MIT/LCS/TM-99

THE SUBGRAPH HOMEOMORPHISM PROBLEM

Andrea Suzanne LaPaugh

February 1978

THE SUBGRAPH HOMEOMORPHISM PROBLEM*

by

Andrea Suzanne LaPaugh

Submitted to the Department of Electrical Engineering
and Computer Science
on May 31, 1977 in partial fulfillment of the requirements
for the Degree of Master of Science

ABSTRACT

The problem investigated in this thesis is that of finding homeomorphic images of a given graph, called the pattern graph, in a larger graph. A homeomorphism is a pair of mappings, (ν,α), such that ν maps the nodes of the pattern graph to nodes of the larger graph, and α maps the edges of the pattern graph to (edge or node) disjoint paths in the larger graph. A homeomorphism represents a similarity of structure between the graphs involved. Therefore, it is an important concept for both graph theory and applications such as programming schema.

We give a formal definition of the subgraph homeomorphism problem. In our investigation, we focus on algorithms which depend on the pattern graph and allow the node mapping, ν, to be partially or totally specified. Reductions between node disjoint and edge disjoint formulations of the problem are discussed. Also, reductions facilitating the solution of given subgraph homeomorphism problems are formulated. A linear time algorithm for finding a cycle in a graph containing three given nodes of the graph is presented. Finally, the two disjoint paths problem, an open problem, is discussed in detail.

Thesis Supervisor: Ronald L. Rivest, Associate Professor of Electrical
Engineering and Computer Science

* This research was supported in part by National Science Foundation grant MCS76-14294.

Author's Note

This publication contains minor revisions to the thesis submitted in May,1977. Since that time, an $\mathcal{O}(|V||E|)$ time algorithm to solve the two disjoint paths problem has been found by Y. Shiloach[1]. This problem is discussed in Chapter V as an open problem. Shiloach's solution includes a proof of Watkin's conjecture (cf. Chap. V, p. 106). The new algorithm extends the earlier work of Perl and Shiloach [Perl] for planar graphs.

1. Shiloach, Y., private communication.

Key words: homeomorphisms of graphs, path-finding algorithms, forbidden subgraph properties, cycles in graphs .

Acknowledgements

This research has been done in close association with my thesis supervisor, Ronald L. Rivest. He has contributed many ideas to this thesis, especially in the original formulation of the problem and in the development of reductions between various problems. I would like to thank him not only for his contributions, but also for his encouragement throughout the writing of this thesis.

I would also like to thank the residents of Suite 308 and my parents for their moral support, and J. Hartmanis and A. Nerode for introducing me to the theory of computation.

Finally, to Michael: we made it!

TABLE OF CONTENTS

I Introduction

In this thesis we examine various forms of the subgraph homeomorphism problem. A subgraph homeomorphism problem is a problem which may be described as the search for a homeomorphism from a pattern graph H to a subgraph of another graph G. A homeomorphism between two graphs characterizes a similarity of structure within the two graphs. Similarity of structure is an important concept for graph theory and for applications such as modeling computer programs.

Our goal in the body of this thesis is to present a precise definition of each of the forms of the subgraph homeomorphism problem, and to present solutions to specific subgraph homeomorphism problems. We also relate the various forms to each other in the hope that solving one type of subgraph homeomorphism problem may lead to the solution of other types. In Chapter II, we define the subgraph homeomorphism problem and give some motivation for our interest in it. We also prove various relationships between subgraph homeomorphism problems defined using node disjoint paths and those defined using edge disjoint paths. In Chapter III, we discuss several methods of solving subgraph homeomorphism problems based on a network flow algorithm and reductions which we will describe there. In Chapter IV, we present a new algorithm which solves a subgraph homeomorphism problem when the pattern graph is a cycle of length three. This problem is equivalent to finding a cycle in an undirected graph containing three given nodes. In Chapter V, we concentrate on what we believe to be the most alluring open subgraph homeomorphism problem -- the two disjoint paths problem. Chapter VI summarizes our results and presents other directions of possible future research.

II Preliminaries

II.1 Definitions

A <u>directed graph</u> $G = \langle V,E\rangle$ consists of a finite set V of <u>nodes</u> (or vertices) and a set $E \subseteq V\times V$ of <u>edges</u>. If $e=(u,v)$ is an edge in E, we say edge e goes from node u to node v, and nodes u and v are <u>adjacent</u> nodes. If a node v has k edges entering it and j edges leaving it, we say that v has <u>indegree</u> k and <u>outdegree</u> j. An <u>undirected graph</u> $G = \langle V,E\rangle$ is defined similarly except that E contains unordered pairs of nodes. For an undirected graph, we will retain the notation $e=(u,v)$ with the understanding that $(u,v)=(v,u)$, and edge e goes from u to v or from v to u. In this case, the indegree and outdegree of a node are the same and will simply be called the <u>degree</u> of the node. A <u>subgraph</u> of a graph G is a graph $S = \langle U,A\rangle$ such that $U\subseteq V$ and A is a subset of E containing only edges which go between nodes in U.

Given a graph $G = \langle V,E\rangle$ (directed or undirected), we define a <u>path</u> p in G to be a sequence $\langle(v_1,v_2)(v_2,v_3)\ldots(v_{n-1},v_n)\rangle$ of edges in G, where $n\geq 2$. The path may also be denoted $\langle v_1,v_2,\ldots v_n\rangle$. We say that p goes from node v_1 to v_n and is of length n-1. Each edge (v_i,v_{i+1}), $1\leq i\leq n-1$, is an edge of p, and each node, v_i, $1\leq i\leq n$, is on path p. The nodes v_1 and v_n are <u>endpoints</u> of p, and nodes v_i, $1< i\leq n-1$, are <u>interior points</u> of p. A <u>subpath</u> of p is any path $\langle(v_i,v_{i+1})(v_{i+1},v_{i+2})\ldots(v_{j-1},v_j)\rangle$ where $1\leq i< j\leq n$. We call a path <u>simple</u> if $v_i\neq v_j$ for $i\neq j$, $1\leq i\leq n$, $1\leq j\leq n$, except that v_1 may equal v_n, in which case the path is a <u>simple cycle</u>. Two paths, p_1 and p_2, are <u>node disjoint</u> if $v_i\neq u_j$, $1\leq i\leq m$, $1\leq j\leq n$, where

$p_1=\langle(v_1,v_2)(v_2,v_3)\ldots(v_{m-1},v_m)\rangle$ and $p_2=\langle(u_1,u_2)(u_2,u_3)\ldots(u_{n-1},u_n)\rangle$. Two paths, p_1 and p_2 as above, are edge disjoint if $(v_i,v_{i+1}) \neq (u_j,u_{j+1})$, $1\leq i\leq m-1$, $1\leq j\leq n-1$.

A (rooted) tree is a graph G containing no cycles (when direction of edges in directed graphs is ignored). One node is distinguished as the root, and there is a path from the root to each node in G. Nodes in G of degree one other than the root (or of outdegree 0 for directed graphs) are termed leaves.

In the following discussion, let $G = \langle V_G,E_G\rangle$ and $H = \langle V_H, E_H\rangle$ both be directed graphs. Let P(G) be the set of all simple paths in G. H is said to be (node disjoint) homeomorphic to a subgraph of G, denoted $H\leq_N G$, if there exists a one to one mapping $\nu:V_H\to V_G$ and a one to one mapping $\alpha:E_H\to P(G)$ such that:

i) $\alpha((u_1,u_2))=p$ implies p goes from $\nu(u_1)$ to $\nu(u_2)$;
ii) for any two distinct paths p_1 and p_2 in $\alpha(E_H)$, p_1 and p_2 share at most one vertex, which is an endpoint for both p_1 and p_2;
iii) for all $u\epsilon V_H$, $\nu(u)$ is on p, a path in $\alpha(E_H)$, if and only if $\nu(u)$ is an endpoint of p.

Then (ν,α) is a homeomorphism from H to the subgraph of G:

$$\langle \nu(V_H)\cup\{v\epsilon V_G \mid (\exists p)\ p\epsilon\alpha(E_H) \text{ and } v \text{ is an interior node of } p\},\ \{e\epsilon E_G \mid (\exists p)\ p\epsilon\alpha(E_H) \text{ and } e \text{ is an edge of } p\}\rangle .$$

Condition iii is implied by conditions i and ii if H contains no isolated nodes (i.e. nodes adjacent to no other node). If we replace conditions ii and iii by the condition that $\alpha(E_H)$ be a set of pairwise edge disjoint simple paths of G, then H is edge disjoint homeomorphic to a subgraph of G, denoted $H\leq_E G$.

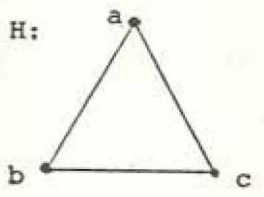
In the case of undirected graphs, an alternate definition of node disjoint homeomorphism has been used historically. In this definition, $H\leq_N G$ if nodes can be inserted along the edges of H to yield a new graph H' which is isomorphic to a subgraph of G [Ha 1973, p.8]. This definition is readily seen to be equivalent to the above definition. (See Figure II.1-1.) We have chosen to use the above definition since defining a homeomorphism in terms of paths of G allows one to conceptualize the problem of finding homeomorphisms in terms of finding paths in G and to readily use the body of pathfinding algorithms already in the literature.

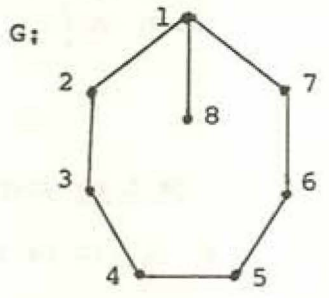
Our research has been primarily concerned with node disjoint homeomorphisms. Section II.3 discusses relationships between node disjoint and edge disjoint homeomorphisms. In the remainder of this section and the next, we discuss only node disjoint homeomorphisms, hereafter simply called homeomorphisms.

For both the directed and undirected cases, the general subgraph homeomorphism problem -- given H and G, is H homeomorphic to a subgraph of G -- is NP-complete. This can easily be seen by considering the Hamiltonian Circuit problem [Ah, pp. 378-394]. Given the question, "Does G contain a Hamiltonian circuit?" we construct H such that $|V_H|=|V_G|$ and the edges of H connect the vertices in a cycle. We then ask, "Is H homeomorphic to a subgraph of G?" Then H is homeomorphic to a subgraph of G if and only if G contains a Hamiltonian circuit.

Given that the general subgraph homeomorphism problem is NP-complete, our research has focused on the existence of polynomial time algorithms when H is a constant. (Thus, these algorithms may take a number of steps polynomial in the size of G, i.e. $|V_G|+|E_G|$, where the degree may be a function of the size of H.) We further allow as input a partial or total specification of ν. In this case, the subset of V_H which serves as the domain of the

Figure II.1-1 Illustration of the two definitions of homeomorphism.





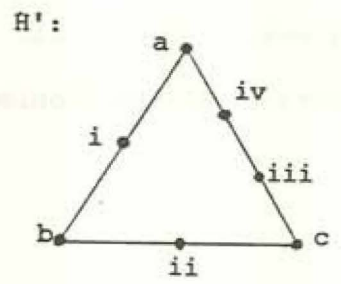
Under definition given, $H \leq_N G$ by:

ν: a → 1 $\qquad$ α: (a,b) → <(1,2),(2,3)>

b → 3 $\qquad$ (b,c) → <(3,4),(4,5)>

c → 5 $\qquad$ (c,a) → <(5,6),(6,7),(7,1)>

Under alternate definition:



Mapping:	a → 1
	i → 2
	b → 3
	ii → 4
	c → 5
	iii → 6
	iv → 7

partial specification of ν is also a constant. The problem under consideration can then be stated as follows:

Given: $H=<V_H,E_H>$ a fixed undirected (directed) graph.
Let $N_H \subseteq V_H$ also be fixed for the problem.

Input: Undirected (directed) graph $G=<V_G,E_G>$ and $\rho : N_H \rightarrow V_G$ a one to one function.

Problem: Find $\nu : V_H \rightarrow V_G$ one to one and $\alpha : E_H \rightarrow P(G)$ one to one such that:

i) (ν,α) is a homeomorphism from H to a subgraph of G.

ii) $(\forall v \in N_H)\ \nu(v)=\rho(v)$.

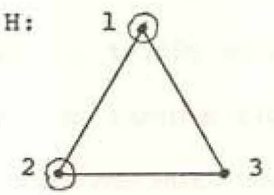
H is called the <u>pattern</u> graph, and G the <u>input</u> graph.

If $N_H = \emptyset$ (i.e. ρ is vacuous), ν is unspecified. We call this an instance of the <u>floating</u> subgraph homeomorphism problem. This terminology arises from the fact that the vertices of H can be mapped anywhere in G. If $N_H = V_H$, ν is totally specified. We call this an instance of the <u>fixed</u> subgraph homeomorphism problem. If $\emptyset \neq N_H \subsetneq V_H$, ν is partially specified.

Consider the example of Figure II.1-2. This example illustrates an instance of the subgraph homeomorphism problem when ν is partially specified. The presented solutions make it clear that there need not be a unique solution to any instance of the problem.

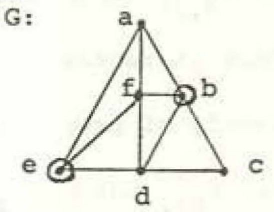
Figure II.1-2: Example of an instance of the subgraph homeomorphism problem with ν partially specified.

Fixed: H:



$N_H = \{1.2\}$

Input: G:



ρ: $1 \rightarrow b$
$2 \rightarrow e$

Solution 1: ν: $1 \rightarrow b$, $2 \rightarrow e$, $3 \rightarrow c$

α: $(1,3) \rightarrow \langle(b,c)\rangle$
$(3,2) \rightarrow \langle(c,d),(d,e)\rangle$
$(2,1) \rightarrow \langle(e,a),(a,b)\rangle$

Solution 2: ν: $1 \rightarrow b$, $2 \rightarrow e$, $3 \rightarrow d$

α: $(1,3) \rightarrow \langle(b,c),(c,d)\rangle$
$(3,2) \rightarrow \langle(d,e)\rangle$
$(2,1) \rightarrow \langle(e,a),(a,b)\rangle$

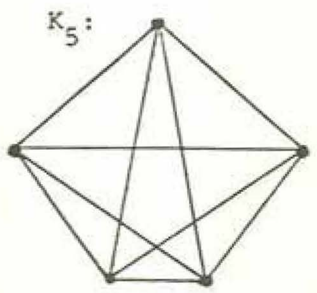
$\odot$ indicates a node whose image in V_G or inverse image in V_H is specified.

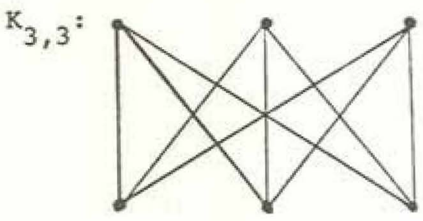
II.2 Motivation and Applications

The concept of subgraph homeomorphism is not new to graph theory. In 1930, G. Kuratowski established that a necessary and sufficient condition for a graph G to be planar is that there is neither a homeomorphism from the complete graph on five points, K_5, to a subgraph of G, nor a homeomorphism from the complete bipartite graph on two sets of three nodes each, $K_{3,3}$, to a subgraph of G [Be, p. 211]. (See Figure II.2-1.) This characterization of planarity has been extended to define a family of properties, P_n, such that graph G has property P_n if neither K_{n+1} nor $K_{\lfloor (n+1)/2 \rfloor +1, \lceil (n+1)/2 \rceil}$ are homeomorphic to a subgraph of G. [Ge, pp. 37-47] Here, K_n is the complete graph on n nodes, and $K_{p,q}$ is the complete bipartite graph on one set of p nodes and one set of q nodes. Given this definition, planarity is property P_4.

