THE SUBGRAPH HOMEOMORPHISM PROBLEM

Andrea S. LaPaugh and Ronald L. Rivest*
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

We investigate the problem of finding a homeomorphic image of a "pattern" graph H in a larger input graph G. We view this problem as finding specified sets of edge disjoint or node disjoint paths in G. Our main result is a linear time algorithm to determine if there exists a simple cycle containing three given nodes in G; here H is a triangle. No polynomial time algorithm for this problem was previously known. We also discuss a variety of reductions between related versions of this problem and a number of open problems.

I. Introduction

The subgraph homeomorphism problem (SHP) is to find a homeomorphic image of a "pattern" graph H in an input graph G. The images of certain nodes of H, which are nodes of G, may be specified a priori and the images of the edges of H, which are paths in G, may be required to be node-disjoint or edge-disjoint. Graphs G and H are either both directed or both undirected. Examples are finding a Kuratowski subgraph (K_5 or $K_{3,3}$) in a non-planar graph, finding certain kinds of network flows, finding Hamiltonian cycles, finding a simple cycle containing given nodes of G, and finding a set of disjoint paths connecting certain nodes of G. Subgraph homeomorphism problems also arise in the study of programming schema, since many schema properties are characterized by the presence or reachability of certain substructures [Hunt].

We summarize our research on the SHP as follows. We observe that the SHP is NP-complete if both H and G are given as input; this follows from the Hamiltonian circuit problem for node-disjoint SHPs and from the multi-commodity integral network flow problem for edge-disjoint SHPs. We therefore consider the various SHPs derived by fixing H. The main open question for such SHPs is, "For every pattern graph H, is there a polynomial time algorithm which, given an input graph G, will determine whether there is a homeomorhpic image of H occurring in G?" We can neither find a fixed H whose SHP is NP-complete nor demonstrate that every H has a polynomial-time algorithm for its SHP. We therefore concentrate on the problem of determining which pattern graphs H have polynomial-time algorithms. These problems turn out to be surprisingly difficult, even for very simple graphs H.

* This research was supported in part by NSF grant MC76-14294.

The main result of this paper is a linear-time algorithm which determines if there exists a simple cycle containing three given points of an undirected input graph G. This problem is an instance of the SHP where the pattern graph is a cycle of length three (a triangle). Although this problem has a simple pattern graph, no polynomial time algorithm was previously known. A linear time algorithm for another SHP with simple pattern graph--the two disjoint paths problem for undirected graphs--was recently found by Y. Shiloach. A third such problem--finding a cycle in a directed graph containing two given nodes--is still an open problem.

In Section II, we present definitions and a general discussion of the subgraph homeomorphism problem. In Section III we briefly sketch some reductions relating various versions of the SHP and in Section IV we present the linear time algorithm for the triangle problem. Section V is dedicated to a discussion of open problems and other work in the area.

II. Background

We assume the reader is familar with standard graph and network flow concepts [Ah][Hu]. Let H and G be two graphs, both directed or both undirected. A subgraph homeomorphism is formally defined as a pair of one-to-one mappings, (v,a), the first from nodes of H to nodes of G; the second from edges of H to simple paths of G. We require that a path in G which corresponds to edge (x,y) in H go from v(x) in G to v(y) in G. The graph H is called the pattern graph. If the image of the edges of H is a set of paths which are node disjoint up to endpoints, the homeomorphism is a node disjoint homeomorphism. We say that H is node disjoint homeomorphic to a subgraph of G, denoted $H \leq_N G$. If the set of paths is edge disjoint, then H is edge disjoint homeomorphic to a subgraph of G, denoted $H \leq_E G$.

We should note here that an alternate, but equivalent, definition of node disjoint homeomorphism has been used historically. In the alternate definition, $H \leq_N G$ if nodes can be inserted along the edges of H to yield a new graph H' which is isomorphic to a subgraph of G [Ha 1973]. We prefer our definition since defining a homeomorphism in terms of paths of G allows one to conceptualize the subgraph homeomorphism problem in terms of finding paths in G and to readily use the body of path-finding algorithms already in the literature.

The most general subgraph homeomorhpism problem--given as input graphs G and H, is H homeomorphic to a subgraph of G?--is NP complete for both node disjoint and edge disjoint homeomorphisms, and both directed and undirected graphs. For node disjoint homeomorphisms, this follows from the

Hamiltonian Circuit problem. Given the question, "Does G contain a Hamiltonian circuit?" we construct H such that the number of nodes in H is the same as the number of nodes in G, and the edges of H connect the nodes in a cycle. We then ask, "Is H homeomorphic to a subgraph of G?" This will be the case if and only if G contains a Hamiltonian circuit.

For edge disjoint homeomorphism, we use the result of Even, Itai, and Shamir that the two commodity integral network flow problem for unit edge capacities is NP-complete [Ev 1976]. Suppose we are given a directed network, N, with sources s_1 and s_2, sinks t_1 and t_2, and all edge capacities equal to one. We show how to reduce the question, "Are there simultaneous integral flows from s_1 to t_1 and s_2 to t_2 of values k_1 and k_2, respectively?" to an edge disjoint SHP for directed graphs. The question is equivalent to asking whether there is a set of $k_1 + k_2$ edge disjoint paths in N such that k_1 go from s_1 to t_1 and k_2 go from s_2 to t_2. Divide each edge out of s_1 or s_2 into two edges by inserting one new node on each edge. Now each edge disjoint path from s_1 or s_2 must have a distinct second node. We can model the $k_1 + k_2$ edge disjoint paths by a pattern graph, H, which has two source nodes, c_1 and c_2, two terminal nodes, d_1 and d_2, k_1 distinct length two paths from c_1 to d_1, and k_2 distinct length two paths from c_2 to d_2. The paths from c_2 to d_2 are node disjoint from the paths from c_1 to d_1. If we can modify the present network, producing a network N', to insure that c_1 maps to s_1, c_2 maps to s_2, d_1 maps to t_1, and d_2 maps to t_2, then H is edge disjoint homeomorphic to a subgraph of N' if and only if the desired flow exists in the original network, N. Let the original network have n nodes. To insure the desired node mapping, we add 4n new nodes to each of H and N' and edges from each of these new nodes to c_1 and s_1, respectively. Similarly, we connect 3n additional new nodes to c_2 and s_2, 2n additional new nodes to d_1 and t_1, and n additional new nodes to d_2 and t_2. Node c_1 must map to s_1, since s_1 is the only node in N' with indegree at least 4n. (Self-loops are not counted.) Node c_2 must map to s_2, since s_1 and s_2 are the only nodes in N' with indegree at least 3n and s_1 corresdponds to c_1. Continuing this reasoning, d_1 must map to t_1 and d_2 must map to t_2. We have found a pattern graph, H, with $10n + 4 + k_1 + k_2$ nodes and a modification of network N, N', such that $H \leq_E N'$ if and only if there are integral flows of two commodities in N with values k_1 and k_2.

