v has an undirected
        incident edge e and there exists some edge directed out of v then
        direct e into v;
        comment this is called an unforced move;
end; end; end;
```

Lemma 11:

When procedure DIRECT is applied to a set of trees G_o' with p pendants all the edges in G_o' become directed. Furthermore, the resulting directed graph G_o'' has $\lceil p/2 \rceil$ sinks and $p - \lceil p/2 \rceil$ sources.

Proof:

A satisfactory set P of pendants can always be chosen because any non-trivial tree contains at least two pendants. Pick any tree T in G_o' . Initially some edge in T is directed. Suppose DIRECT stops without directing all the edges in T. Let e = (v,w) be an edge remaining undirected. From w there is an (undirected) path p in T to some pendant whose incident edge was initially directed. Let e' = (v',w') be the last edge on this path remaining undirected (if there is no such edge let e' = e). Some edge incident to w' is directed. Because of the rules of

DIRECT, the first edge incident to a vertex (excluding the initial pendants) which becomes directed must be directed outward. Thus, some edge incident to w' is directed outward, a move is possible, and DIRECT is not finished. This contradiction guarantees that all the edges in G_0' eventually become directed.

Since every non-isolated vertex except the initial pendants has at least one outwardly directed edge, G_0'' contains exactly $\lceil p/2 \rceil$ sinks and at least $p - \lceil p/2 \rceil$ sources (the non-initial pendants). We must prove that G_0'' contains no sources which are not pendant. Suppose to the contrary that vertex v is a non-pendant source in G_0''. Vertex v must have become a source because some edge (v, w) was directed into w. Just before edge (v,w) was directed, at least two forced moves were possible, directing edge (v,w) in either direction. No non-forced move could take place before edge (v,w) was directed, since then either (v,w) would be directed the other way or vertex v would not become a source. Thus, all moves performed by DIRECT up to the time of directing (v,w) are forced.

At the time of directing (v,w), the already directed edges form an arborescence with root w, another aborescence with root v, and possibly some other arborescences. No undirected edges can be incident to any of the vertices in the aborescences containing v and w, since all moves forming them were forced. But the connected component of G_0' containing v and w also contains at least one pendant x in $V - P$ whose incident edge is in neither of the arborescences containing v and w. There is some (undirected) path in T from x to w, and some edge on this path must be undirected but incident to one of the arborescences containing v and w. This is impossible. Thus, G_0'' can contain no source except a pendant, and the Lemma is proved.

G'_o contains O(V) edges since it is a set of trees, and DIRECT can be easily implemented to run in O(V) time by keeping track of the number of directed and undirected edges incident to each vertex. The entire bridge connectivity augmentation algorithm is sketched below in Algolic notation.

```
procedure BRCONNECT begin
  BR1:  use depth-first search to find bridge components of graph $G_o$.
          Shrink each to a single vertex, labeled with the name of some
          vertex in the corresponding bridge component. Let the resulting
          set of trees be $G'_o$, containing p pendants;
  BR2:  choose a set P of $\lceil p/2 \rceil$ pendants of $G'_o$, with at least one in
          each non-trivial tree of $G'_o$ and such that $V$ - P contains at
          least one pendant in each non-trivial tree of $G'_o$. Direct each
          edge incident to a pendant of P into that pendant;
    for each vertex v let U(v) be a list of the undirected edges
          incident to v;
          comment MARK(v) = 0 means no edge is directed out of v,
                  MARK(v) = 1 means some edge is directed out of v,
                  MARK(v) = 2 means some edge is directed out of v and some
                              edge is directed into v,
                  MARK(v) increases as edges become directed;
    for v $\in V$ do MARK(v):=0;
    for (v,w) such that w $\in$ P do MARK(v):=1;