The homeomorphism from a graph H to a graph G reflects the structural properties of G represented by H. For example, to see if G contains a tree-like structure, we would seek a homeomorphism from the desired tree to G. As another example, consider the graph representing flow of control of an ALGOL program. The nodes of G

Figure II.2-1 The Kuratowski Graphs





are the program statements, and edges go from each statement to possible next executed statements. In this example, G is directed. To decide if the potential exists to execute a loop containing three particular statements, we would ask if the directed graph which is a cycle of length three is homeomorphic to a subgraph of G with ν specified. In a more rigorous, but similar application, Hunt et. al. present properties of a programming scheme which can be characterized by the reachability of certain substructures of the programming scheme [Hunt]. They pose the question of how complex a structure can be and still allow a polynomial time algorithm for finding it in a programming scheme. This is essentially the question of how complex H can be while the homeomorphic subgraph problem for H is solvable in polynomial time.

The family of properties presented by Hunt et. al. is an example of properties characterized by forbidden subgraphs. Property P is characterized by forbidden subgraphs if G has P if and only if G contains no subgraphs isomorphic to any of a family of graphs determined for P. When a homeomorphism is used to characterize P, the pattern graph (or graphs) which must not be homeomorphic to any subgraph of G defines an infinite family of graphs which must not be isomorphic to any subgraph of G. This infinite family is produced by generating all possible graphs obtainable from the pattern graph by inserting nodes on the edges of the pattern graph. When we are testing for properties characterized by forbidden subgraphs, we are interested in instances of the floating subgraph homeomorphism problem.

The fixed subgraph homeomorphism problem is applicable when information about specific nodes in a graph is needed, as in the ALGOL example above. Such problems as determining if two disjoint paths exist connecting pairs of given nodes and determining if any simple path exists containing a given set of k nodes can be formulated as fixed subgraph homeomorphism problems. These problems, as well as examples of floating and partially specified subgraph homeomorphism problems, will be discussed in the following chapters. In the next section, however, we return to a comparison of edge disjoint versus node disjoint homeomorphism.

II.3 Edge Disjoint Homeomorphism versus Node Disjoint Homeomorphism

In this section, we consider the relationship between the subgraph homeomorphism problem for edge disjoint homeomorphism versus node disjoint homeomorphism. We shall say that one type of subgraph homeomorphism problem is reducible to another if, given H and G (and ρ) of the first type, we can find H' and G' (and ρ') of the second type such that H is homeomorphic to a subgraph of G if and only if H' is homeomorphic to a subgraph of G'. We require that H' and G' can be constructed in a number of steps polynomial in the sizes of G and H, and that the construction for H' is independent of G. Then any algorithm which can determine if H' is homeomorphic to a subgraph of G' in polynomial time can be used to determine if H is homeomorphic to a subgraph of G.

Below we present several reductions for node disjoint and edge disjoint homeomorphisms. We begin by considering only problems where ν is fixed, since the control we have by knowing ν simplifies the constructions needed. The tables in Figure II.3-1 summarize the results we will present.

Lemma II.3.1 Any fixed node disjoint subgraph homeomorphism problem for directed graphs is reducible to a fixed edge disjoint subgraph homeomorphism problem for directed graphs.

Proof: Given directed graphs H and G with ν specified, we construct H' and G' as follows. For each node v in H, H' will have two nodes--HEAD(v) and TAIL(v)-- connected by an edge from HEAD(v) to TAIL(v). Each edge (u,v) in H is reproduced in H' by an edge from TAIL(u) to HEAD(v). Graph G' is constructed in exactly the same manner. Mapping ν' matches HEAD nodes in H' with HEAD nodes

Figure II.3-1 : Summary of Reductions

a: For directed graphs:

is reducible to → ↓	$H \leq_N G$ fixed	$H \leq_N G$ partial or floating	$H \leq_E G$ fixed	$H \leq_E G$ partial or floating
$H \leq_N G$, fixed	=	---	Lemma II.3.1	---
$H \leq_N G$, partial or floating	see Chapter III	=	---	---
$H \leq_E G$, fixed	Lemma II.3.3	---	=	---
$H \leq_E G$, partial or floating	---	Lemma II.3.5	see Chapter III	=

b: For undirected graphs;

is reducible to → ↓	$H \leq_N G$ fixed	$H \leq_N G$ partial or floating	$H \leq_E G$ fixed	$H \leq_E G$ partial or floating
$H \leq_N G$, fixed	=	---	---	---
$H \leq_N G$, partial or floating	see Chapter III	=	---	---
$H \leq_E G$, fixed	Lemma II.3.2	---	=	---
$H \leq_E G$, partial or floating	---	Lemma II.3.4	see Chapter III	=

in G' and TAIL nodes with TAIL nodes consistent with the mapping from H to G. (See Figure II.3-2.) Our construction can be executed by processing all the nodes, followed by all the edges of H and G in a number of steps proportional to $|V_H|+|E_H|+|V_G|+|E_G|$.

It is left to show that $H \leq_N G$ if and only if $H' \leq_E G'$. Suppose we have $\alpha: E_H \rightarrow P(G)$ such that (ν,α) is a node disjoint homeomorphism from H into G. We construct α' to map (HEAD(v), TAIL(v)) edges of H' into corresponding (HEAD(ν(v)),TAIL(ν(v))) edges of G'. Each (TAIL(u),HEAD(v)) edge of H' is mapped into the path of G' which corresponds to path α(u,v) of G. Since the paths of $\alpha(E_H)$ are node disjoint up to endpoints, the paths of $\alpha'(E_{H'})$ are edge disjoint. This can be seen by noting that the only edges on which two paths of $\alpha'(E_{H'})$ could collide would be (HEAD(v), TAIL(v)) edges, but this would imply that the coresponding paths in $\alpha(E_H)$ collide on node v, which is not an endpoint.

Now suppose that we have $\alpha': E_{H'} \rightarrow P(G')$ such that (ν',α') is an edge disjoint homeomorphism from H' into G'. Consider any path in P(G') which is the image of a (TAIL(u),HEAD(v)) edge in H'. This path must start at a TAIL node, go to a HEAD node, and alternate (TAIL,HEAD) and (HEAD,TAIL) type edges. It can be contracted to form a corresponding path in G which is the image of (u,v) in H. If any two paths in $\alpha(E_H)$ so constructed collide on a node v, which is not an endpoint, then the corresponding paths in $\alpha'(E_{H'})$ both contain (HEAD(v), TAIL(v)), and are not edge disjoint.[1]

$\square$

[1]Note that in notational convention, edge (u,v) has tail u and head v. Thus, HEAD(v) is the head node for all (u,v) edges in G; TAIL(v) is the tail of all the (v,u) edges in G.

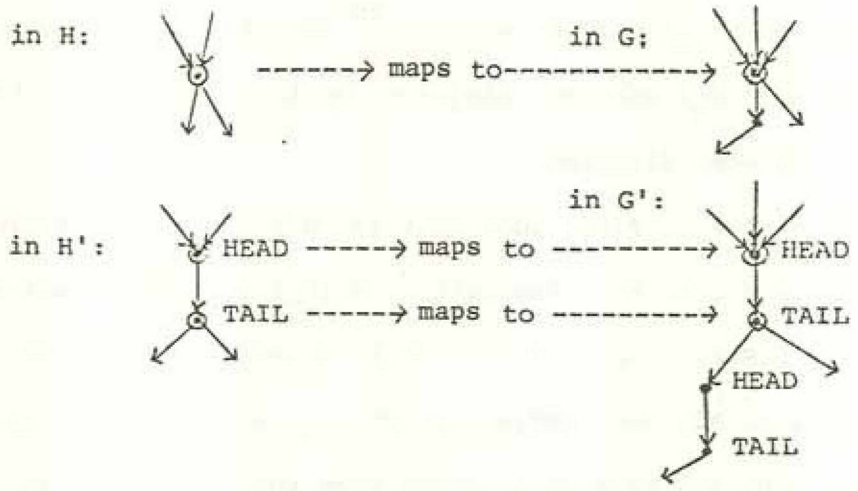
Note that the above construction is essentially the same as that used to change vertex capacities to edge capacities in network flow problems [Ta 1974].

Lemma II.3.2 Any fixed edge disjoint subgraph homeomorphism problem for undirected graphs is reducible to a fixed node disjoint subgraph homeomorphism problem for undirected graphs.

Proof: Given undirected graphs H and G with ν specified, we will construct G' and specify ν' such that $H \leq_E G$ if and only if $H \leq_N G'$. For any node $v \varepsilon V_G$, suppose v has degree d_v. Number the edges of v arbitrarily from 1 through d_v. (Note that any edge will have two numbers, one for each endpoint.) In G', we replace v with d_v nodes -- one for each edge of v. All edges between these nodes are placed in G' to form a complete graph on d_v nodes, which we will denote K_v. For each edge (u,v) of G, if (u,v) is the i^{th} edge of u and the j^{th} edge of v, then there is an edge in G' from the i^{th} node of K_u to the j^{th} node of K_v. In addition, if $v=\nu(u)$, we add an extra node $\hat{v}$ to G' with an edge from $\hat{v}$ to each node in K_v. We define $\nu'(u)=\hat{v}$. (See Figure II.3-3.) The construction can be executed node by node in a number of steps less than $c|E_G|^2$, where c is a constant (i.e. in $\mathcal{O}(|E_G|^2)$).

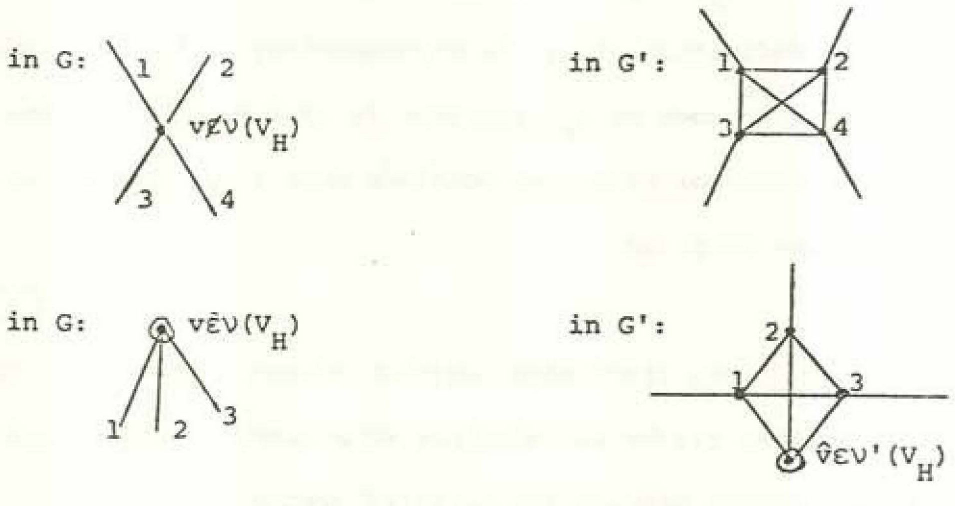
Suppose $\alpha : E_H \rightarrow P(G)$ such that (ν,α) is an edge disjoint homeomorphism from H into G. Let $p=\alpha(x,y)$. In G', $p'=\alpha'(x,y)$ will start at $\nu'(x)$, a $\hat{v}$-type vertex, and go to $\nu'(y)$, a $\hat{v}$-type vertex. The path p' will contain the edges in G' corresponding to those of p in G. Edges in p' corresponding to consecutive edges in p are connected by one edge in the complete graph for their common endpoint on p. The path p' is completed by using the appropriate edge from $\nu'(x)$

Figure II.3-2: Construction for Lemma II.3.1



⊙ indicates a node whose image in V_G or inverse image in V_H is specified.

Figure II.3-3: Construction for Lemma II.3.2



⊙ indicates a node whose image or inverse image is specified.

to $K_{\nu(x)}$ and the appropriate edge from $\nu'(y)$ to $K_{\nu(y)}$. If two paths in $\alpha'(E_H)$ collide on a node, say the ith node of K_v, then the corresponding paths in $\alpha(E_H)$ collide on the ith edge of v. We conclude that the paths of $\alpha'(E_H)$ are node disjoint (up to endpoints) if the paths of $\alpha(E_H)$ are edge disjoint.

Now suppose $\alpha':E_H \to P(G')$ such that (ν',α') is a node disjoint homeomorphism from H into G'. Any path in $\alpha'(E_H)$ must start and end at $\hat{v}$-type nodes. Suppose a path in $\alpha'(E_H)$ contains two or more consecutive edges within one complete graph, K_w, $w\varepsilon G$. We can replace this subpath of p by one edge in K_w going from the first node on this subpath to the last. Therefore, we may assume that any path in $\alpha'(E_H)$ alternates edges corresponding to edges in G with edges in the complete graphs. Thus, each path in $\alpha'(E_H)$ has a corresponding path in G. We define $\alpha:E_H \to P(G)$ using this correspondence. Suppose two paths in $\alpha(E_H)$ collide on some edge, say the ith edge of node v. By our definition of α, the corresponding paths in G' must both contain the ith node of K_v, implying (ν',α') does not define a node disjoint homeomorphism. We conclude that (ν,α) defines an edge disjoint homeomorphism.

$\square$

<u>Lemma II.3.3</u> Any fixed edge disjoint subgraph homeomorphism problem for directed graphs is reducible to a fixed node disjoint subgraph homeomorphism problem for directed graphs.

Proof: The construction is very similar to that in the proof of Lemma II.3.2 and will be only briefly described. Let $v\varepsilon G$ have indegree IN_v and outdegree OUT_v. In G', v is replaced by IN_v nodes called head nodes and OUT_v nodes called tail nodes. There is an edge from each head node for v to each tail node for v. This graph of head and tail nodes corresponds to the graph K_v in Lemma II.3.2. In addition, for each $v \in \nu(V_H)$ there is a node $\hat{v}$, an edge from each head node of v to $\hat{v}$, and an edge from $\hat{v}$ to each tail node of v. We define $\nu'(x) = \hat{v}$ if $\nu(x)=v$. Finally, if edge (u,v) in G is the ith edge out of u and the jth edge into v, then, in G', there is an edge from the ith tail node of u to the jth head node of v. (See Figure II.3-4.)

The remainder of the proof follows closely that of Lemma II.3.2 and is omitted here.

$\square$

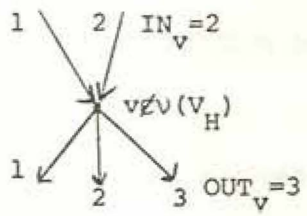
When ν is not completely specified, the previously described constructions may result in an instance of H being embedded in one of the constructs. The tactic we will use to avoid this is to control the degrees of nodes in the construction so that we have control over what nodes will be paired by ν'. In the lemmas below, ρ may be empty, in which case we have a floating problem. The constructions described below may be used in the fixed case. However, they are more complicated than those used specifically for the fixed case and would not be preferred.

Lemma II.3.4 Any edge disjoint subgraph homeomorphism problem for undirected graphs is reducible to a node disjoint subgraph homeomorphism for undirected graphs.