Given the above NP-completeness results, we focus on the solution of problems where the pattern graph H is fixed. A graph G and possibly, a partial or total specification of the mapping from nodes of H to nodes of G are given as input. An example of a node disjoint homeomorphism problem is given in Figure 1. An algorithm to solve such a problem may depend on the pattern graph and the subset of nodes of the pattern graph on which the node mapping will be specified. We measure the time required by an algorithm to solve the problem on input G as a function of the size of G, i.e. the number of nodes in G plus the number of edges in G.

The variety of properties characterized by subgraph homeomorphisms and the applications of these properties motivate our interest in efficient algorithms for solving the SHP for fixed pattern graphs. Ultimately, we would like to know if all subgraph homeomorphism problems with fixed pattern graphs can be solved in time polynomial in the size of the input graph. This question was in fact proposed by Hunt et. al. in relation to programming schema and their substructures [Hunt]. The question appears to be quite diffficult. We have concentrated on two research areas in the hope of learning more about the answer:

i) Methods of reducing one SHP to another.

i) The solution of the SHP for particular pattern graphs, particularly for node disjoint homeomorphism (since edge disjoint homeomorphism problems are reducible to node disjoint homeomorphism problems as discussed in the next section.)

Our polynomial time algorithm for finding a cycle containing three given nodes of a graph is our contribution to the second line of research above. Other contributions are discussed in Section V.

III. Reductions

The reductions which we will present are of two types: those which relate edge disjoint SHP's to node disjoint SHP's and those which reduce a node disjoint SHP for a particular pattern graph to a node disjoint SHP for another pattern graph. Each reduction takes a pattern graph, H, an input graph G, and a partial specification of the node mapping. It produces a pattern graph, H', and a set of graphs, G', with corresponding partial specifications of the node mapping. The pattern graph H is homeomomphic to a subgraph of G if and only if H' is homeomorphic to a subgraph of one of the graphs in the set G'. The construction of H' depends only on H and is independent of G. All constructions will take at most polynomial time in the sizes of H and G. Finally, the number of graphs in the set G' is at most polynomial in the size of G. Note that the reductions we are using correspond to the notion of Turing reducibility [Rog]. Given an instance of the SHP for pattern graph H, we may not be able to find just one instance of the SHP for pattern graph H' whose solution corresponds to the solution of the first SHP. However, we can use a given procedure to solve the SHP for pattern graph H' as a subroutine, to solve the SHP for each graph in G', and thereby determine whether a homeomorphic image of H exists in G.

To simplify the statement of reduction results, we use the term fixed SHP when the node mapping is known a priori.

III.1 Edge Disjoint Homeomorphism versus Node Disjoint Homeomorphism

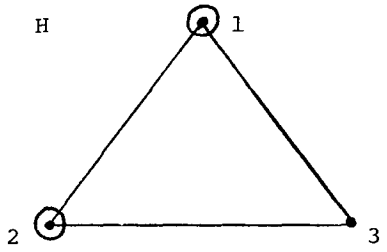
Within the reductions which relate node disjoint homeomorphism to edge disjoint homeomorphism, we have the following lemmas:

Lemma 1: Any fixed node disjoint SHP for directed graphs is reducible to a fixed edge disjoint SHP for directed graphs.

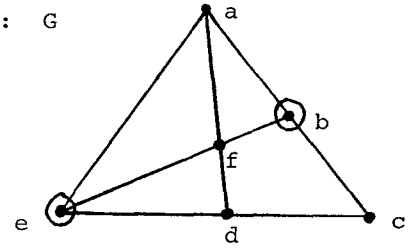
Proof: The construction used is analogous to that for changing vertex capacities to edge capacities in network flow problems [Ta 1974]. Suppose we have a pattern graph, H, and a input graph, G. The reduction is accomplished by replacing each node, x, in G by two nodes, HEAD(x) and TAIL(x). A directed edge is added from HEAD(x) to TAIL(x). All edges of G into x become edges into HEAD(x), and all edges of G out of x become edges out of TAIL(x). We now have a new directed graph, G'. A new pattern graph, H', is constructed analagously. If, in the original problem, node h of H was mapped to node g of G, then the node HEAD(h) of H' maps to HEAD(g) of G', and TAIL(h) maps to TAIL(g). A straightforward argument left to the reader proves $H \leq_N G$ if and only if $H' \leq_E G'$.

Figure 1: Example of a node disjoint SHP

Fixed: H



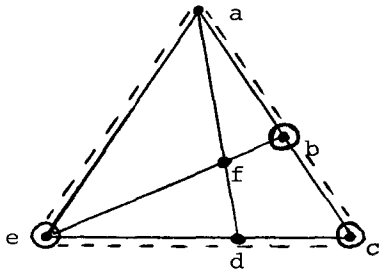
Input: G



with partial specification:

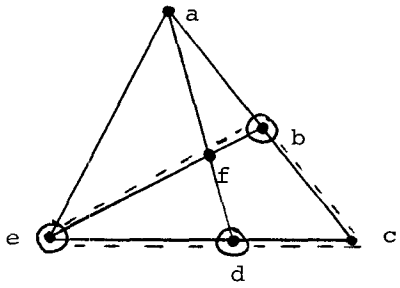
v(1) = b
v(2) = e

Solution 1:



v(3) = c
(1,3) -- <(b,c)>
(3,2) -- <(c,d),(d,e)> mapping of
(2,1) -- <(e,a),(a,b)> edges in H

Solution 2:



v(3) = d
(1,3) -- <(b,c),(c,d)>
(3,2) -- <(d,e)> mapping of
(2,1) -- <(e,f),(f,b)> edges in H

In this example, the node mapping, v, is partially specified.

⊙ indicates a node whose image or inverse image under v is known.

Lemma 2: Any edge disjoint SHP for directed (undirected) graphs is reducible to a node disjoint SHP for directed (undirected) graphs.

Proof: The reduction replaces each node in the input graph, G, by a "switch". Let the nodes on a path other than the endpoints of the path be called interior nodes. Edge disjoint paths in G containing the same interior node pass through node disjoint paths within the switch corresponding to that node.