    Let $L_1$ be a list of vertices with MARK(v) = 1 and $|U(v)|$ = 1;
    Let $L_2$ be a list of vertices with MARK(v) $\geq$ 1 and U(v) $\neq \emptyset$;
    Comment Step BR3 executes all possible forced and unforced moves;
```

```
BR3: while some $U(v) \neq \emptyset$ do
        if $L_1 \neq \emptyset$ then begin let $v \in L_1$ and $(w,v) \in U(v)$;
            comment this is a forced move;
            $L_1 = L_1 - \{v\}$;
        end else begin let $v \in L_2$ and $(w,v) \in U(v)$;
        end; comment this is an unforced move;
        comment update MARK, U, $L_1$, $L_2$;
        MARK(v):=2;
        if MARK(w) = 0 then MARK(w):=1;
        $U(v):=U(v) - \{(w,v)\}$;
        $U(w):=U(w) - \{(w,v)\}$;
        direct (w,v) from w to v;
        if MARK(w) = 1 and $|U(w)| = 1$ then $L_1:=L_1 \cup \{w\}$;
        if $U(v) = \emptyset$ then $L_2 = L_2 - \{v\}$;
        if $U(w) = \emptyset$ then $L_2 = L_2 - \{w\}$;
     end;
     let the resulting directed graph be $G''_o$ ;

BR4: apply SCONNECT to $G''_o$ to find a minimum set A' of edges
     which strongly connect $G''_o$;
     Let A be the set of undirected edges corresponding to the directed
         edges in A';
end  comment A is a minimum set of edges which bridge connect $G'_o$
         and hence $G_o$;
```

Figures 4-5 show the application of this graph to an example.

Theorem 12:

BRCONNECT requires O(V+E) time and storage space to find a minimum set of edges which bridge connect a graph G_o.

Proof:

The results in Lemmas 9 and 11 and Theorem 10 guarantee that BRCONNECT correctly finds a minimum augmenting set of edges. Step BR1 requires O(V+E) time [1,2]. Step BR2 obviously requires O(V) time since graph G_o' has O(V) edges. Initialization for the process which constructs G_o'' also requires O(V) time. Only a fixed time is required before some edge is directed in Step BR3, so the time to construct G_o'' is O(V). Step BR4 requires O(V) time by the results in the previous section. Thus the total time required by BRCONNECT is O(V+E). BRCONNECT obviously requires O(V+E) storage space.

The augmenting set A produced by BRCONNECT may contain edges which are already in G_o; we want to eliminate such duplicate edges if possible. An inspection of SCONNECT reveals that when it is used to solve the bridge connectivity augmentation problem, it only produces duplicate edges when G_o' has exactly two vertices. If G_o' has exactly two vertices and G_o also has exactly two vertices, then G_o cannot be made bridge connected without using duplicate edges. If G_o has more than two vertices, one of the vertices in G_o' corresponds to two or more vertices in G_o, and we can easily modify the set A (which consists of either one or two edges) so that it doesn't duplicate edges in G_o. This trivial modification takes care of the problem of multiple edges.

It is also possible to construct an O(V+E) algorithm for finding a minimum set of edges to biconnect a graph, though the present authors know of no way to reduce this problem to the strong connectivity augmentation problem. A nice algorithm for solving the biconnectivity augmentation problem has been developed by Pecherer and Rosenthal [12].

REFERENCES

[1] R. Tarjan, "Depth-first search and linear graph algorithms", SIAM J. Comput., Vol. 1, No. 2, (June 1972), 146-160.

[2] J. Hopcroft and R. Tarjan, "Efficient algorithms for graph manipulation," Comm. A.C.M., Vol. 16, No. 6 (June 1973), 372-378.

[3] R. Tarjan,"Testing graph connectivity", to be presented at 5th Annual ACM Symposium on the Theory of Computing,Seattle,Washington,(May 1974).

[4] S. Cook, "The complexity of theorem-proving procedures", Proceedings Third Annual A.C.M. Symposium on the Theory of Computing, Shaker Heights, Ohio, (1971) 151-158.

[5] R. Karp, "Reducibility among combinatorial problems", Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds. Plenum Press, New York (1972), 85-104.

[6] D.B. Johnson, "Algorithms for shortest paths", Report TR 73-169, Department of Computer Science, Cornell University, (May 1973).

[7] A. Kershenbaum and R. Van Slyke,"Computing minimum spanning trees efficiently", Proceedings: 25th Annual Conference of the A.C.M. (1972), 518-527.