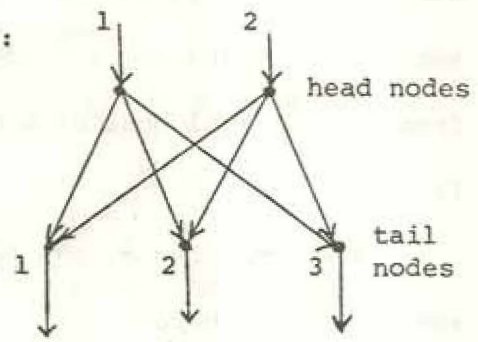
Proof: We construct H' and G' such that $H \leq_E G$ if and only if $H' \leq_N G'$. To do this, we will construct for each node, v, in G, a graph N_v such that only one node in N_v has degree ≥ 4. In H', each node corresponding to a node in H will have degree ≥ 4. To do this, first consider H'. The graph H' will contain all nodes and edges

Figure II.3-4: Construction for Lemma II.3.3

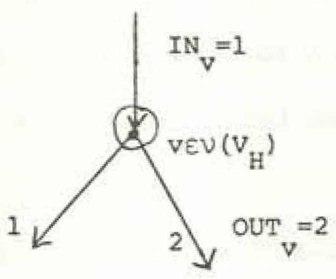
in G:



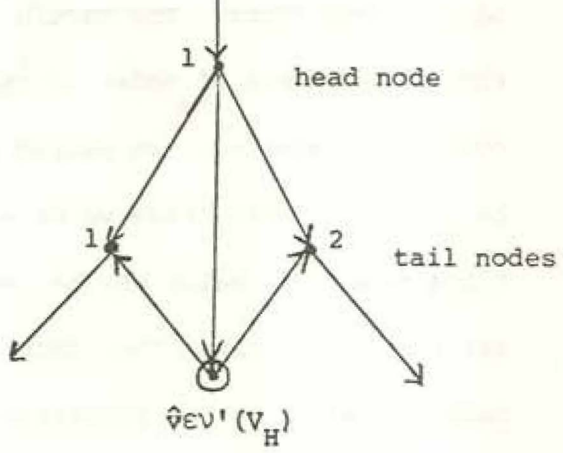
in G':



in G:



in G':

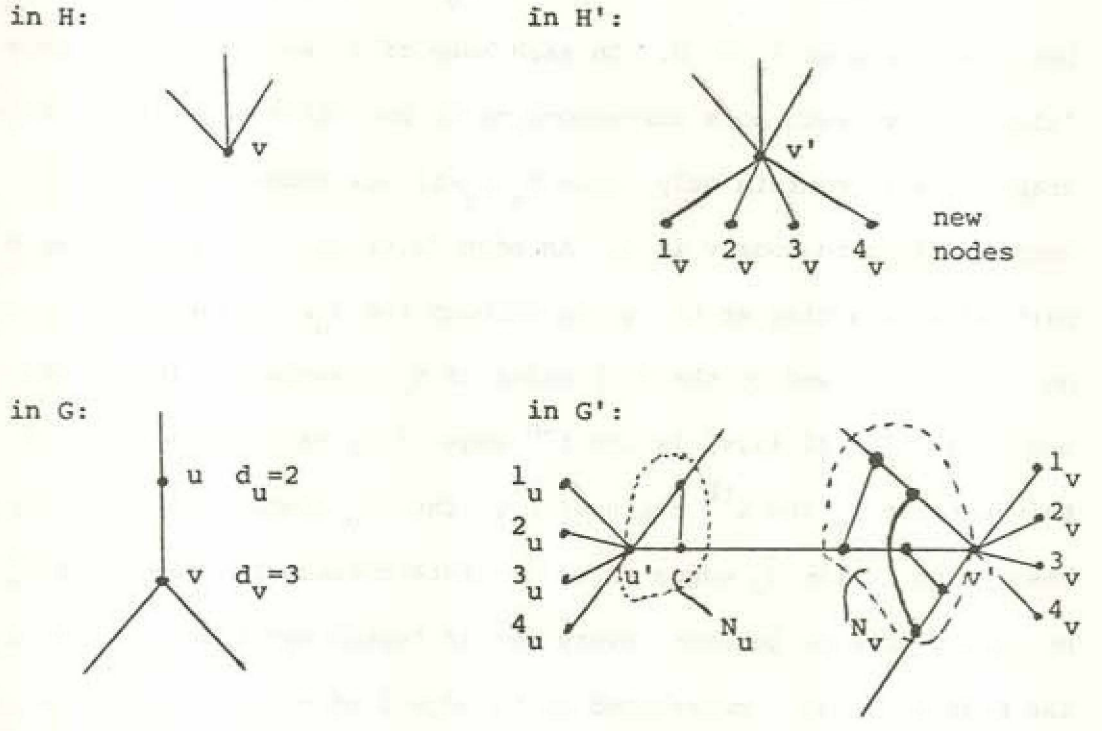


$\odot$ indicates a node whose image or inverse image is specified.

of H. In addition, for each node v in H we add four new nodes to H', denoted $1_v, 2_v, 3_v$, and 4_v, and connect each of these nodes to node v' in H' (corresponding to v in H) by an edge. Thus, H' has V_H nodes of degree ≥ 4 and $4|V_H|$ nodes of degree 1.

Now consider G'. We construct N_v for each node $v \epsilon V_G$ as follows. Let v have degree d_v in G. On each edge of v, we insert d_v-1 nodes "close to" v, each node corresponding to one other edge of v. Subgraph N_v will contain only these $d_v(d_v$-1) new nodes plus node v', corresponding to node v in G. An edge (u,v) in G has now become a path in G' starting at u', going through the d_u-1 nodes of N_u inserted on (u,v), followed by the d_v-1 nodes of N_v inserted on (u,v), and ending at v'. If (u,v) is the i^{th} edge of v, call the portion of this path in N_v the i^{th} chain of N_v. Thus N_v contains d_v chains corresponding to the d_v edges of v. We interconnect the chains of N_v by adding an edge between every set of "matching" nodes. That is, the node on chain j corresponding to edge i of v is connected to the node on chain i corresponding to edge j of v, $i \neq j$. This interconnection allows us to simulate a path in G which goes through v without going through v' in G'. The construction of N_v is now complete. Each node of N_v except v' is of degree 3. To insure that v' is of degree ≥ 4, we add four new nodes to G'($1_v, 2_v, 3_v$, and 4_v) and connect them to v'. These nodes and edges are not in N_v. Node v' now has degree d_v+4. Note that nodes in H' which correspond to nodes in H must map under ν' to nodes in G' which correspond to nodes in G, by the degree requirements of these nodes. If ν was partially specified in the original problem by ρ, then ν' is partially specified by ρ' such that $\rho'(u')=v'$ if and only if $\rho(u)=v$. Figure II.3-5 illustrates

Figure II.3-5: Construction for Lemma II.3.4



the construction. The construction can be executed node by node for both H and G. The number of steps required is $\mathcal{O}(|E_G|^2)$.

We must now show that $H \leq_E G$ if and only if $H' \leq_N G'$. Suppose we have (ν,α), an edge disjoint homeomorphism from H into G. Let $\nu(x)=v$. We define ν' such that $\nu'(x')=(v')$ and $\nu'(i_x)=i_v$,$1 \leq i \leq 4$. Consider any path, p, in $\alpha(E_H)$, $p=\alpha(x,y)$. In G', we can define path p' corresponding to p. Path p' will start at $\nu'(x')$ and end at $\nu'(y')$. If path p contains edge (u,v), path p' will contain the corresponding edge from N_u to N_v. Consecutive edges entering a particular N_v are connected by portions of the two appropriate chains in N_v and one interconnecting edge. Thus only "new" nodes of N_v are used as interior nodes of a path. We can now define α'. Let $\alpha'(x',y')=p'$, where $\alpha(x,y)=p$, and $\alpha'(x',i_x)= (\nu'(x'), i_{\nu(x)})$, $1 \leq i \leq 4$. Suppose two paths in $\alpha'(E_{H'})$ collide on some interior node. By our definition of α', this node must be one of the new nodes of some N_v. Suppose this node is on chain j of N_v. A node on chain j can appear on two paths in $\alpha'(E_{H'})$ only if the j^{th} edge of v appears on two paths in $\alpha(E_H)$, contrary to our assumption that (ν,α) is an edge disjoint homeomorphism. We conclude that (ν',α') is a node disjoint homeomorphism from H' into G'.

Now suppose (ν',α') is a node disjoint homeomorphism from H' into G'. The degree requirements of nodes in H' and G' assure us that for any $x \in V_{H'}$, $\nu'(x')=v'$, where $v \in V_G$. We therefore define ν such that $\nu(x)=v$ if and only if $\nu'(x')=v'$. Consider any path p' in $\alpha'(E_{H'})$ such that $p'=\alpha'(x',y')$, $(x,y) \in E_H$. Path p' must consist of subpaths within particular N_v's connected by edges between different N_v's. We define path p in G corresponding to p' by deleting the subpaths within particular N_v's. Edge (u,v) appears on p whenever the edge from N_u to N_v is used in p'. Then $\alpha(x,y)=p$. Suppose two paths in $\alpha(E_H)$ contain the same edge (u,v). The corresponding paths in $\alpha'(E_{H'})$ must both contain the edge from N_u to N_v, contrary to our assumption that (ν',α') is a node disjoint homeomorphism. We conclude that (ν,α) is an edge disjoint homeomorphism from H into G.

$\square$

Lemma II.3.5 Any edge disjoint subgraph homeomorphism problem for directed graphs is reducible to a node disjoint subgraph homeomorphism problem for directed graphs.

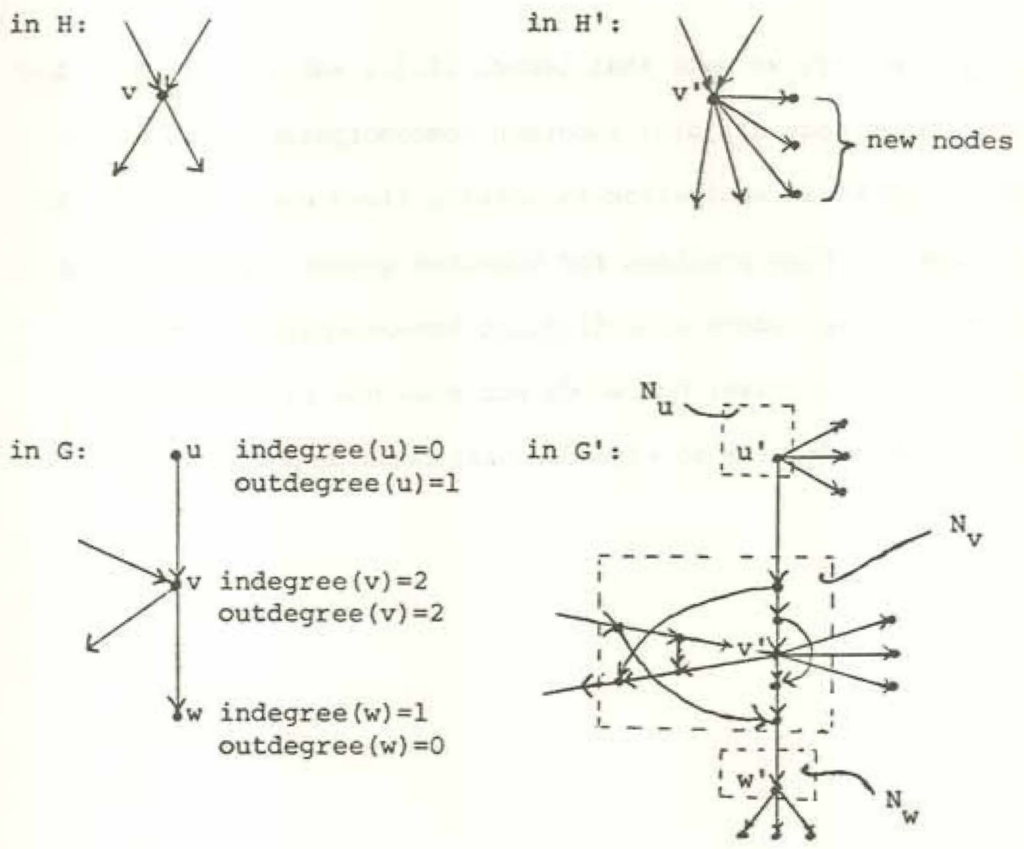
Proof: The construction is very similar to that used for Lemma II.3.4, and is described briefly. We shall control the mapping defined by ν' by controlling the outdegree of each node. In H', three nodes are added for each v' corresponding to v in H, and an edge is added from v' to each of these new nodes. In G', N_v will now consist of two types of chains--in-chains and out-chains. Each edge into a node v of G is changed into an in-chain by inserting outdegree of v new nodes and directing all new edges toward v'. Similarly, each edge out of v is changed into an out-chain by inserting indegree of v new nodes and directing all new edges away from v'. Interconnections are made from each in-chain to each out-chain. Note that the edge between N_u and N_v corresponding to edge (u,v) in G now goes from an out-chain of u to an in-chain of v. Each node on an in-chain has indegree 1 and outdegree 2; each node on an out-chain has indegree 2 and outdegree 1. For each v' in G' corresponding to v in G, we add three new nodes to G' and edges from v' to each. Then the indegree of v' in G' is equal to the indegree of v in G; the outdegree of v' is equal to the outdegree of v plus three. Figure II.3-6 illustrates the construction.

Since the remainder of the proof parallels that of Lemma II.3-4, we omit it here.

□

In summary, we note that Lemmas II.3.1 and II.3.3 imply that solving fixed node disjoint subgraph homeomorphism problems for directed graphs is equivalent to solving fixed edge disjoint subgraph homeomorphism problems for directed graphs. For all other problems, we can reduce edge disjoint homeomorphism to node disjoint homeomorphism, but we do not know how to reduce node disjoint homeomorphism to edge disjoint homeomorphism.

Figure II.3-6: Construction for Lemma II.3.5



III Methods of Solution

III.1 Foundations

In this chapter, we investigate methods of solving node disjoint subgraph homeomorphism problems, hereafter referred to simply as subgraph homeomorphism problems. Our approach is to find reductions which allow us to solve a particular subgraph homeomorphism problem by solving several instances of a subgraph homeomorphism problem for which we have a polynomial time algorithm. We limit the number of instances to be at most a polynomial in the size of the input graph, G, and require the reductions to be executable in polynomial time. Therefore, the original problem can also be solved in a polynomial number of steps in the size of G.

The foundation of our solutions will be the class of fixed subgraph homeomorphism problems in which H is a tree of depth one. This problem is treated as a network flow problem with unit vertex and edge capacities. Definitions and algorithms for the network flow problem can be found in [Hu, pp. 105-111] and [Ta 1974]. For our application, a network is a directed graph N = <V,E> with one node, s, identified as the source and one node, t, identified as the sink. The source has indegree = 0, and the sink has outdegree = 0. To each edge of N, we assign a non-negative integer capacity, c(v,w), and to each node other that s and t we assign a non-negative integer capacity, c(v). A flow, f, in the network is a real-valued function from $V \times V$ such that:

i) $f(v,w) = 0$ if $(v,w) \notin E$

ii) the flow on each edge of N, $f(v,w)$, is non-negative and does not exceed the capacity of the edge, $c(v,w)$.

iii) for each node, v, except s and t, the flow into that node ($\sum_{w \in V} f(w,v)$) is equal to the flow out of that node ($\sum_{w \in V} f(v,w)$) and does not exceed the node capacity, $c(v)$,

The value of the flow, $v(f)$, is the flow out of s, ($\sum_{w \in V} f(s,w)$), which, by the conditions above, is equal to the flow into t.[1] The network flow problem is as follows: Given network N, find the maximum of $v(f)$ over all flows in N. When the edge and node capacities are integer, there always exists an integer-valued maximum flow [Fo, p. 19]. It has been shown [Ta 1974, Ev 1975] that an algorithm by Dinic [Di] to solve the network flow problem executes $\mathcal{O}(|V|^{1/2}|E|)$ operations for networks with unit vertex and edge capacities. Thus, we may use this algorithm a polynomial number of times in our algorithms for subgraph homeomorphism.