When a fixed SHP is being considered, no modification to the pattern graph is necessary. The input graph is modified using switches as follows. In undirected graphs, switches are cliques the size of the degree of the corresponding node. In directed graphs, switches are bipartite graphs with one set of nodes the size of the indegree of the corresponding node and one set of nodes the size of the outdegree of the corresponding node. All possible edges from nodes of the indegree set to nodes of the outdegree set are present in the switch. Each edge to a node, x, in the original input graph now goes to a distinct node in the switch for x. In the directed case, edges into x become edges into a node of the indegree set and edges out of x become edges out of a node of the outdegree set. If node x of the input graph corresponds to a node, h_x, in the pattern graph, then there is a distinct node, x', in the modified input graph in addition to the switch for x. In the new node mapping, h_x maps to x'. In addition to the edges between nodes of the switch for x, there is an edge connecting node x' to each node in this switch. In the directed case, these edges are directed from nodes of the indegree set and to nodes of the outdegree set.

When the subgraph homeomorphism problem is not fixed, more care need be taken so that we will not create an instance of the pattern graph within a switch. Therefore, the switches are required to have nodes of fixed small degree while the "original" nodes of both the pattern graph and the input graph are guaranteed to have larger degree by adding extra adjacent nodes. Suppose we have a directed edge disjoint SHP. Consider a node y of the input graph, G. Let y have indegree IN_y and outdegree OUT_y. To create the switch for y, insert OUT_y nodes "close to" y on each edge into y and IN_y nodes "close to" y on each edge out of y. Each original edge into y is connected to all original edges out of y by adding an edge from each inserted node on the incoming edge to an inserted node on an outgoing edge. Each inserted node is used for exactly one interconnection. Therefore, nodes inserted on incoming edges have indegree one and outdegree two; nodes inserted on outgoing edges have indegree two and outdegree one. Each edge, (u,t), in G has become a path from u, through the IN_u nodes inserted near u followed by the OUT_t nodes inserted near t, to t. A path in the original graph, G, which uses a node y as an interior node can bypass y in the modified input graph by using an interconnecting edge in the switch for y. To insure that nodes of the pattern graph do not map to inserted nodes of the new input graph, we do the following: For each node of the pattern graph and each node of the original input graph we add three nodes to the pattern graph and the new input graph, respectively. A new edge is directed to each new node from the node for which it was added. In the resulting pattern graph, all original nodes have outdegree at least three. In the resulting input graph, all nodes which were nodes in the original input graph have outdegree at least three. All inserted nodes still have outdegree one or two, and, therefore, cannot correspond to nodes of the original pattern graph.

Given an undirected edge disjoint SHP, we construct the new pattern graph and input graph in the same manner as above. We insert $d_x - 1$ nodes on each edge to a node x of G, where d_x is the degree of x, and make the interconnections. Each inserted node will have degree three. Therefore, we add four new nodes and edges for each node of the pattern graph and each node of the original input graph to insure that nodes of the original pattern graph map to nodes of the original input graph. Figure 2 illustrates the construction used for undirected graphs.

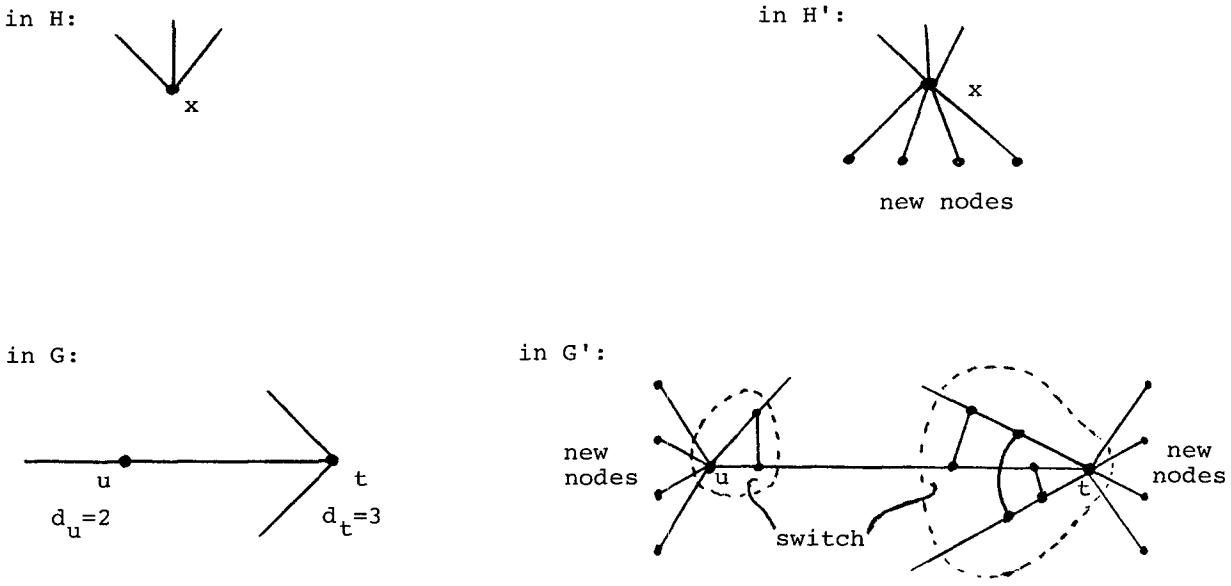
III.2 Reductions among Node Disjoint Subgraph Homeomorphism Problems

Since we are not measuring the time taken by an algorithm as a function of the size of the pattern graph, we can solve any SHP for a particular pattern graph in polynomial time if we can solve the fixed SHP for the pattern graph in polynomial time. This is accomplished by solving the fixed SHP for each node mapping consistent with any previously given partial specification of the mapping. This technique results in an algorithm which is polynomial in the size of the input graph, where the order of this polynomial may be the size of the vertex set of the pattern graph. This reduction is not very appealing for two reasons. First, the exponent may be large. Second, the reduction yields no simplification of the pattern graph itself. On the other hand, the two reductions discussed below do simplify the pattern graph but can only be used when certain subgraphs exist within the pattern graph. The reductions are presented in terms of directed graphs, but completely analogous reductions exist for the undirected case.

Reduction 1: The first reduction is applicable when H contains a path of length k>1 from one node, called the tail node of the path, to another node, called the head node. The correspondence between the tail and head nodes in H and nodes of the input graph, G, must be known. Also, all interior nodes on this path must have indegree one and outdegree one, and their correspondence to nodes in G must be unspecified. Number the interior nodes of the path in H from 1 through k-1, beginning at the tail node. We delete the first k-2 interior nodes on the path in H, producing a new pattern graph H'. In G, we find all k-1 length paths such that (i) the tail node of each path corresponds to the tail node in H, and (ii) the tail node of each path is the only node on the path which is known to correspond to a node in H. For each such path, we delete the k-2 interior nodes from G and make the head node correspond to the $k\text{-}1^{st}$ interior node on the path in H. Then $H \leq_N G$ if and only if there is a graph, G', derived from G by the above method, such that $H' \leq_N G'$. Since k is a constant with respect to the size of G, the path enumeration can be done in time polynomial in the size of G. The reduction for each path takes only constant time. Figure 3 illustrates the reduction for a particular path in G.