[8] J. Edmonds, "Optimum branchings", J. Res. Nat. Bur. Standards, Sect. B, Vol. 71, (1967), 233-240.

[9] R. Tarjan, "Analysis of algorithms for finding minimum spanning trees and optimum branchings", presented at the VIII International Symposium on Mathematical Programming, Stanford University (August 1973).

[10] K. Eswaran, "Representation of graphs and minimally augmented Eulerian graphs with applications in data base management", Report RJ 1305, IBM Research, Yorktown Heights, New York (November 1973).

[11] H. Frank and W. Chou, "Connectivity considerations in the design of survivable networks", IEEE Trans. on Circuit Theory, Vol. CT-17, No. 4 (November 1970), 486-490.

[12] R. Pecherer and A. Rosenthal, private communication, Computer Science Division, University of California, Berkeley (March 1973).

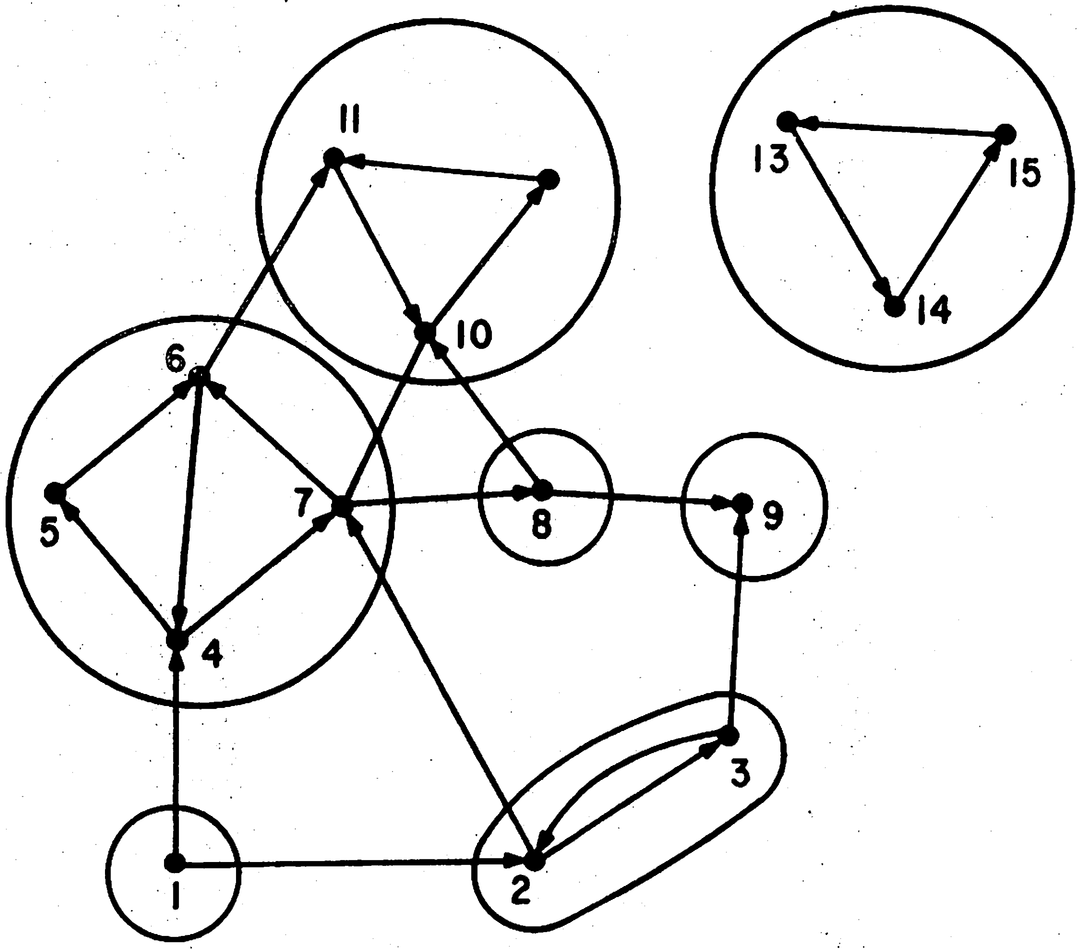

Figure 1: A directed graph G_o which we wish to make strongly connected. Strongly connected components are circled.