To find a subgraph homeomorphism from H into G when H is a directed tree of depth one, and ν is specified, we transform G into a network N_G. Suppose H has root r and leaves $l_1, \ldots, l_k$. In G, we make $\nu(r)$ the source, removing all incoming edges. We add a new node, t, to G, which is the sink, and connect t to each of $\nu(l_i)$, $1 \leq i \leq k$, by an edge $(\nu(l_i), t)$. All edge and vertex capacities are one. If $H \leq_N G$, then the k paths corresponding to the edges of H define a flow of k from $\nu(r)$ through $\nu(l_i)$, $1 \leq i \leq k$, to t. If there is a flow

[1] This definition is a modification of that presented in [Ta 1974]. The definition presented there includes the one presented above, but is more general.

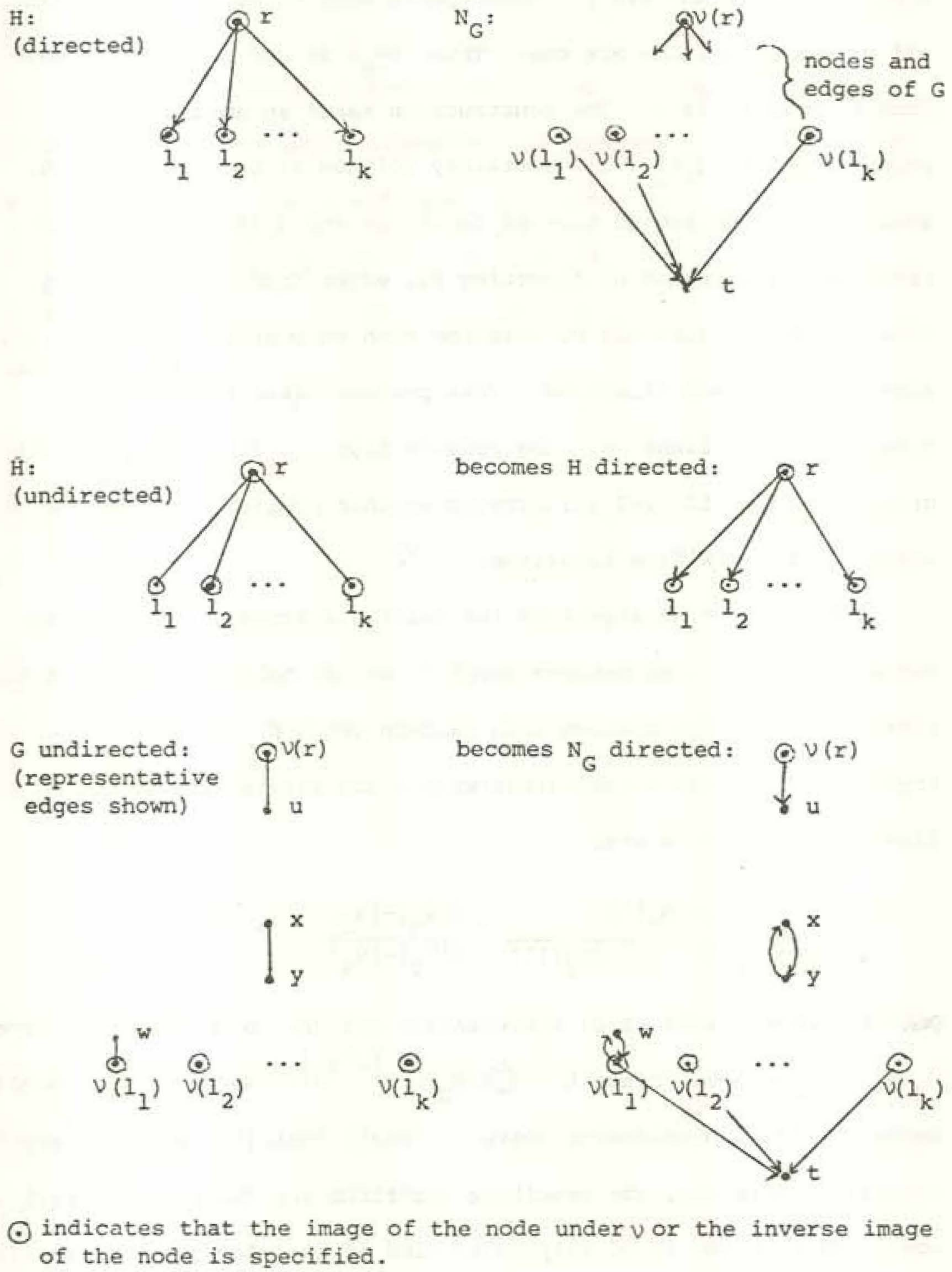
from $\nu(r)$ to t with value equal to k, then there is an integer-valued flow. This flow defines k paths from $\nu(r)$ to t, each path going through a different $\nu(l_i)$. These paths must be node disjoint since all vertex capacities are one. Thus, $H \leq_N G$ if and only if the maximum flow in N_G is k. The construction takes an amount of time proportional to $(|E_G| + k)$, certainly polynomial in the size of G. When H is an undirected tree of depth one and G is undirected, we first make H directed by directing all edges from root to leaves. Then, we make G directed by changing each edge of G into two directed edges, one in each direction. This process takes time $\mathcal{O}(|E_G|)$. Figure III.1-1 illustrates the construction for directed and undirected graphs. Figure III.1-2 illustrates another problem which can be solved using the network flow algorithm.

Once we have an algorithm for solving a fixed subgraph homeomorphism problem with pattern graph H, we can solve any floating or partially specified problem with pattern graph H. We do this by trying all possible ν consistent with ρ and solving the resulting fixed problem. There are:

$$\frac{[|V_G|-|N_H|]!}{[(|V_G|-|N_H|)-(|V_H|-|N_H|)]!} = \frac{[|V_G|-|N_H|]!}{[|V_G|-|V_H|]!} = \Omega$$

possible completions of ρ, representing all one to one function from V_H-N_H to $V_G-\rho(N_H)$. Since $\Omega = \mathcal{O}(|V_G|^{|V_H|-|N_H|})$, we have a polynomial number of fixed problems to solve. (Recall that $|V_H|$ and $|N_H|$ are constants.) Therefore, the resulting algorithm for floating and partially specified problems is of polynomial time if the algorithm for the fixed

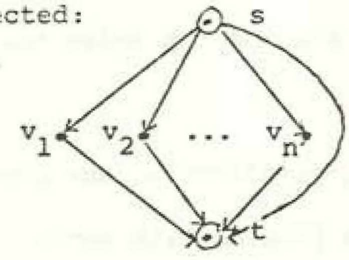
Figure III.1-1 Construction when H is a tree of depth 1.



⊙ indicates that the image of the node under ν or the inverse image of the node is specified.

Figure III.1-2 Other problems solvable using network flow.

for H directed:



G directed becomes N_G:
(unit vertex and
edge capacities)

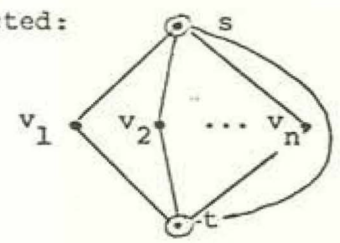
⊙ $\nu(s)$ becomes source-- remove incoming edges in G

} nodes and edges of G

⊙ $\nu(t)$ becomes sink-- remove outgoing edges in G

$H \leq_N G$ iff maximum flow in N_G is $\geq n+1$. Since all paths which constitute the flow must be of length ≥ 2 except possibly one path (if $(\nu(s),\nu(t)) \varepsilon E_G$), we can identify $\nu(v_i)$, $1 \leq i \leq n$, on these paths.

If H undirected:



then direct H as above.

and for G undirected, direct G as for depth one tree problem.

⊙ indicates that the image of the node under ν or its inverse image is specified.

problem is of polynomial time. In particular, whenever the pattern graph H is a tree of depth one, we can solve the subgraph homeomorphism problem in polynomial time.

The above reduction generalizes to any partially specified problem. If we can solve a problem with pattern graph H and $N_H \neq \emptyset$, we can solve any problem with pattern graph H and $N_H' \subseteq N_H$. However, this reduction does not simplify H itself by removing nodes or edges. In Section III.2, we present two reductions which do simplify the pattern graph.

III.2 Reductions

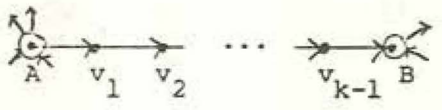
We shall present two special purpose reductions which alter both the pattern graph, H, and the input graph, G. In both cases, ν will be partially specified. These reductions can be combined with the general reduction described in Section III.1 to further expand the number of subgraph homeomorphism problems for which we have polynomial time algorithms. The reductions will be presented in the directed case, but completely analogous reductions exist for the undirected case.

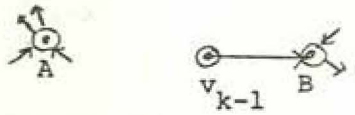
The first reduction is applicable when H contains a path of length k, k>1, from a node $A \in N_H$ to a node $B \in N_H$ on which all interior nodes are not in N_H and have indegree one and outdegree one. Call this path P_H. Nodes A and B need not be distinct. (In fact, B may be absent, in which case the last node of the path is of indegree one, outdegree zero, and is not in N_H.) Any corresponding path in G under a homeomorphism (ν,α) must be of length $\geq k$. Each of the interior nodes on this path will not appear on any other path in $\alpha(E_H)$. Therefore, we can assume that ν maps the interior nodes of P_H to the first k-1 interior nodes in the path of G. Correspondingly, α maps the first k-1 edges of P_H to the first k-1 edges of the path in G and maps the last edge of P_H to the remainder of the path in G. Given this, we can use the following reduction. For any input graph G and partial specification $\rho:N_H \to V_G$, generate all length k-1 paths from $\rho(A)$ which contain no nodes in $\rho(N_H)$ other than $\rho(A)$. Since k is a constant, even the crudest methods of exhaustive enumeration, taking $\mathcal{O}(|V_G|^k)$ steps are still executable in time a polynomial in the size of G. For each generated path P, we do the following. Extend ρ to map each interior node on P_H to the corresponding node of the generated path. Construct H' by removing the first k-2 interior nodes on P_H and the edges associated with them. The last interior node on P_H is now in $N_{H'}$. In G, remove the interior nodes of the path P and all edges associated with them , yielding G'. Specification $\rho':N_{H'} \to V_{G'}$ is the restriction of the extended ρ to nodes of $N_{H'}$. Now solve the subgraph homeomorphism problem for pattern graph H' and inputs G' and ρ'. If this problem can be solved in polynomial time, the original problem can be solved in polynomial time by solving at most $\mathcal{O}(|V_G|^k)$ instances of the problem for H' and G'. Figure III.2-1 illustrates the construction for both directed and undirected graphs. Figure III.2-2 gives two pattern graphs H for which the subgraph homeomorphism problem can be solved using this reduction.

The second reduction is applicable when H contains a node A in N_H which is adjacent to k ($k \geq 1$) nodes not in N_H, each of indegree one and outdegree zero. Label these nodes $l_1,\ldots,l_k$. Suppose (ν,α) is a homeomorphism from H to an input graph G consistent with partial specification ρ. If α maps edge (A, l_i), for some i, to a path of length greater than one, say P_i, none of the nodes on this path will be on any other path in $\alpha(E_H)$. We can alter ν and α so that (A, l_i) maps to the first edge of P_i and l_i maps to the endpoint of this edge without changing any other values of ν and α. The new mappings still constitute a homeomorphism consistent with ρ. Given this, we construct H' and G' as follows. Consider all the nodes in $V_G-\rho(N_H)$

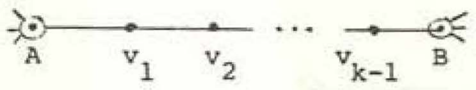
Figure II.2-1 The First Reduction

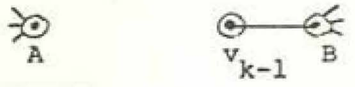
H directed containing: reduced to H' directed containing:



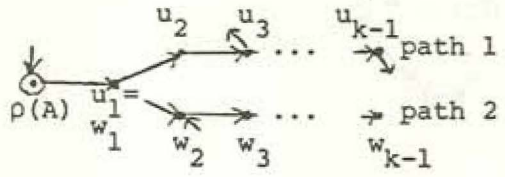


H undirected containing: reduced to H' undirected containing:

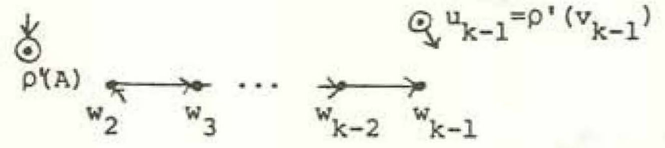




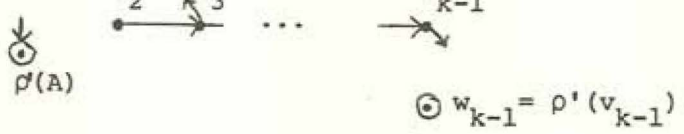
G directed containing:



reduced to G' for path 1 containing:



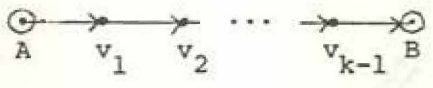
and to G' for path 2 containing:

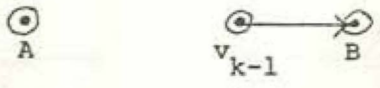


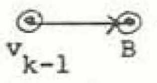
⊙ indicates a node in N_H or $\rho(N_H)$.

Figure III.2-2: Problems solvable using the first reduction.

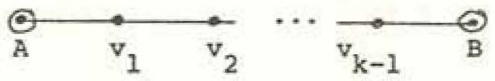
a: H_1 directed: reduces to H_1':

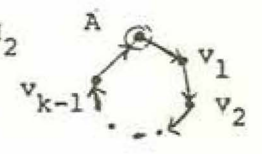
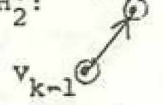




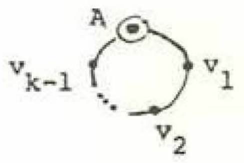
H_1' solved by removing $\rho(A)$ in G' and solving  .

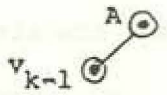
Similarly, H_1 undirected: is solved.



b: H_2  reduces to H_2':  solvable.

Similarly, H_2 undirected: reduces to H_2':



 solvable.

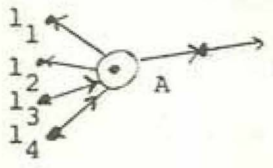
$\odot$ indicates that the node is in N_H.

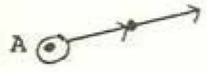
Here, solvable means solvable by a polynomial time algorithm.

adjacent to $\rho(A)$ by edges from $\rho(A)$. These are the candidates for images of the l_i under ν. For each set of k of these nodes, we construct G' by removing the k nodes and all edges associated with them. These k nodes will be the images of the l_i under ν; the actual correspondence is arbitrary. Graph H' is constructed by removing l_i, $1 \leq i \leq k$, and the edge from A to each. Then $H \leq_N G$ if and only if for some G' so constructed, $H' \leq_N G'$. We construct at most $\binom{\text{outdegree } \rho(A)}{k}$ (equal to $\mathcal{O}(|V_G|^k)$) different graphs G', each construction executable in at most $\mathcal{O}(|E_G|)$ steps (the time to add back in or delete k nodes adjacent to $\rho(A)$). Therefore, if the subgraph homeomorphism problem for H' with ν specified on $N_{H'} = N_H$ is solvable in polynomial time, the subgraph homeomorphism problem for H with ν specified on N_H is solvable in polynomial time. We may extend this reduction to include nodes adjacent to A with outdegree one and indegree zero. Then, two sets of nodes must be considered for $\rho(A)$, those with edges to $\rho(A)$ and those with edges from $\rho(A)$. Figure III.2-3 illustrates the construction in the most general form. Figure III.2-4 gives two pattern graphs for which we can obtain polynomial time algorithms using the reduction. Note that the solution of H_2 in Figure III.2-4 implies that the subgraph homeomorphism problem for any pattern graph which is a tree of depth two such that the leaves are not in N_H is solvable in polynomial time.