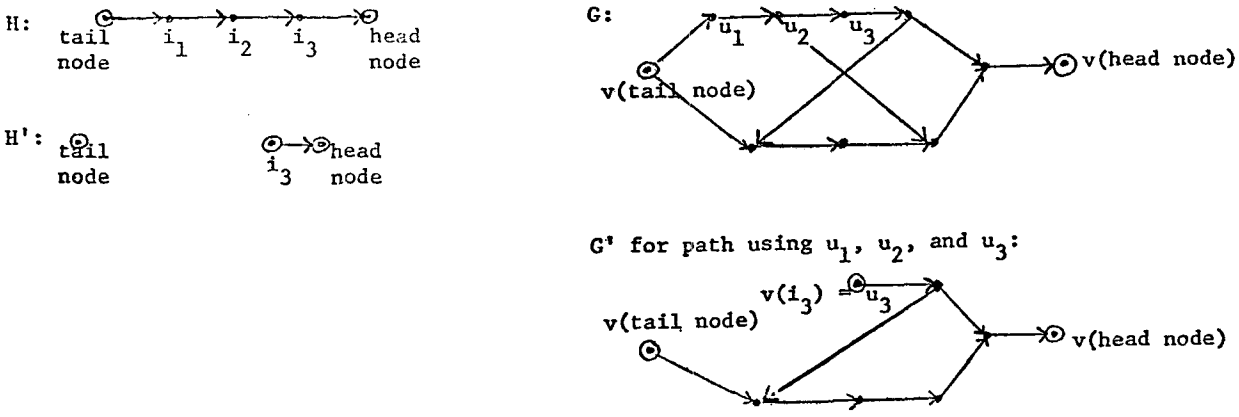
Reduction 2: The second reduction is applicable when H contains a node, called a parent node, which is adjacent to k>0 nodes, called leaf nodes. The leaf nodes must have no other nodes adjacent to them. The correspondence of the parent node to a node in the input graph, G, must be known, and the correspondence of the leaf nodes to nodes in G must be unknown. Then, in G, we may assume that the nodes corresponding to leaf nodes are adjacent to the node corresponding to the parent node by edges of the appropriate direction. For each possible image set in G of the leaf nodes in H, we delete the leaf nodes in H and the image set in G to produce two new graphs, H' and G'. If for any such H' and G', $H' \leq_N G'$, then $H \leq_N G$. Figure 4 illustrates the reduction.

Figure 2: Construction for Lemma 2



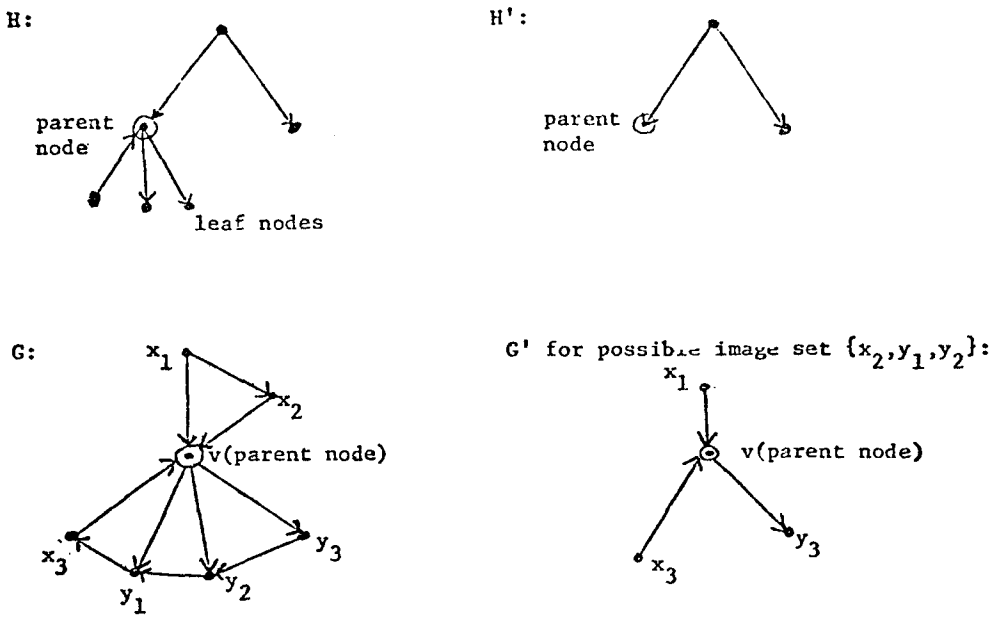
d_u is the degree of node u; d_t is the degree of node v.
G' and H' are the graphs produced by the reduction.

Figure 3: Illustration of Reduction 1



⊙ indicates a node whose image or inverse image under node mapping v is known.

Figure 4: Illustration of Reduction 2



⊙ indicates a node whose image or inverse image under the node mapping, v, is known.

The two special purpose reductions just presented are only useful if we know how to solve the SHP for the resulting pattern graph, H'. Dinic's algorithm for single commodity network flow problems [Di] provides polynomial time algorithms for some SHP's, for example: any fixed SHP when the pattern graph is a tree of depth one (directed or undirected). However, for fixed SHP's, such simple pattern graphs as a cycle of length three or two disjoint edges do not lend themselves to single commodity network flow formulation. In the next section, we outline a linear time algorithm to solve the fixed SHP when the pattern graph is an undirected cycle of length three.

IV. A Linear Time Algorithm for the Triangle Problem

We now present a linear time algorithm for the following problem: given an undirected graph, G, and three nodes of G, determine whether the three nodes lie on a common simple cycle, and construct that cycle if it exists. We attack the problem by breaking G into components and looking for paths which must exist in these components if the cycle is to exist in G. We build up sets of node disjoint paths known to be in G until we can piece together the desired cycle from these paths or declare that the cycle does not exist. The algorithm is rather lengthy and involves much case analysis. Some details have been omitted due to space and readibility considerations. The algorithm was originally presented in [Lap], where all details can be found.

Let the three specified nodes of the graph G be A, B, and C. We assume that G is biconnected since A, B, and C must be in the same biconnected component if they lie on the same cycle. We also assume that none of the edges (A,B), (A,C), or (C,B) are in G. If one of them, say (A,C), is in G, the problem is reduced to finding a path from A to C which contains B. This can be done in linear time as a unit vertex and edge capacity single commodity network flow problem with source B and sinks A and C.