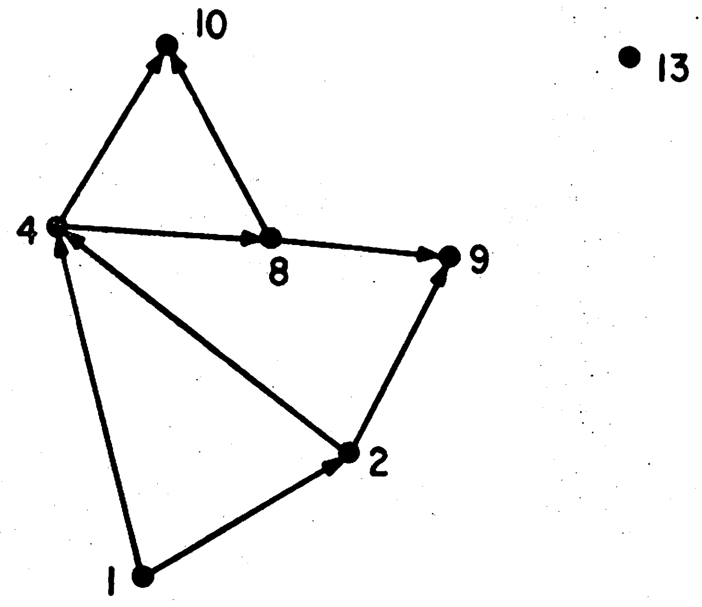

Figure 2: Acyclic directed graph corresponding to graph in Figure 1. Numbers are those of arbitrary vertices in the corresponding strongly connected components.

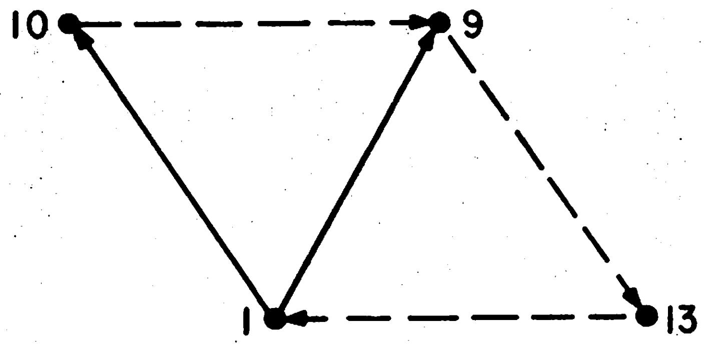

Figure 3: Source-sink graph generated from graph in Figure 2. Graph has one source, two sinks, and one isolated vertex. Dotted edges give a minimum strongly connecting augmentation, which also works for graph in Figure 1.