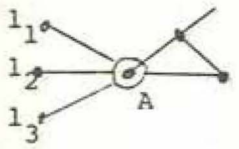
Figure III.2-3: The Second Reduction

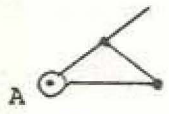
H directed containing: reduces to H' directed containing:



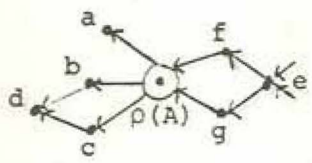


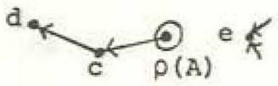
H undirected containing: reduces to H' undirected containing:



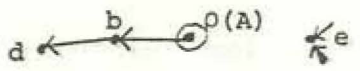


For H directed above,
G directed containing: reduces to G' containing:

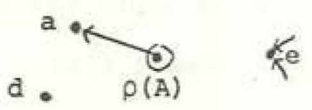




or G' containing:



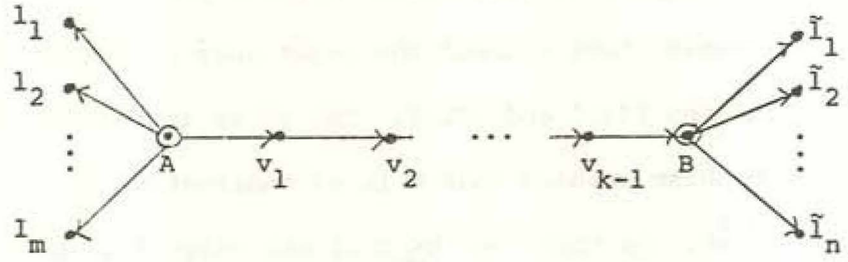
or G' containing:



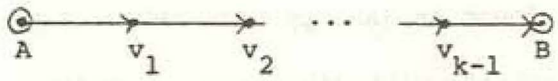
⊙ indicates that the node is in N_H or $\rho(N_H)$

Figure III.2-4 Problems solvable using the second reduction.

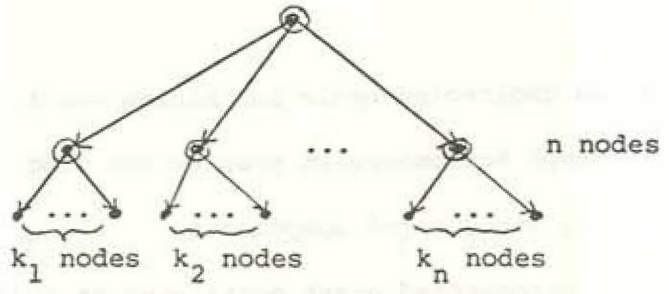
H_1:



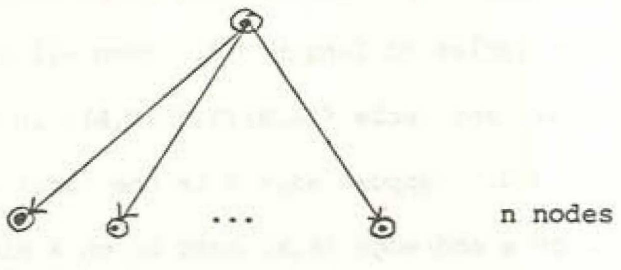
solved using two applications of the reduction and then solving:



H_2:



solved using n applications of the reduction and then solving:



⊙ indicates that the node is in N_H.

Here solvable means solvable in polynomial time.

III.3 Special Cases

We have found three subgraph homeomorphism problems which can be solved in polynomial time without the assistance of the reductions described in Sections III.1 and III.2. The first is the floating subgraph homeomorphism problem when H is an undirected cycle containing exactly three nodes. In this case $H \leq_N G$ if and only if G contains a biconnected component with at least three nodes. Definitions and and $\mathcal{O}(|V_G|+|E_G|)$ time algorithm for determining the biconnected components of G can be found in [Ah, pp. 176-187]. A biconnected graph is an undirected graph in which, given any two nodes, there is a path between them, and given any two edges, there is a simple cycle containing them. The biconnected components of G break up G into biconnected subgraphs.

When H is an undirected cycle containing exactly four nodes, the floating subgraph homeomorphism problem can also be solved in $\mathcal{O}(|V_G|+|E_G|)$ using biconnected components.

Claim: Any biconnected graph containing at least four nodes has a cycle of length ≥ 4.

Proof: Suppose G is a biconnected graph containing at least four nodes but no cycles of length ≥ 4. Then all cycles are of length 3. Consider any cycle <(A,B)(B,C)(C,A)> in G. Now consider a fourth node D in G. Suppose edge e is the first edge on a path from D to A. Edge e and edge (A,B) must be on a simple cycle, and this cycle must be of length 3. Therefore, this cycle can only be <(D,A)(A,B)(B,D)>. But then <(D,A)(A,C)(C,B)(B,D)> is a cycle of length 4 in G, contradicting our assumption.

$\square$

Therefore, $H \leq_N G$ if and only if G contains a biconnected component with at least four nodes.

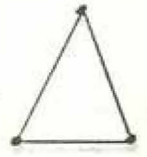
When H (directed or undirected) consists of three nodes, A, B, and v, and two edges, (A,v) and (v,B), and $\rho:\{A,B\} \to V_G$, we can solve the subgraph homeomorphism problem by removing edge $(\rho(A), \rho(B))$ from E_G, if it is in E_G, and looking for a path from $\rho(A)$ to $\rho(B)$ in G minus $(\rho(A), \rho(B))$. This can be done in time $\mathcal{O}(|V_G|+|E_G|)$ using a depth first search of G rooted at $\rho(A)$ (cf. [Ah, pp. 176-187]).

When H is an undirected tree of depth one with exactly two leaves, and ν is specified, an alternate algorithm to the network flow algorithm discussed in Section III.1 has been suggested by R. Rivest and A. Yao [Ri]. The problem is viewed as finding a simple path in G from $\nu(l_1)$ to $\nu(l_2)$ containing $\nu(r)$. (See Figure III.3-1d.) Using a depth first spanning tree of G [Ah, pp. 176-187], the problem is determined to be infeasible or is reduced to finding a simple path from $n_1 \varepsilon V_G$ to $n_2 \varepsilon V_G$ containing $n_3 \varepsilon V_G$, where n_1, n_2, and n_3 are in a biconnected component of G. A simple cycle containing n_1 and n_3 and a path from n_3 to n_2 which does not contain n_1 are then used to construct the desired path. The existence of both the cycle and the path is guaranteed by the properties of biconnected components.

Figure III.3-1 summarizes the special cases discussed above.

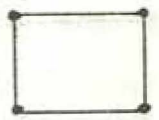
Figure III.3-1: Special Cases

a) H:



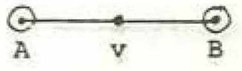
G contains a biconnected component with at least three nodes.

b) H:



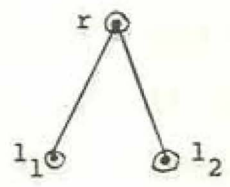
G contains a biconnected component with at least four nodes.

c) H:



$G' = \langle V_G, E_G - \{(\rho(A), \rho(B))\}\rangle$ contains a path from $\rho(A)$ to $\rho(B)$.

d) H:



G: find a path from $\rho(1_1)$ to $\rho(1_2)$ containing $\rho(r)$ using depth first spanning tree.

⊙ indicates that the node is in N_H.

III.4 Conclusion

Although in Section III.2, we have presented two very useful reductions for solving subgraph homeomorphism problems, our knowledge of how to solve these problems is still based on the solution of fixed problems when H is a tree of depth one. In Chapter IV, we extend this foundation for undirected graphs by presenting a linear time algorithm for the fixed subgraph homeomorphism problem when H is a cycle containing exactly three nodes. In Chapter V, we discuss the fixed subgraph homeomorphism problem when H consists of two disjoint edges. This problem remains open and proves to be a fundamental open problem for undirected graphs.

Turning to directed graphs, we are not as fortunate. The fixed subgraph homeomorphism problem for H a tree of depth one is the only fixed subgraph homeomorphism problem for which we have a polynomial time algorithm. The three most basic problems to investigate are:

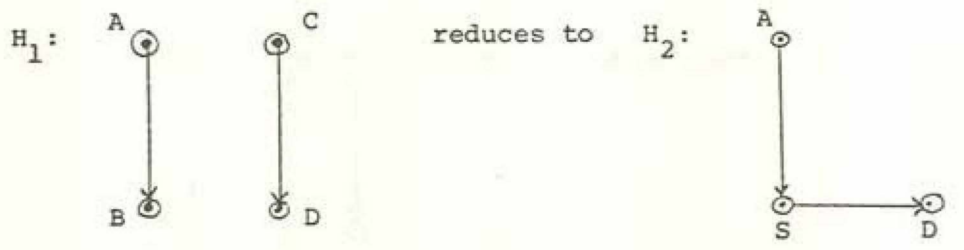
i) H_1 consists of two disjoint edges.

ii) H_2 consists of three nodes, A, B, and C, and two edges, (A,B) and (B,C).

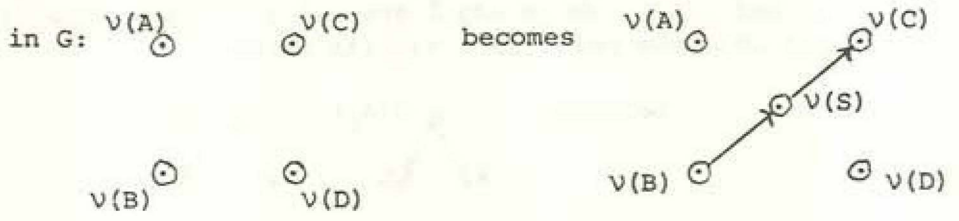
iii) H_3 consists of two nodes and both edges between them.

In each of these problems, the pattern graph has only two edges. Unfortunately, all of these problems are equivalent, as illustrated in Figures III.4-1 through III.4-3. In addition, if we can solve the fixed subgraph homeomorphism problem for H_1, we can certainly solve it when H consists of two undirected disjoint edges. Thus, the fixed problem for H consisting of two undirected disjoint edges not only proves to be a fundamental problem for undirected graphs, but also promises to be the simplest of the open problems discussed above.

Figure III.4-1 Reducing H_1 to H_2

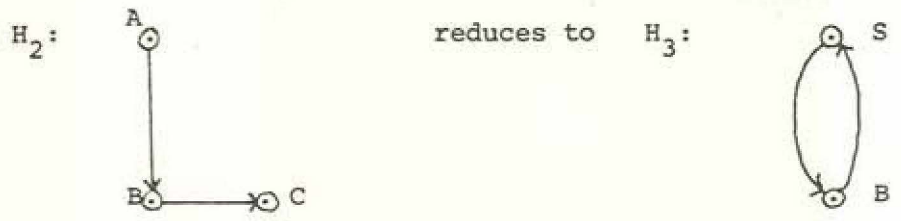


by adding new node ν(S) to G and edges (ν(B),ν(S)) and (ν(S),ν(C)):

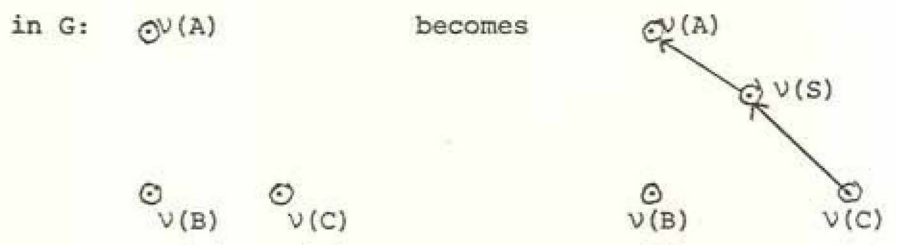


Then any path from ν(A) to ν(S) must go through ν(B) and any path from ν(S) to ν(D) must go through ν(C).

Figure III.4-2 Reducing H_2 to H_3



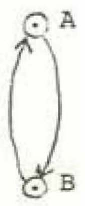
by adding new node ν(S) to G and edges (ν(S),ν(A)) and (ν(C),ν(S)).



Then any path from ν(S) to ν(B) must contain ν(A), and any path from ν(B) to ν(S) must contain ν(C).

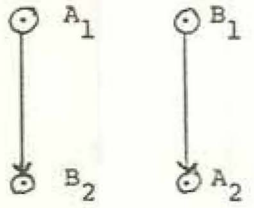
Figure III.4-3: Reducing H_3 to H_1

H_3:



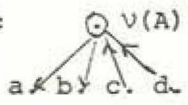
reduces to

H_1:

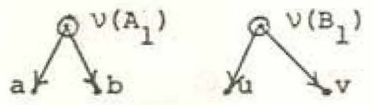


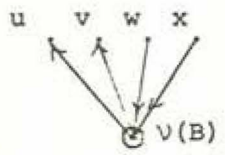
by breaking $\nu(A)$ in G into two nodes $\nu(A_1)$, whose edges are all the edges out of $\nu(A)$, and $\nu(A_2)$, whose edges are all the edges into $\nu(A)$. Node $\nu(B)$ is broken into two nodes in a similar manner.

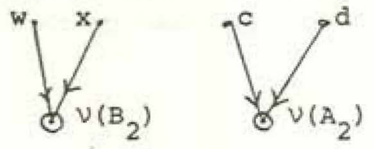
in G:



becomes







Then any path from $\nu(A)$ to $\nu(B)$ is equivalent to a path from $\nu(A_1)$ to $\nu(B_2)$, and any path from $\nu(B)$ to $\nu(A)$ is equivalent to a path from $\nu(B_1)$ to $\nu(A_2)$.

⊙ indicates a node in N_H or $\rho(N_H)$

IV A Linear Time Algorithm

IV.1 Introduction

In this chapter, we present a linear time algorithm for determining if the graph C_3 shown in Figure IV.1-1 is homeomorphic to a subgraph of G when ν is specified. Both C_3 and G are undirected. Let $\nu(a)=A$, $\nu(b)=B$, and $\nu(c)=C$ in G. Then, we would like to find in G a simple cycle containing A, B, and C.

Several concepts are needed before the algorithm can be presented. In this discussion, G is always an undirected graph. Two nodes in a graph are connected if there is a path between them. A graph G is connected if every pair of nodes in G is connected. A set, S, of nodes in G is a (vertex) cutset for two nodes, x and y, in G if every path between x and y contains a node of S. Thus, removing the nodes in S (and the edges incident on them) from G separates x from y. A set S of nodes in G is a (vertex) cutset for G if it is a cutset for some pair of nodes in G. The connectivity of nodes x and y in G, denoted K(x,y), is the minimum number of nodes in a cutset for x and y.[1] In particular, two nodes, x and y, are biconnected if $K(x,y)\geq 2$; two nodes are triconnected if $K(x,y)\geq 3$. The connectivity of a graph G, K(G), is defined to be $\min_{x,y\in V_G} K(x,y)$. A graph is biconnected if $K(G)\geq 2$;[2] a graph is triconnected if $K(G)\geq 3$.

[1]This notation follows that in [Ha 1971, p. 49].

[2]The equivalence of this definition for a biconnected graph and that given in Chapter III is presented in [Ah, pp. 179-182].

Menger's Theorem states that if $K(x,y)\geq n$ for two nodes, x and y, in G, then there are at least n node disjoint paths between x and y [Ha 1971, p. 49].