If G contains three node disjoint paths, each with A as one endpoint and one of B or C as the other endpoint (with renaming of the nodes if necessary), then a cycle can be constructed by piecing together parts of these paths with parts of other paths which must exist by the biconnectivity of G. Three such node disjoint paths can be found by merging nodes B and C in G into one node, denoted BC, and applying Dinic's network flow algorithm to the resulting graph with source BC and sink A. When B and C are merged, each edge to/from B or C becomes an edge to/from BC. (Duplicate edges are removed.)

If G contains three node disjoint paths, each with A as one endpoint and one of B or C as the other endpoint, we can always find three node disjoint paths from A such that two have B as the other endpoint and one has C as the other endpoint (or vice versa). To see this, suppose we have three node disjoint paths from A to B. There must be a path, R, from C to A which does not contain B, by the biconnectivity of G. Some initial portion of this path, R[C,z], is node disjoint from the three paths between A and B except at z. (The notation "p[u,t]" denotes the portion of a path p from node u to node t.) Using this initial portion and the portion of the path it intersects from z to A gives a path from A to C which is node disjoint from the two remaining paths between A and B.

Now assume that two of the paths found have B as one endpoint and one has C as an endpoint (or vice versa). Call these paths P_1, P_2, and P_3 respectively. Since G is biconnected, we can find two node disjoint paths from B to C. Call these paths Q_1 and Q_2. Define x_1 to be the closest node to C on $Q_1[C,B]$ which is also on $P_1[A,B]$ or $P_2[A,B]$. Define x_2 on Q_2 similarly. Both x_1 and x_2 may equal B, but at most one equals A, since Q_1

and Q_2 are node disjoint. Let y be the closest node to A on $P_3[A,C]$ which is also on $Q_1[C,x_1]$ or $Q_2[C,x_2]$. Since x_1 (or x_2) is on P_3 only if $x_1 = A$ (or $x_2 = A$), and y is on P_3, y can equal one of x_1 or x_2 only if y is equal to A. Without loss of generality, assume y is on Q_1. Then y cannot equal x_2. If x_2 is on P_2, the path composed of subpaths $P_1[A,B]$, $P_2[B,x_2]$, $Q_2[x_2,C]$, $Q_1[C,y]$, and $P_3[y,A]$, in that order, is a simple cycle containing A, B, and C. If x_2 is on P_1, subpaths $P_2[A,B]$ and $P_1[B,x_2]$ are used instead of $P_1[A,B]$ and $P_2[B,x_2]$.

If G does not contain three node disjoint paths from A to B and C, then we do the following:

Step 1: We find a node cutset of size two which separates A from B and C and such that this cutset cannot be separated from B and C by removing any other two nodes of G. If this cutset also separates B from C, there is no cycle containing A, B, and C, and we are done.

The required cutset can be found using information provided by Dinic's network flow algorithm. Given a flow from a source node, s, to a terminal node, t, Dinic's algorithm proceeds by finding an augmenting path along which flow can be increased while maintaining the edge and node capacity restrictions on the flow. For networks with unit edge and node capacities, as in our application, the augmenting path can use edges not used by the present flow, and edges used by the present flow but in the opposite direction to the flow. The augmenting path will contain an interior node on a path of present flow only if at least one edge incident on that node in the augmenting path is used by a path of present flow in the opposite direction. Upon termination of the algorithm, there are no augmenting paths to t. However, we know to which nodes there remain augmenting paths from s.

For our application, A is the terminal node, and B and C are merged into one source node, BC. If we are executing Step 1, we can find exactly two node disjoint paths from A to B or C, corresponding to a flow of 2 from BC to A. Call the paths of flow from BC to A, P1 and P2. Suppose the set of nodes to which there are augmenting paths from source BC upon termination of Dinic's algorithm includes a node on P1. Let A1 be the closest such node to A on P1. Otherwise, let A1 be the node adjacent to BC on P1. Define A2 on P2 similarly. The set {A1,A2} is a cutset separating A from B and C in G. This can be seen by noting that any path from A to B or C which does not contain A1 or A2 would define an augmenting path from BC to some node (possibly A) closer to A on P1 than A1 or closer to A on P2 than A2.

Suppose there are two nodes different from A1, A2, B and C which separate A1 and A2 from B and C in G. Then one of these nodes must appear on P1[BC,A1] and the other must appear on P2[BC,A2]. However, either A1 is adjacent to BC, or, given flows along P1[BC,A1] and P2[BC,A2], there is an augmenting path to A1. This augmenting path increases the flow into A1 while leaving the flow to A2 unchanged. The new flow corresponds to three node disjoint paths in G--two from B or C to A1, and one from B or C to A2. Thus, two nodes cannot separate B and C from A1 and A2.

Step 2: We now consider only that component K of G which contains B and C when nodes A1 and A2 are removed. (If B and C are not in one component, the cycle does not exist.) We have a path in G from A1 to A2 containing A which lies outside K. To complete the cycle, we would like to find a path from A1 to A2 containing B and C in either order whose interior nodes are in the component K. If both of edges (A1,B) and (A2,C) or both of edges (A1,C) and (A2,B) are in G, we are done. Any path from B to C in component K will complete the desired cycle.

If neither of the above pairs of edges is in G, we test if B and C are biconnected in K. If B and C are not biconnected in K, we can determine if a path from A1 to A2 containing B and C exists. Let x be a node separating B and C in K. (Remember that edge (B,C) doesn't exist.) Removing x separates K into components. Let K_B denote the component containing B and K_C denote the component containing C. Any path from A1 to A2 containing B and C must consist of a path from A1 (or A2) to x containing B whose interior nodes are all in K_B, and a path from A2 (respectively A1) to x containing C whose interior nodes are all in K_C. The existence of these paths is easily determined by solving appropriate network flow problems.

If B and C are biconnected in K, we continue.

Step 3: To component K, we add A1, A2, and all edges of G which go from A1 or A2 to nodes of K. Call the resulting subgraph of G, K'. Due to our choice of A1 and A2 as the "nearest" cutset of size two to B and C, there are two cases for each of A1 and A2. The cases for A1 are:

(a) There are three node disjoint paths in the new graph, K', such that two of the paths have A1 as one endpoint and either B or C as the other endpoint. The third path connects A2 with one of B and C.

(b) The only possible paths from A1 to B or C which do not contain A2 are edges (A1,B) and (A1,C). At least one of these edges is an edge of K'.