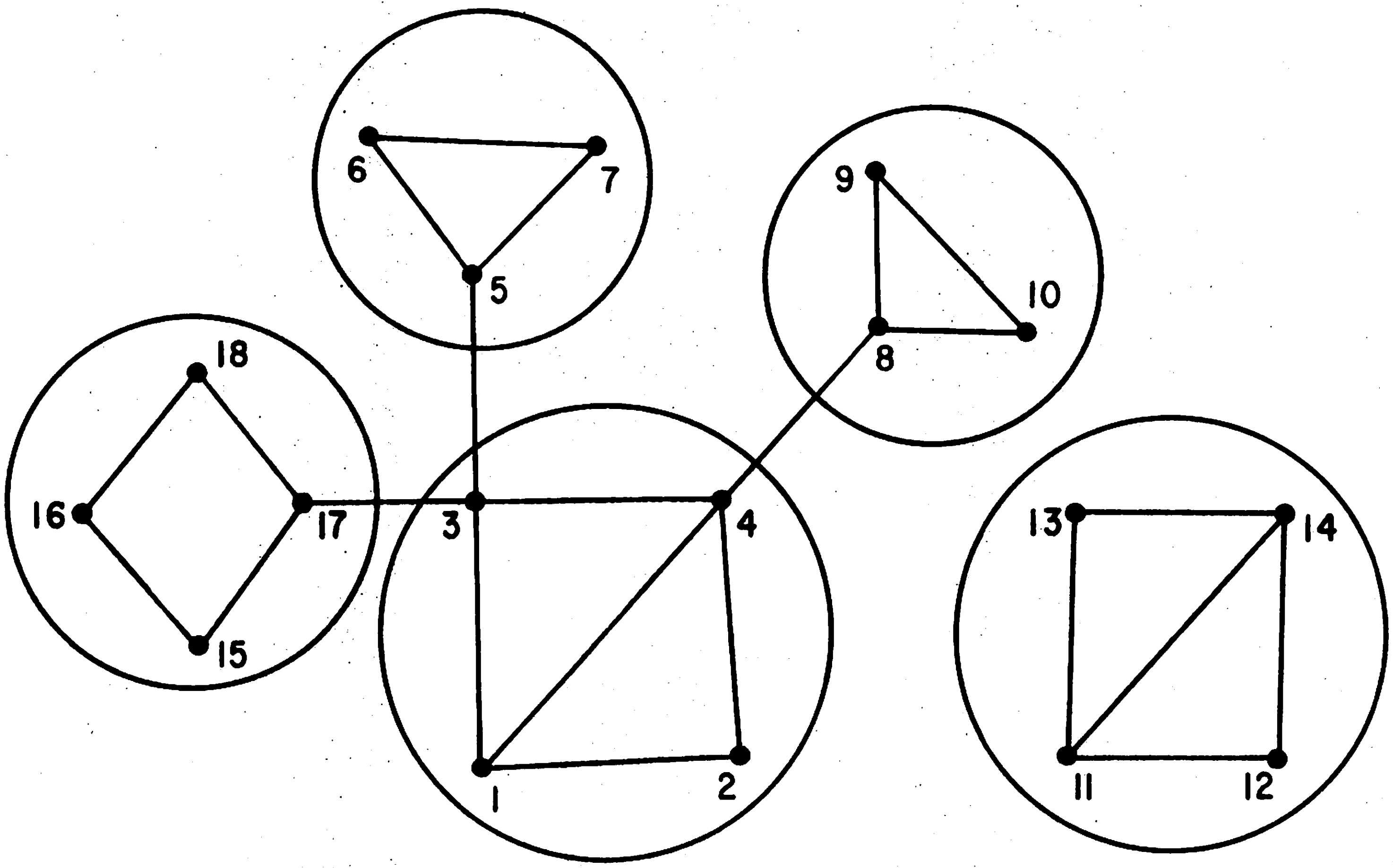

Figure 4: Undirected graph we wish to make bridge-connected. Bridge components are circled.

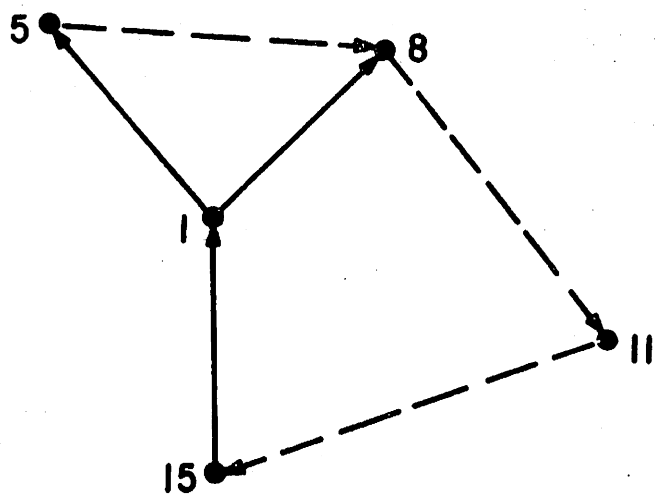

Figure 5: Set of trees corresponding to graph in Figure 4. Graph has three pendants and one isolated vertex. Arrows on edges are directions given by DIRECT. When directions on dotted edges are ignored, they give a minimum bridge connecting augmentation of this graph and the one in Figure 4.