Suppose we remove a cutset S from a connected graph G. Graph G separates into several connected subgraphs. Denote the connected subgraph containing a particular node $u \varepsilon V_G - S$ by $(G-S)_u$. Using $(G-S)_u$, we will construct what we call the S-component of G containing u, denoted $(G/S)_u$. The notation "$(G/S)_u$" indicates that the only paths from u contained in $(G/S)_u$ are those which either do not contain nodes of S or contain nodes of S as endpoints. Thus, the nodes reachable from u in $(G/S)_u$ are restricted by S. To construct $(G/S)_u$, we add to $(G-S)_u$ all nodes of S and any edges in E_G which go from a node in $(G-S)_u$ to a node in S. Note that we do not add edges between nodes in S. It may be the case that some nodes of S are isolated in $(G/S)_u$. This can only occur if some proper subset of S is also a cutset of G. Figure IV.1-2 illustrates the construction.

We will use the following algorithms within our algorithm for C_3:

(1) An $\mathcal{O}(|V_G|+|E_G|)$ time algorithm for finding the biconnected components of G. This algorithm is due to Hopcroft. [Ah, pp. 176-187] [Ho 1973a] [Ta 1972]

(2) The network flow algorithm discussed in Section III.1. We use this algorithm to find paths in G and cutsets for pairs of nodes. Since we will need to find a flow of at most three using this algorithm, the time taken will be $\mathcal{O}(|V_G|+|E_G|)$ rather than $\mathcal{O}(|V_G|^{1/2}|E_G|)$ for each application. [Ta 1974] [Ev 1975]

The use of the network flow algorithm for finding cutsets of size two is presented in the next section.

Figure IV.1-1

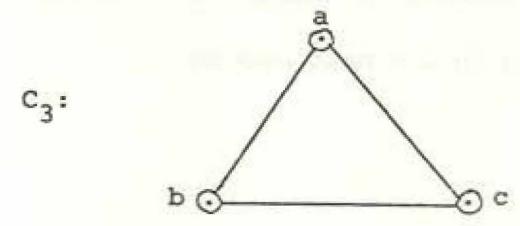
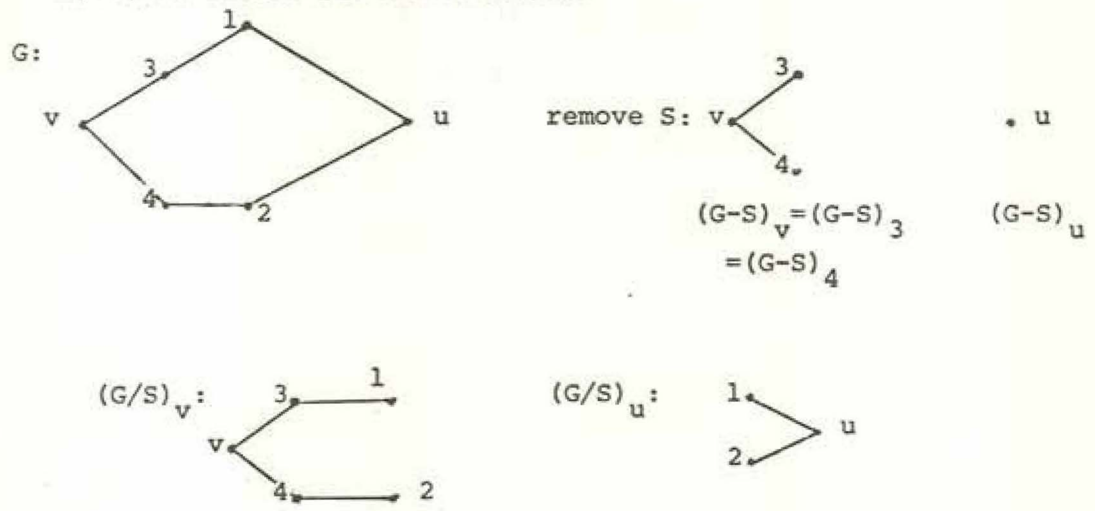
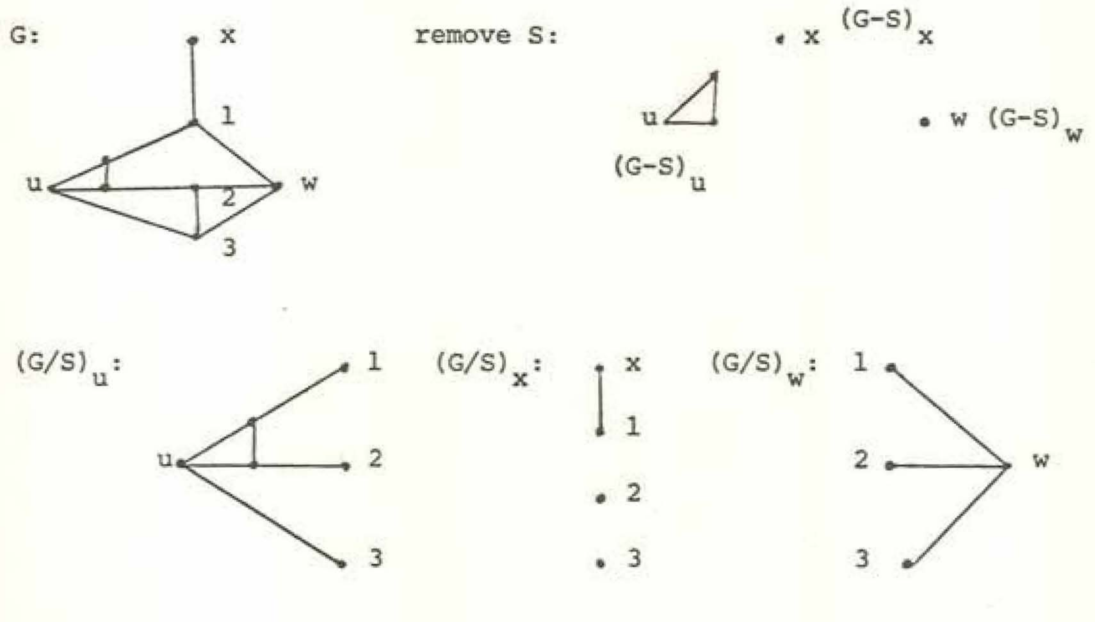


Figure IV.1-2:Construction of $(G/S)_u$

$S = \{1,2\}$; no subset of S is a cutset.



$S = \{1,2,3\}$; $\{1\}$ is also a cutset.



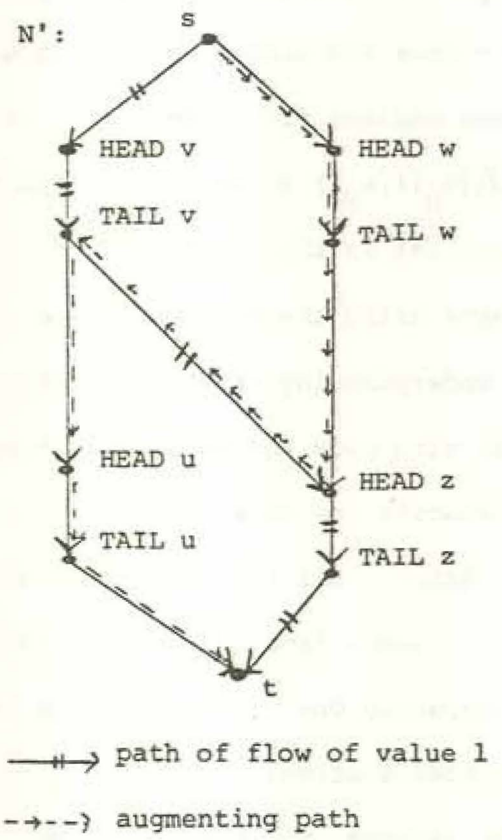
IV.2 Finding Cutsets of Size Two

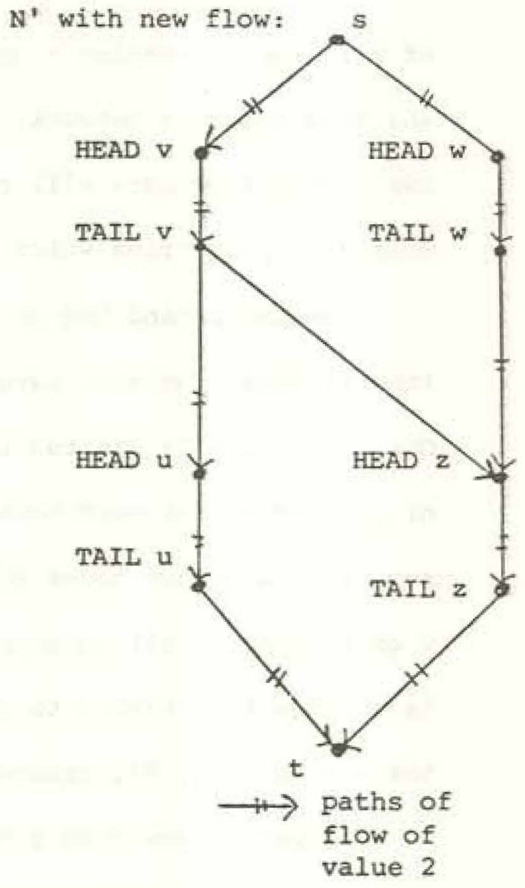
In this section, we outline a procedure for finding a cutset of size two separating s and t, the source and sink of a unit node and edge capacity network, N, when the maximum flow from s to t is two. This procedure will require $\mathcal{O}(|V_N|+|E_N|)$ steps. The cutpoints will have properties which will be crucial in the algorithm for C_3.

To understand how we find cutsets using the network flow algorithm of Dinic, we must have a basic understanding of the algorithm. Our discussion is limited to networks with node and edge capacities of one. Note that each node, v, with capacity one in a network, N, is represented by two nodes, HEAD v and TAIL v. All incoming edges of v go to HEAD v, all outgoing edges of v leave from TAIL v, and there is an edge from HEAD v to TAIL v of capacity one. The algorithm uses the new network, N', created by this modification.

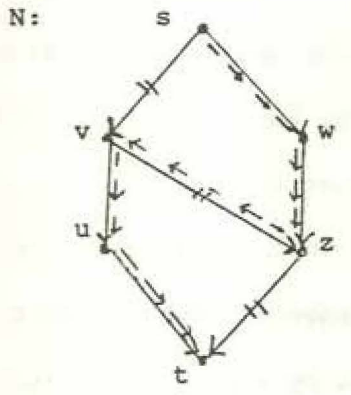
Given a flow from s to t in the network, N', the algorithm proceeds by finding an augmenting path in N' along which flow can be increased while maintaining the edge capacity restrictions on the flow. This augmenting path can use edges not used by the present flow and edges used by the present flow in the opposite direction from that for the present flow. Using an edge in the opposite direction cancels the flow in the edge. Note that the augmenting path in N' corresponds to an augmenting path in N which uses a node on a path of present flow only if at least one edge incident on that node used by the augmenting path is also used by a path of present flow, in the opposite direction. Figure IV.2-1 illustrates the use of an augmenting path.

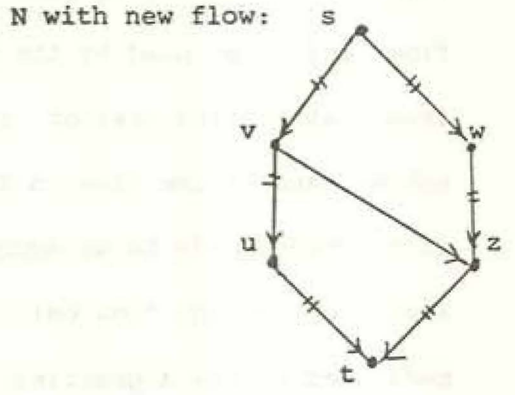
Figure IV.2-1: An Augmenting Path





Correspondence in N:





To find augmenting paths, modify N' so that each edge used by the present flow is replaced by an edge in the opposite direction. Call the new network $\tilde{N}$. A breadth first search of $\tilde{N}$, beginning with node s, is used to create a spanning tree of $\tilde{N}$. This tree contains all nodes reachable from s by an augmenting path. If t is reachable from s, a new augmenting path has been found. If not, the present flow in N' is of maximum value. Details of the algorithm can be found in [Ta 1974]and [Hu, pp. 105-120].

We now show how to find a cutset of size two separating s from t when the maximum flow from s to t is of value two. In the discussion below, we assume that N is a network formed from an undirected graph G containing s and t by replacing each edge of G by two directed edges and removing incoming (respectively outgoing) edges of s(respectively t). Consider the spanning tree, T, of $\tilde{N}$ when the network flow algorithm terminates. Let P_1 and P_2 be the paths from s to t in G corresponding to the flow of two in N' (and N). Let A_1 be the closest node to t on P_1 such that HEAD A_1 is in T. If A_1=s, then, instead, let A_1 be the node adjacent to s on P_1. Define A_2 on P_2 similarly.

<u>Lemma IV.2.1</u> The set$\{A_1,A_2\}$is a cutset separating s from t in G.

Proof: Suppose, to the contrary, that there is a path, Q, in G from s to t which does not contain A_1 or A_2. Let v be the node closest to s, but not equal to s, on Q which is also on P_1 or P_2. This node must be on $P_1[s,A_1]$ or $P_2[s,A_2]$, where P[u,v] denotes the portion of path P from node u to node v. Otherwise, Q[s,v] corresponds to an augmenting path in N' to a node closer to t on P_1 (or P_2) than A_1 (or A_2). Note that since t is on P_1, P_2, and Q, the node v always exists. Let w be the closest node to t on Q which is also on $P_1[s,A_1]$ or $P_2[s,A_2]$. The node w is at least as close to t as v on Q (i.e. w may equal v). Therefore, $w \neq s$. Also, $w \neq A_1$ and $w \neq A_2$ by our assumption that Q does not contain A_1 or A_2. Let z be the closest node to w on Q[w,t] which is also on $P_1[A_1,t]$ or $P_2[A_2,t]$. We know that $z \neq w$, but z may equal t. Without loss of generality, assume w is on $P_1[s,A_1]$. This implies that there is an augmenting path from s to A_1 in N', since $P_1[s,A_1]$ cannot consist of a single edge. Consider the augmenting path constructed from the following segments by removing interior cycles:

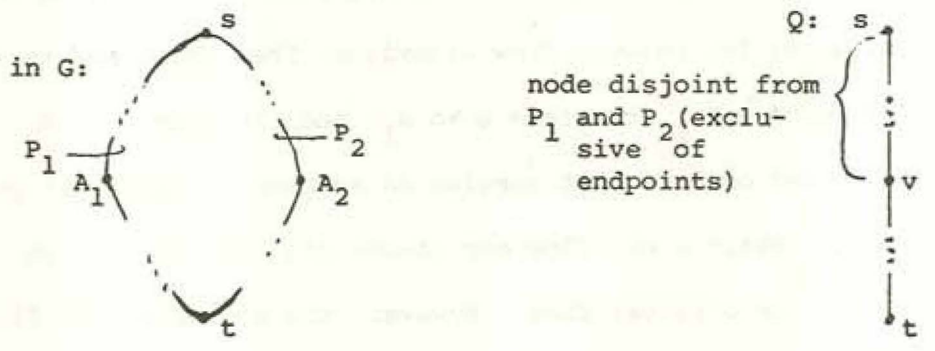
(i) the augmenting path from s to HEAD A_1.
(ii) the path in N' corresponding to $P_1[A_1,w]$. (Note that this is the opposite direction on P_1 from that used in the flow from s to t, and is therefore usable as an augmenting path.)
(iii) the path in N' corresponding to Q[w,z]. (Note that this path is node disjoint from P_1 and P_2 except at w and z.)

This augmenting path goes from s to HEAD z. Since z is on $P_1[A_1,t]$ or $P_2[A_2,t]$, and z is distinct from A_1 and A_2, the existence of this augmenting path is contradictory to our definition of A_1 and A_2. Figure IV.2-2 illustrates one possible configuration of v, w, and z on P_1 and P_2.