Cases (a) and (b) are not mutually exclusive, but if (a) does not hold, (b) must hold. The cases for A2 are analogous with the roles of A1 and A2 interchanged. If it is not the case that (a) holds for both A1 and A2, we can reduce the problem of finding a path from A1 to A2 containing B and C to at most two instances of a problem of the form: "Is there a path in K' from A2 to B containing C but not A1?" (This is the question used if (A1,B) exists; B and C are interchanged for the second question if (A1,C) exists. Obviously, if neither path exists, the cycle doesn't exist either.) If the new graph does contain both sets of node disjoint paths, i.e. (a) holds for both A1 and A2, we continue.

Step 4: The new graph, K', may not be biconnected, regardless of whether or not the component K from which it was constructed is biconnected.

Claim: If K' is not biconnected, B and C are the only possible articulation points.

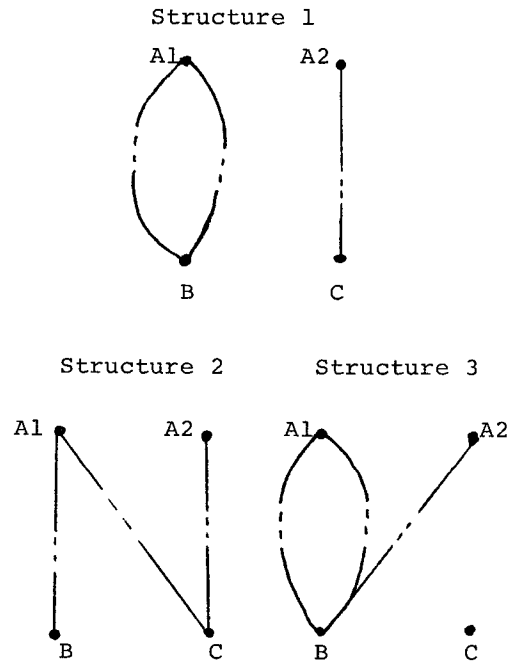
Proof: Observe that (i) A1 and A2 cannot be articulation points of K', since K' was formed by adding A1 and A2 to a component of G resulting from the removal of A1 and A2. (ii) Any articulation point of K' must separate A1 from A2 in K'. Otherwise, the articulation point is also an articulation point of G, but G is biconnected. (iii) There are two node disjoint paths in K' from B to C which contain neither A1 nor A2. (iv) There are two node disjoint paths in K' from A1 to B or C. (v) There are two node disjoint paths in K' from A2 to B or C.

Given observations (iii)-(v) above, we can construct a path in K' from A1 to A2 which does not contain node x, where x is any node in K' other than A1, A2, B and C. The details of the construction are left to the reader and are available in [Lap].

If K' is not biconnected, it is simple to split K' into its biconnected components and determine if the required subpaths of a path from A1 to A2 containing B and C exist in the appropriate components. If K' is biconnected, we continue.

Step 5: All sets of three node disjoint paths satisfying Case (a) of Step 3 must have Structure 1, Structure 2, or Structure 3, as shown in Figure 5. Since K' is biconnected, Structure 3 can be reduced to Structure 1 or Structure 2 by a method similar to that used on the original graph, G, to reduce three node disjoint paths from A to B to three node disjoint paths with two from A to B and one from A to C. If Structure 1 is found for either A1 or A2, we can piece together the desired path from A1 to A2 containing B and C using the three disjoint paths of Structure 1 and other paths known to exist in K'. The technique used is essentially the same as that used when there are three node disjoint paths from A to B or C in the original graph, G. If only Structure 2 is found for each of A1 and A2, we continue. Note that we now have two sets of node disjoint paths --one of Structure 2 for A1 and one of Structure 2 for A2.

Figure 5: Possible structures for Step 5



Structures shown are for A1. B and C are interchangeable. Structures for A2 are obtained by interchanging A1 and A2.

Step 6: We determine if there are three node disjoint paths in K', each with A1 or B as one endpoint and A2 or C as the other endpoint, or each with A2 or B as one endpoint and A1 or C as the other. If neither set of node disjoint paths exists, then the desired path from A1 to A2 containing B and C cannot exist since the subpaths of this path would be one such set of node disjoint paths. Again, we can use Dinic's network flow algorithm to test if three such paths exist.

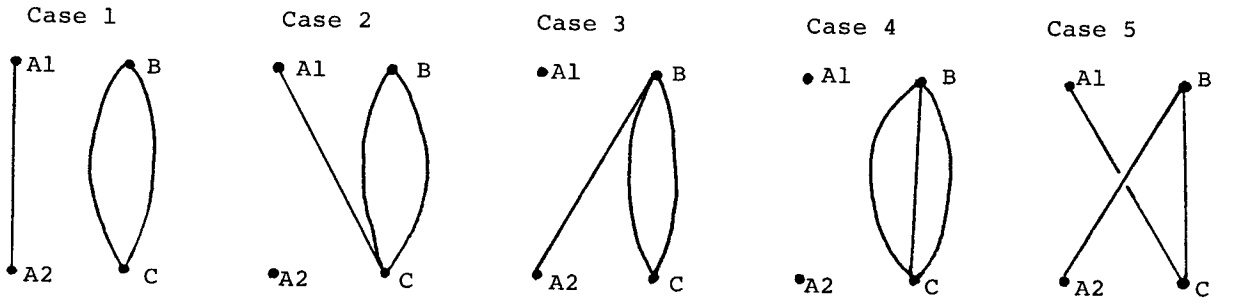
If we find any such set of three node disjoint paths, the desired path from A1 to A2 containing B and C is guaranteed to exist. We use the sets of node disjoint paths found so far to construct the desired path. In the following discussion, we assume that there are three node disjoint paths each with A1 or B as one endpoint and A2 or C as the other endpoint. A parallel argument deals with the case when the three node disjoint paths instead go from A1 or C to A2 or B.

Consider the two disjoint paths from B to C containing neither A1 nor A2 which are guaranteed to exist by Step 2. Since there are three node disjoint paths from A1 or B to A2 or C, there is at least one augmenting path in a network constructed from K' with flow corresponding to the two node disjoint paths between B and C. This augmenting path results in three node disjoint paths, each from A1 or B to A2 or C. In addition, we now have that at least two of the paths must have B as an endpoint and at least two must have C as an endpoint, since augmenting paths can only increase the flow into or out of endpoints of flow. The possible configurations of these paths are shown in Figure 6. Cases 4 and 5 require no further discussion. Cases 2 and 3 can be reduce to Case 1 or Case 5 by recalling that K' is biconnected and using the same technique used when we had three node disjoint paths from A to B. We now deal with Case 1.

Let us review the sets of node disjoint paths at our disposal. We have: (i) By Case 1 above, three node disjoint paths -- one from A1 to A2 and two from B to C, (ii) By Step 5, three node disjoint paths --one from A1 to B, one from A1 to C, and one from A2 to B or C, (iii) Again by Step 5, three node disjoint paths -- one from A2 to B, one from A2 to C, and one from A1 to B or C.