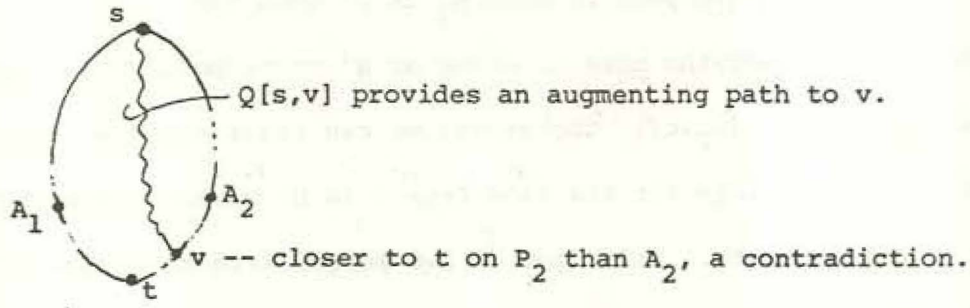
$\square$

We would now like to prove an important property of A_1 and A_2. This property of A_1 and A_2 is crucial in the algoritm for C_3. Let $G_1 = (G/\{A_1,A_2\})_s$.

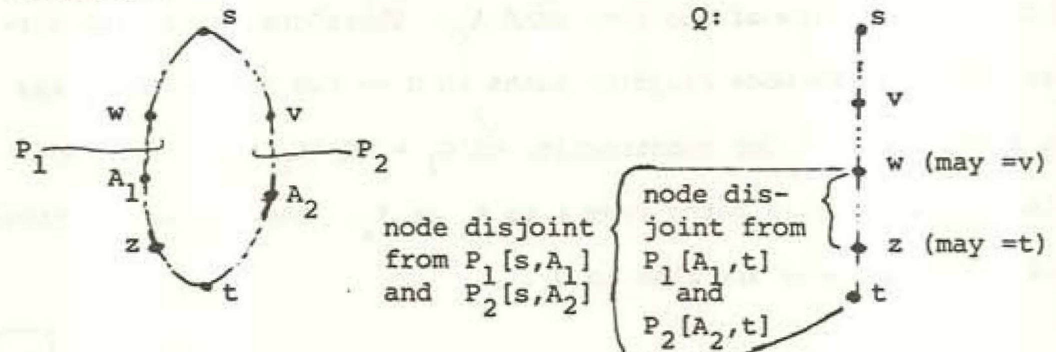
Figure IV.2-2: A configuration of v,w,and z in the proof of Lemma II.2.1.



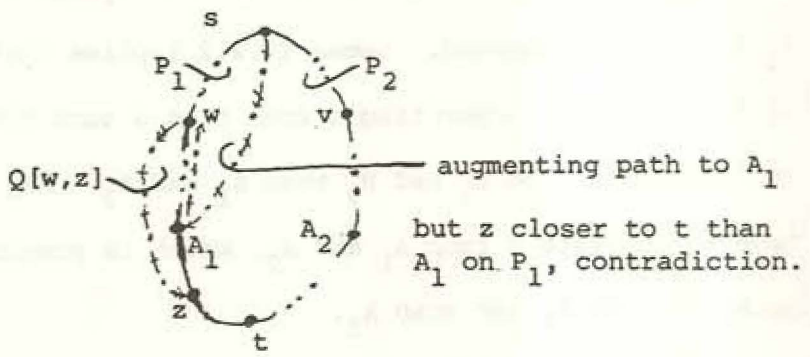
If:


s
Q[s,v] provides an augmenting path to v.
A_1
A_2
v -- closer to t on P_2 than A_2, a contradiction.
t

Otherwise:


s
w
v
P_1
P_2
A_1
A_2
z
t
node disjoint from $P_1[s,A_1]$ and $P_2[s,A_2]$
Q:
s
v
w (may =v)
node disjoint from $P_1[A_1,t]$ and $P_2[A_2,t]$
z (may =t)
t

gives augmenting path in N' corresponding to ⇸⇸⇸:



Lemma IV.2.2 If there is an augmenting path to HEAD A_1 in N' upon termination of the network flow algorithm, then there are three node disjoint paths in G_1, two from s to A_1, and one from s to A_2.

Proof: First observe that merging an augmenting path with an existing flow to obtain a new flow may cancel all flow through an interior node of the original flow. However, the amount of new flow into and out of endpoints of the flow can only increase.

The augmenting path to HEAD A_1 in N' upon termination of the network flow algorithm uses no nodes of N' corresponding to nodes on $P_1[A_1,t]$ or $P_2[A_2,t]$. Therefore, we can regard HEAD A_1 and HEAD A_2 as endpoints for the flow from s in N' without affecting the augmenting path. This flow of two merged with the augmenting path yields a flow of three in N'. There is a flow of two into HEAD A_1, and a flow of one into HEAD A_2. Therefore, this flow corresponds to three node disjoint paths in G -- two from s to A_1 and one from s to A_2. Our construction of $G_1 = (G/\{A_1,A_2\})_s$ does not eliminate any simple paths from s to A_1 or A_2. Therefore, the three node disjoint paths are also in G_1.

$\square$

An analogous version of Lemma IV.2.2 exists with the roles of A_1 and A_2 interchanged. Lemma IV.2.2 implies that there is no set of two cutpoints separating s from t in G such that the cutpoints are closer to s on P_1 and P_2 than A_1 and A_2. Any such cutpoints would have to separate s from A_1 and A_2, which is precluded by the augmenting paths to HEAD A_1 and HEAD A_2.

To conclude this section, we discuss the time required to find

A_1 and A_2 and to construct G_1. The $\mathcal{O}(|V_G|+|E_G|)$ network flow algorithm terminates with T, the breadth first search tree, constructed. We search T and create a table recording which nodes of G appear in T. This takes $\mathcal{O}(|V_G|)$ operations. The network flow algorithm provides us with the flow of two from s to t. We traverse the paths P_1 and P_2 in G corresponding to the network flow and mark all nodes on P_1 and P_2 in T using the table. This requires $\mathcal{O}(|E_G|)$ operations. Pointers can be used to keep track of the candidate nodes for A_1 and A_2 while traversing P_1 and P_2. Then, when the traversal is completed, A_1 and A_2 are known. Thus, we can find A_1 and A_2 in $\mathcal{O}(|V_G|+|E_G|)$ operations.

To construct $G_1 = (G/\{A_1,A_2\})_s$, we remove A_1 and A_2 from G. Graph G is stored as a set of adjacency lists.[1] Therefore, to remove A_1 and A_2, we must create a new set of adjacency lists, which requires $\mathcal{O}(|E_G|)$ operations. After A_1 and A_2 have been removed, we do a depth first search of $G-\{A_1,A_2\}$[2] starting at s. This search requires $\mathcal{O}(|V_G|+|E_G|)$ operations [Ah, pp.176-179] and will produce $(G-\{A_1,A_2\})_s$. To add nodes A_1 and A_2 and edges from A_1 and A_2 to nodes of $(G-\{A_1,A_2\})_s$ requires at most $\mathcal{O}(|V_G|)$ operations. Therefore, we can construct $(G/\{A_1,A_2\})_s$ in $\mathcal{O}(|V_G|+|E_G|)$ operations.

[1] There is an adjacency list for each node in V_G. The list for a node contains all nodes adjacent to that node. Each edge is represented twice, on the adjacency list for each endpoint.

[2] The notation G-S denotes graph G with the nodes in S, a subset of V_G, and their incident edges removed.

IV.3 The Algorithm for C_3

We determine if $C_3 \leq_{3-N} G$ by attempting to find a simple cycle containing nodes A, B, and C of G. We attack the problem by breaking G into components and looking for the paths which must exist in those components if the desired cycle exists in G. We do this by looking for certain sets of node disjoint paths in G or a component of G. If a set of node disjoint paths does not exist, we either know the cycle does not exist or find cutpoints which we use to break up G further. If a set of node disjoint paths does exist, we add this to our knowledge of G. We build up the number of sets of node disjoint paths known to be in G until we can piece together the desired cycle from the paths. The algorithm is presented in a step by step fashion.

Algorithm IV.3: Determining if $C_3 \leq_{3-N} G$

Step 1: If one of edges (A,B), (A,C), or (C,B) is an edge in G, we can find the cycle immediately, if it exists, as follows. Without loss of generality, assume $(A,B) \in E_G$. Find a path from A to B containing C. This is an instance of the fixed subgraph homeomorphism problem when the pattern graph is a tree of depth one with exactly two leaves, denoted $K_{1,2}$. The solution of this problem has been presented in Chapter III. The path from A to B containing C joined with the edge (A,B) forms a simple cycle containing A, B, and C.

Step 2: Break G into biconnected components. If A, B, and C are not in the same biconnected component, then no simple cycle containing all three exists. If A, B, and C are in the same biconnected component, consider only this component. Any simple cycle containing

A, B, and C must be in this component. Let this component be graph G. Graph G is now a biconnected graph containing A, B, and C. None of edges (A,B), (B,C), or (A,C) is an edge in G.

Step 3: Determine whether or not G contains three node disjoint paths, each with A as one endpoint and either B or C as the other endpoint. To do this, merge B and C into one node, denoted [BC]. Each edge (u, B) or (u, C) becomes edge (u,[BC]); duplicate edges are removed. Call the graph resulting from merging B and C, G(BC). Determine if A and [BC] are triconnected in G(BC) by constructing a unit vertex and edge capacity network, N(BC), from G(BC) with source [BC] and sink A. Nodes A and [BC] are triconnected in G(BC) if and only if there is a flow of three from [BC] to A in N(BC). If A and [BC] are triconnected in G(BC), then the desired node disjoint paths exist in G. We apply Algorithm IV.4, described in Section IV.4, to find a cycle in G containing A, B, and C.

If A and [BC] are not triconnected, test whether or not B and [AC] or C and [AB] are triconnected in G(AC) and G(AB), respectively. If any of these pairings is triconnected, we use Algorithm IV.4 to find a cycle containing A, B, and C. If none of the pairings is triconnected, continue to Step 4.

Step 4: At this point, we know that G is a biconnected graph which does not contain edges (A,B), (A,C), and (B,C). We also know that A and [BC] are not triconnected in G(BC).

Since G is biconnected, A and [BC] must be biconnected in G(BC). Therefore, when we used the network flow algorithm on N(BC) with [BC] as the source, the network flow algorithm must have found a flow of two from [BC] to A in N(BC). We can apply the procedure described in Section IV.2 to find a cutset $\{A_1,A_2\}$ separating [BC] from A in G(BC). This cutset separates A from B and C in G. If $\{A_1,A_2\}$ also separates B from C in G, then a simple cycle in G containing A, B, and C does not exist. Each portion of the cycle -- from A to B, from B to C, and from C to A -- would contain one of nodes A_1 or A_2. To determine if $\{A_1,A_2\}$ separates B from C, we need only construct $(G-\{A_1,A_2\})_B$ and ask if C is a node in $(G-\{A_1,A_2\})_B$. If not, $\{A_1,A_2\}$ separates B from C, and we can halt. If $(G-\{A_1,A_2\})_B$ contains C, then we continue to Step 5.

Step 5: We now have biconnected graph G containing cutpoints A_1 and A_2 separating A from B and C. We also know:

(1) (A,B), (B,C), $(C,A) \notin E_G$.

(2) $(G-\{A_1,A_2\})_B = (G-\{A_1,A_2\})_C$.

(3) There is a path in G from A_1 to A_2 containing A which contains no nodes in $(G-\{A_1,A_2\})_B$.

(4) Lemma IV.2.2 applies to $(G(BC)/\{A_1,A_2\})_{[BC]}$.

Fact 1 follows directly from Step 1. Fact 2 is a direct consequence of the definition of $(G-\{A_1,A_2\})_C$, and the fact that C is a node in $(G-\{A_1,A_2\})_B$. Fact 3 is guaranteed by the flow of two from [BC] to A in N(BC). The path defined by the flow segments from A_1 to A and from A_2 to A cannot contain any nodes in $(G-\{A_1,A_2\})_B$, since A_1 and A_2 separate A from B and C. Fact 4 follows from our application of the network flow algorithm with [BC] as the source.

Given fact 3, we can complete our cycle in G containing A, B, and C, if and only if we can find a path in $(G/\{A_1,A_2\})_B$ from A_1 to A_2 containing B and C (in either order). This follows from the fact that $(G/\{A_1,A_2\})_B = (G/\{A_1,A_2\})_C$ and that $(G/\{A_1,A_2\})_B$ (respectively $(G/\{A_1,A_2\})_C$) contains all simple paths in G from A_1 or A_2 to B (respectively C). Let $G_1 = (G/\{A_1,A_2\})_B$. We will now limit our search to the desired path in G_1. Note that neither A_1 nor A_2 is isolated in G_1, since there are two node disjoint paths--from A_1 to [BC] and from A_2 to [BC]--in G(BC). Therefore, G_1 is connected.

Consider $(G-\{A_1,A_2\})_B$. Are B and C biconnected in $(G-\{A_1,A_2\})_B$? If so, continue to Step 6. Otherwise, find any cutpoint x (called an articulation point) which separates B from C in $(G-\{A_1,A_2\})_B$. In G_1, $\{A_1,A_2,x\}$ is a cutset separating B and C, since $(G-\{A_1,A_2\})_B$ is simply G_1 with A_1 and A_2 (and their incident edges) removed. Let $S_B = (G_1/\{A_1,A_2,x\})_B$ and $S_C = (G_1/\{A_1,A_2,x\})_C$. Note that $\{A_1,x\}$, $\{A_2,x\}$ or $\{x\}$ may be a cutset of G_1. Therefore, A_1 or A_2 may be isolated in S_B or S_C . (See Figure IV.3-1.) Any path in G_1 from A_1 to A_2 containing B and C must be of the form: path 1 in S_B from A_1 (or A_2) to x, containing B but not containing A_2 (or A_1), followed by path 2 in S_C from x to A_2 (or A_1), containing C but not containing A_1 (or A_2). (See Figure IV.3-2.) The existence of these paths can be determined by solving the fixed subgraph homeomorphism problems: $S_B-\{A_1\} \leq_N K_{1,2}$ with $\nu(r)=B$, $\nu(l_1)=A_2$, $\nu(l_2)=x$ and

$S_C-\{A_2\} \leq_N K_{1,2}$ with $\nu(r)=C$, $\nu(l_1)=A_1$, $\nu(l_2)=x$

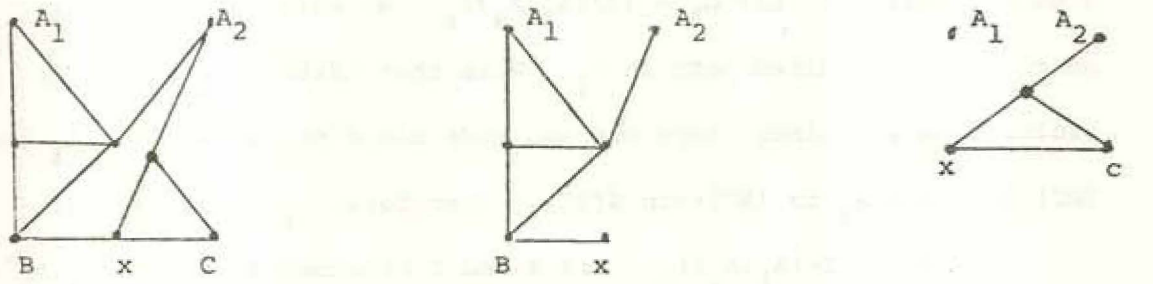
or: $S_B-\{A_2\} \leq_N K_{1,2}$ with $\nu(r)=B$, $\nu(l_1)=A_1$, $\nu(l_2)=x$ and

$S_C-\{A_1\} \leq_N K_{1,2}$ with $\nu(r)= C$, $\nu(l_1)=A_2$, $\nu(l_2)=x$.