Denote the paths of (ii) above collectively by Q. For each path of (ii) above, there is some initial portion, $Q[A1,q_1]$, $Q[A1,q_2]$, or $Q[A2,q_3]$, respectively, which is node disjoint from the disjoint paths from B to C except at endpoint q_1, q_2, or q_3.

Figure 6: Configurations of node disjoint paths for Step 6



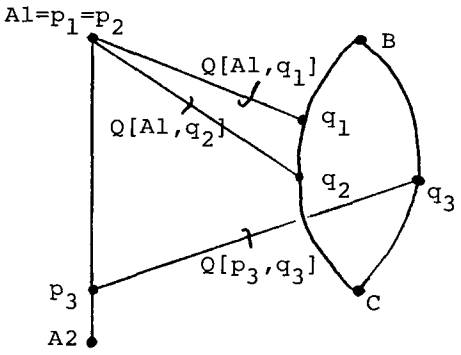
Some final portion, $Q[p_k,q_k]$, of each of these subpaths is also node disjoint from the path from A1 to A2 except at p_k, k=1, 2, or 3. All configurations of the p_k's and q_k's on the paths of (i) except the configuration shown in Figure 7a yield a path from A1 to A2 containing B and C. Figure 7b illustrates one successful configuration. If we have the configuration shown in Figure 7a, a similar analysis is done for the paths of (iii). The subpaths of the paths of (iii) analogous to $Q[p_k,q_k]$, k= 1, 2, and 3, are denoted $R[s_k,r_k]$. We take into account the ways in which the $R[s_k,r_k]$ can intersect $Q[A1,q_1]$ and $Q[A1,q_2]$ already found. If we have not found a path from A1 to A2 containing B and C after processing the paths of (iii), we must have the configuration of paths shown in Figure 8.

Our construction of K' insures that edge (A1,A2) is not in K'. Therefore the path from A1 to A2 in (i) above is of length at least two. Let x be any interior node on the path from A1 to A2. Node x is also in component K from which K' was constructed. Therefore, there is a path from x to B which contains neither A1 nor A2. Call this path, P. Let y be the closest node to x on P which is also on the node disjoint paths from B to C, $Q[A1,q_1]$, $Q[A1,q_2]$, $R[A2,r_1]$, or $R[A2,r_2]$. Let z be the closest node to y on P[x,y] which is also on the path from A1 to A2. Note that z does not equal A1 or A2. All possible positions for z and y yield a path from A1 to A2 which contains B and C, as can be verified by examining Figure 9.

The algorithm presented above relies heavily on J. Hopcroft's linear time algorithm to find biconnected components [Ah][Ho 1973a][Ta 1972] and Dinic's algorithm for one-commodity network flow [Di]. We never need to find more that a flow of three using Dinic's algorithm (corresponding to a set of three node disjoint paths). Therefore, our use of the algorithm requires only linear time. (R.E. Tarjan and S. Even have provided a careful analysis of this algorithm from which we make this conclusion [Ev 1975] [Ta 1974].) Each step of the algorithm presented above can be done in linear time and is executed at

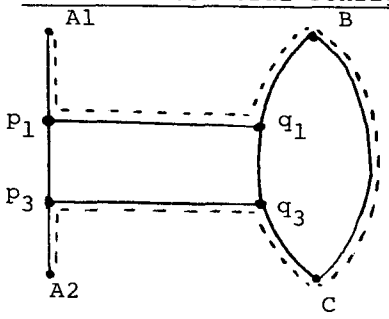
Figure 7: Merging sets of paths in Step 6

7a: failing configuration



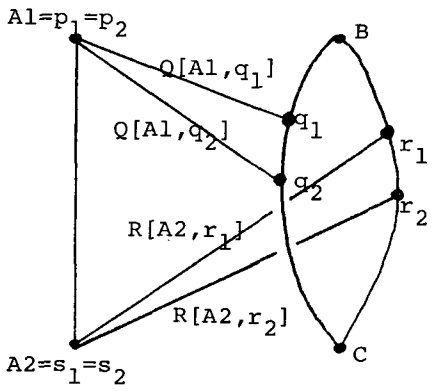
$q_1 \neq$ B or C
$q_2 \neq$ B or C
$q_3 \neq$ B or C

7b: one successful configuration



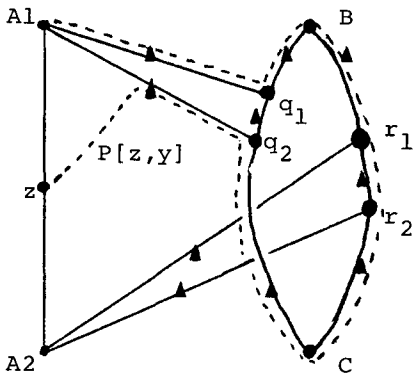
---- indicates desired path

Figure 8: Last configuration of paths before completion of Step 6



r_1, r_2, q_1, and q_2 are not equal to B or C.

Figure 9: Positions for node y of Step 6--all yield desired path



▲ indicates a possible position for for y. In addition, y may equal q_1, q_2, r_1, r_2, B, or C. One solution is indicated by -----.

most once. Therefore, the algorithm is of linear time. A more detailed discussion of the timing of the algorithm, including necessary bookkeeping, can be found in [Lap].

V. Conclusion and Further Problems

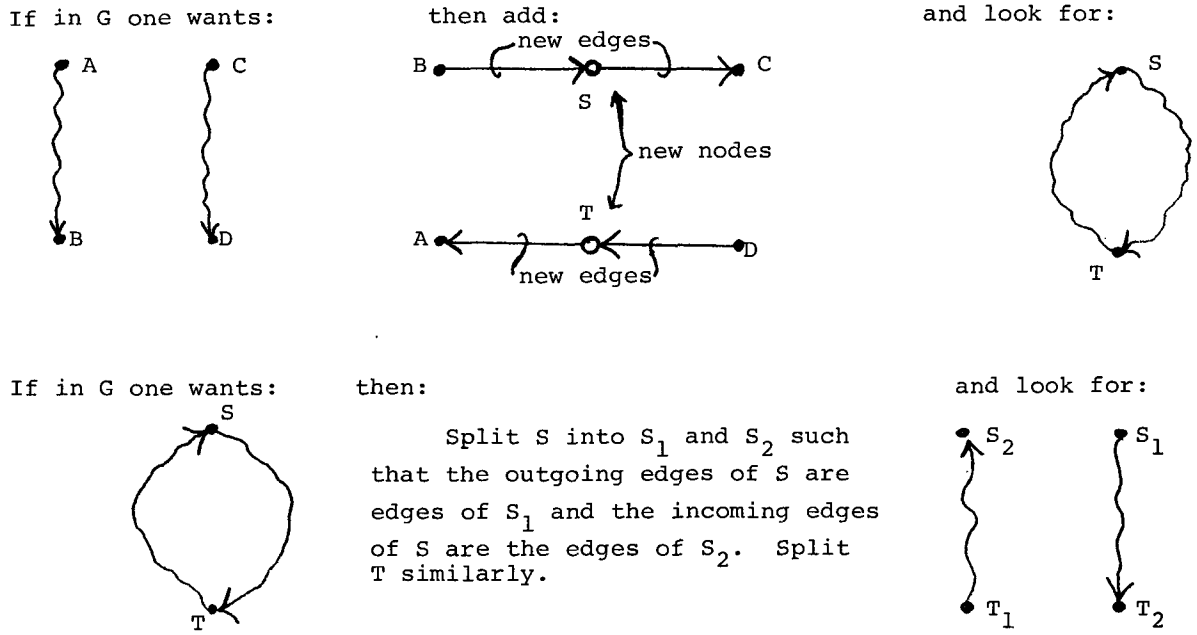
We have presented above a general description of the SHP and some methods of reducing one SHP to another in polynomial time. We have also presented a linear time algorithm to determine if three given nodes of an undirected graph lie on a common simple cycle. This problem and the two disjoint paths problem--given two pairs of nodes of an undirected graph, are there two node disjoint paths in the graph with each pair of nodes serving as the endpoints of one of the paths?--are basic problems for all fixed SHP's for undirected graphs. Any undirected graph which has at least two edges and is not a tree of depth one will contain a cycle of length three or two disjoint edges. Thus, the fixed SHP for any undirected pattern graph with more than two edges will contain one of the above problems as a subproblem unless it is a tree of depth one.

Contributions to the solution of the two disjoint paths problem are found in several places. Larman and Mani [Lar] and Watkins [Wa] address the question: "What properties of an undirected graph guarantee the existence of two node disjoint paths between any two pairs of nodes in the graph?" Yossi Shiloach now claims to have solved the two disjoint paths problem for any undirected graph [Sh]. His solution expands results of Watkins for a graph G such that the complete graph on five nodes is homeomorphic to a subgraph of G and the node connectivity of G is at least four. It extends earlier results of Perl and Shiloach for planar graphs [Perl].

For directed graphs, polynomial time algorithms for the most basic fixed SHP's are still open problems. The two disjoint paths problem is equivalent to finding a cycle containing two given node of a directed graph. (See Figure 10.) S. Even and M. Garey have obtained some results concerning edge disjoint cycles [Ev 1977]. However, the problem of determining whether two given nodes of a directed graph lie on a common simple cycle remains open. The only fixed SHP for directed graphs which we do know how to solve in polynomial time is that for a pattern graph which is a tree of depth one, since this problem can be modeled as a single commodity network flow problem.

We see that SHPs encompass a large number of natural problems in the area of algorithms on graphs. The complexity of most of these problems remains open; our results here represent only the initial steps towards resolving these questions. For example, the reductions outlined in Section III are of somewhat limited use and stronger reductions are desirable. We would like to be able to know when we can add an edge or a node to a pattern graph or further specify a node mapping, and then be able to modify an existing algorithm to solve the new problem. Most of all, we would like to know whether the SHP for every pattern graph H has a polynomial-time algorithm for its solution.

Figure 10: Equivalence of two fixed SHP's for directed graphs



References

[Ah] Aho, A., Hopcroft, J., Ullman, J., The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1974.

[Be] Berge, Claude, The Theory of Graphs and its Applications, John Wiley & Sons, Inc., New York, 1966.

[Di] Dinic, E.A., "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation," Soviet Mathematics Doklady, Vol. 11, No. 5, 1970, pp. 1277-1280.

[Ev 1977] Even, S. private communication.

[Ev 1976] Even, S., Itai, A., Shamir, A., "On the Complexity of Timetable and Multi-commodity Flow Problems," SIAM Journal on Computing, Vol. 5, No. 4, Dec. 1976, pp. 691-703.

[Ev 1975] Even, S., Tarjan, R.E., "Network Flow and Testing Graph Connectivity," SIAM Journal on Computing, Vol. 4, No. 4, Dec. 1975, pp. 507-518.

[Ge] Geller, Dennis, "Forbidden Subgraphs," Proof Techniques in Graph Theory, Frank Harary, ed., Acedemic Press, New York, 1969, pp. 37-47.

[Ha 1971] Harary, Frank, Graph Theory, Addison-Wesley Publishing Co., Reading, Mass., 1971.

[Ha 1973] Harary, Frank, "On the History of the Theory of Graphs," New Directions in the Theory of Graphs, Frank Harary, ed., Acedemic Press, New York, 1973, pp. 1-17.

[Ho 1973a] Hopcroft, J.E., Tarjan, R.E., "Algorithm 447: Efficient Algorithms for Graph Manipulation," Communications of the ACM, Vol. 8, No. 6, June 1973, pp. 372-378.

[Hu] Hu, T.C., Integer Programming and Network Flows, Addison-Wesley Publishing Co., Reading, Mass., 1969.

[Hunt] Hunt, H.B., Szymanski, T.G., "Dichotomization, Reachability, and the Forbidden Subgraph Problem," Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1976, pp. 126-134.

[Lap] LaPaugh, A., "The Subgraph Homeomorphism Problem," Massachusetts Institute of Technology, Laboratory for Computer Science TM 99, Feb. 1978.

[Lar] Larman, D., Mani, P., "On the Existence of Certain Configurations within Graphs and the 1-Skeletons of Polytopes," Proceedings of the London Math. Soc., Vol. 20, No. 3, 1970, pp. 144-160.

[Perl] Perl, Y., Shiloach, Y., "Finding Two Disjoint Paths Between Two Pairs of Vertices in a Graph," Journal of the ACM, Vol. 25, No. 1, Jan. 1978, pp. 1-9.

[Rog] Rogers, H., Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill Book Co., New York, 1967.

[Sh] Shiloach, Y., private communication.

[Ta 1972] Tarjan, R. E., "Depth First Search and Linear Graph Algorithms," SIAM Journal on Computing, Vol 1, No. 2, June 1972, pp. 146-160.

[Ta 1974] Tarjan, R.E., "Testing Graph Connectivity," Proceedings of the Sixth Annunal ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1974, pp. 185-193.

[Wa] Watkins, Mark, "On the Existence of Certain Disjoint Arcs in Graphs," Duke Mathematical Journal, Vol. 35, 1968, pp. 231-246.