Here, $K_{1,2}$ is the tree of depth one with root r and exactly two leaves, l_1 and l_2. We have presented an algorithm for this problem in Chapter III.

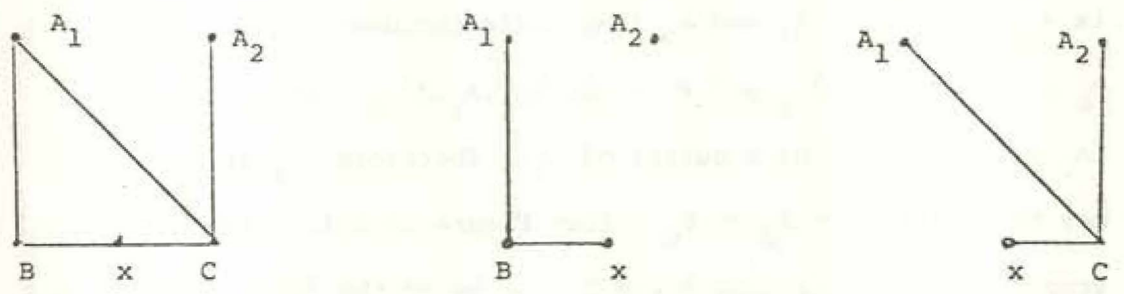
Figure IV.3-1 Examples of isolated A_1 and A_2.

G_1: $(G_1/\{x,A_1,A_2\})_B$: $(G_1/\{x,A_1,A_2\})_C$:



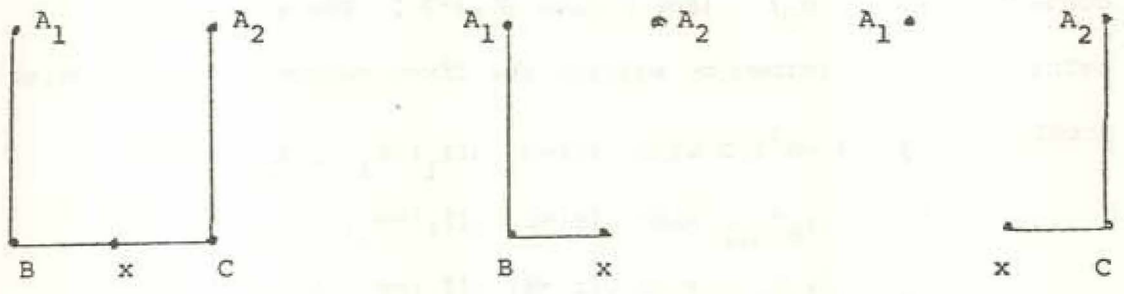
$\{A_2,x\}$ is a cutset.

G_1: $(G_1/\{x,A_1,A_2\})_B$ $(G_1/\{x,A_1,A_2\})_C$:



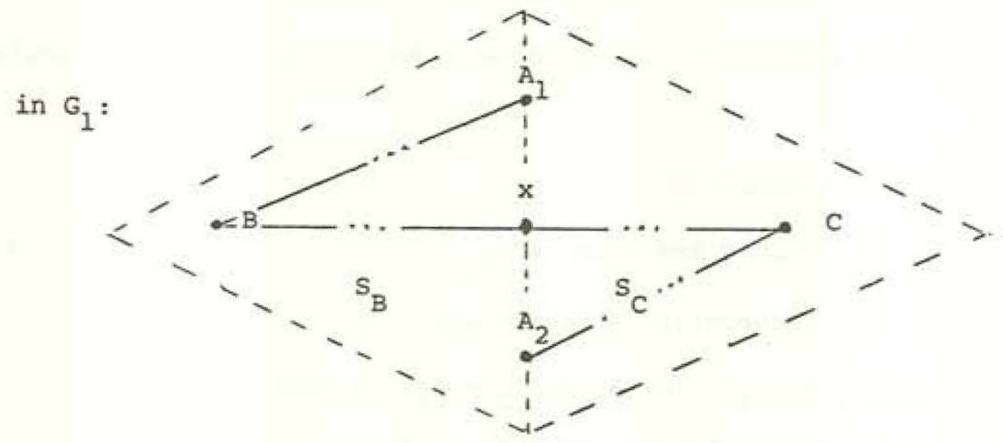
$\{A_1,x\}$ is a cutset.

G_1: $(G_1/\{x,A_1,A_2\})_B$: $(G_1\{x,A_1,A_2\})_C$:

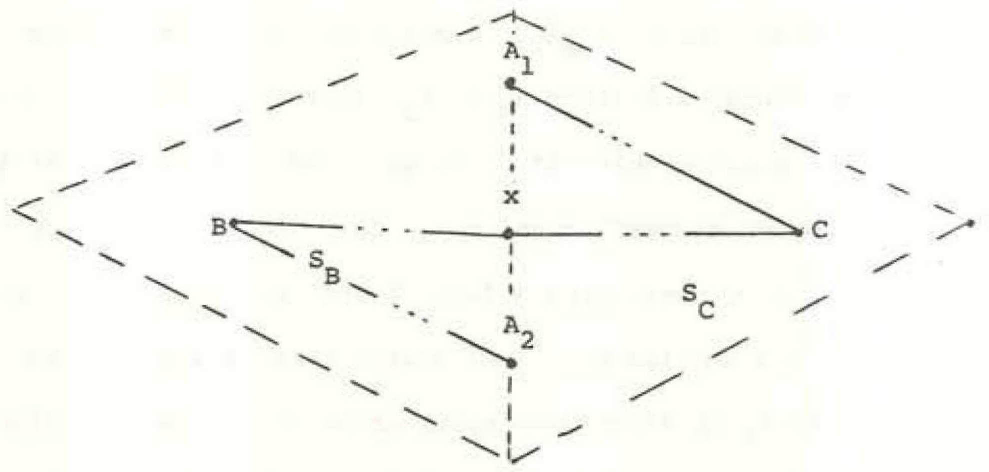


$\{x\}$ is a cutset.

Figure IV.3-2: Paths in S_B and S_C.



or:



Step 6: At this point we would like to find a path from A_1 to A_2 in G_1 containing B and C (in either order). We know the following facts:

(1) G is biconnected.

(2) (A,B), (A,C), and (C,A) $\notin E_G$.

(3) A_1 and A_2 separate A from B and C in G.

(4) $(G/\{A_1,A_2\})_B = (G/\{A_1,A_2\})_C = G_1$, a connected graph.

(5) Lemma IV.2.2 applies to $(G(BC)/\{A_1,A_2\})_{[BC]}$.

(6) There are two node disjoint paths in G_1 from B to C, neither of which contains A_1 or A_2.

Consider $(G(BC)/\{A_1,A_2\})_{[BC]}$. Since there is a path from B to C in G_1 which contains neither A_1 or A_2, $(G(BC)/\{A_1,A_2\})_{[BC]} = G_1(BC)$. That is, the same graph results if we merge B and C in G and then take the $\{A_1,A_2\}$-component of G(BC) containing [BC], or if we take the $\{A_1,A_2\}$-component of G containing B (and C) and then merge nodes B and C. Lemma IV.2.2 applied to $G_1(BC)$ states that if there is an augmenting path to A_1 in N(BC) upon termination of the network flow algorithm, then there are three node disjoint paths in $G_1(BC)$, two from A_1 to [BC], and one from A_2 to [BC]. The analogous result holds if there is an augmenting path to A_2 in N(BC) upon termination of the network flow algorithm.

Determine if any two of edges (A_1,B), (A_2,B), (A_1,C) and (A_2,C) are in G_1. If so, then, without loss of generality, there are three distinct cases.

Case 1: Edges (A_1,B) and (A_1,C) are in G_1. The desired path from A_1 to A_2 containing B and C (in either order) is in G_1 if and only if there is a path in $G_1-\{A_1\}$ from A_2 to B containing C, or a path in $G_1-\{A_1\}$ from A_2 to C containing B. We can determine if either of these paths exists by solving the fixed subgraph homeomorphism problem for pattern graph $K_{1,2}$. If one of the paths exists, we add to it the appropriate edge of (A_1,B) or (A_1,C) to get the desired path in G_1.

Case 2: Edges (A_1,B) and (A_2,B) are in G_1. The desired path in G_1 from A_1 to A_2 exists if and only if a path from B to A_2 containing C is in $G_1-\{A_1\}$ or a path from B to A_1 containing C is in $G_1-\{A_2\}$. We determine if these paths exist as for case 1.

Case 3: Edges (A_1,B) and (A_2,C) are in G_1. By fact 6 above, there is a path in G_1 from B to C containing neither A_1 nor A_2. Adding edges (A_1,B) and (A_2,C) to the ends of this path gives us the desired path in G_1.

If no two of edges (A_1,B), (A_2,B), (A_1,C), and (A_2,C) are in G_1, continue to Step 7.

Step 7: Determine if there are augmenting paths in N(BC) to A_1 and A_2 upon termination of the network flow algorithm. At least one of A_1 and A_2 must have an augmenting path to it. Otherwise, the definition of A_1 and A_2 implies that both $(A_1,[BC])$ and $(A_2,[BC])$ are edges in $G_1(BC)$, contradicting our findings in Step 6. If both A_1 and A_2 have augmenting paths to them, applying Lemma IV.2.2, we know that there are two sets of three node disjoint paths in $G_1(BC)$. In the first set (respectively second set), two paths go from A_1 (respectively A_2) to [BC], and one path goes from A_2 (respectively A_1) to [BC]. We continue to Step 8.

Suppose there is no augmenting path to one of A_1 or A_2. Without loss of generality, assume this node is A_1. This implies that there is no path from A_1 to [BC] in $G_1(BC)-\{A_2\}$ other than edge $(A_1,[BC])$. Otherwise, this path would define an augmenting path to A_1 in N(BC). To see this, note that any path from [BC] to A_1 which uses an interior node of the path defined by the flow from [BC] to A_2 in N(BC) can be combined with the augmenting path to A_2 to form an augmenting path to A_1. Any path which does not use such an interior node automatically defines an augmenting path to A_1. We conclude that a path in G_1 from A_1 to A_2 containing B and C exists if and only if edge $(A_1,[BC])$ corresponds to edge (A_1,B) (respectively (A_1,C)) in G_1, and a path from B (respectively C) to A_2 containing C (respectively B) exists in $G_1-\{A_1\}$. By Step 6, only one of (A_1, B) and (A_1,C) is in G_1. We determine if the path necessary to complete our desired path exists by using the algorithm for the fixed subgraph homeomorphism problem with pattern graph $K_{1,2}$.

Step 8: At this point we know the following facts:

(1) G is biconnected.

(2) (A,B), (B,C), $(C,A) \notin E_G$.

(3) A_1 and A_2 separate A from B and C in G.

(4) $(G/\{A_1,A_2\})_B = (G/\{A_1,A_2\})_C = G_1$, a connected graph.

(5) There is a path in G from A_1 to A_2 containing A which lies outside G_1.

(6) There are two node disjoint paths from B to C in G_1, neither of which contains A_1 or A_2.

(7) There is a set of three node disjoint paths in $G_1(BC)$-- two paths from A_1 to [BC] and one path from A_2 to [BC].

(8) There is a set of three node disjoint paths in $G_1(BC)$--

two paths from A_2 to [BC] and one path from A_1 to [BC].

(9) No two of edges (A_1,B), (A_2,B), (A_1,C), (A_2,C) are in G_1.

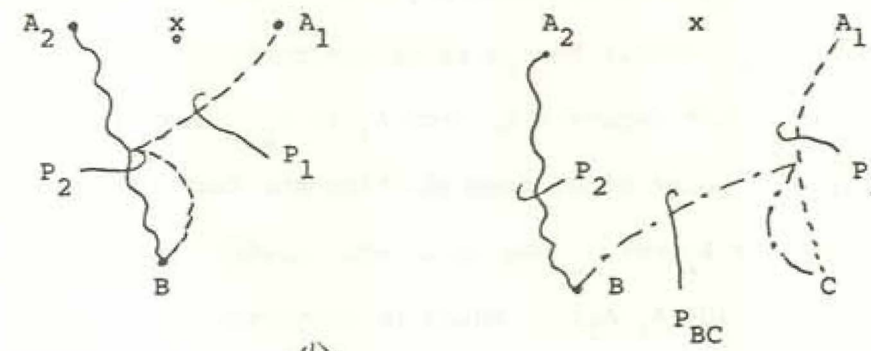
Determine if G_1 is biconnected. If G_1 is biconnected, continue to Step 9. If G_1 is not biconnected, then there is an articulation point, x, in G_1. Recall that G is biconnected. Given this, articulation point x must separate A_1 from A_2 in G_1. Otherwise, x is also an articulation point of G, contradicting the fact that G is biconnected. Neither A_1 nor A_2 can be an articulation point since G_1 was formed from $(G-\{A_1,A_2\})_B$, which is connected, and neither A_1 nor A_2 is isolated.

Suppose x is not equal to B or C. By fact 8 above, there are two node disjoint path from A_2 to [BC] in G_1(BC). Denote the path which does not contain x as P_2. Without loss of generality, assume P_2 goes to B in G_1. By fact 7 above, there are two node disjoint paths from A_1 to [BC] in G_1(BC). Denote the path which does not contain x as P_1. If P_1 goes to B, then P_2 joined with P_1, with any subcycles removed, is a path from A_2 to A_1 which does not contain x. This contradicts the requirement that articulation point x separate A_1 from A_2. Therefore, assume P_1 goes to C. By fact 6 above, there is a path in G_1 from B to C which contains none of A_1, A_2, and x. Call this path P_{BC}. The path composed of P_2, P_{BC}, and P_1 with subcycles removed is a path from A_2 to A_1 which does not contain x. Again we have a contradiction. We conclude that x must equal one of B and C. Figure IV.3-3a illustrates the above argument.

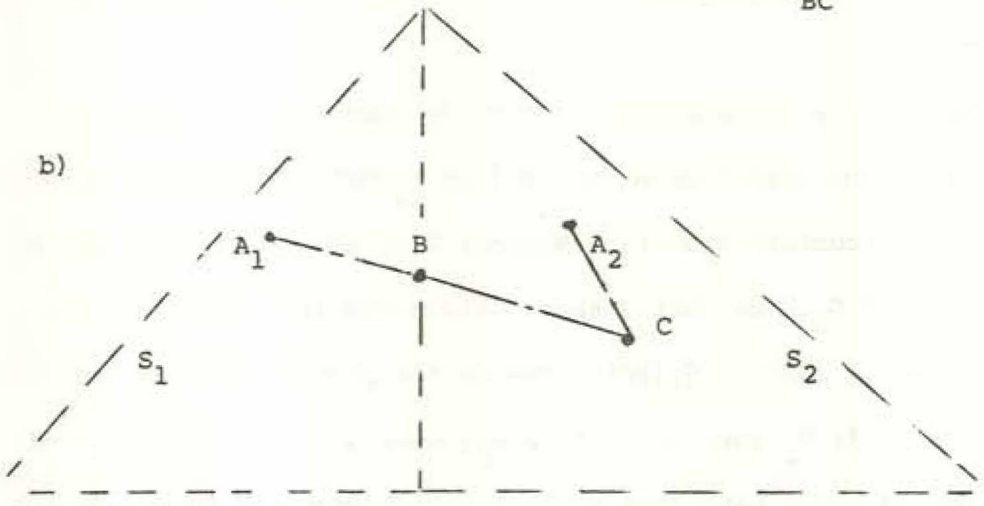
Without loss of generality, assume B is an articulation point. Node B cannot separate all three of A_1, A_2, and C from each other,

Figure IV.3-3: Illustration for Step 8.

a) P_1 goes to B: P_1 goes to C:



b)



since there must be a path from C to A_2 not containing B in G, by the biconnectivity of G. This path is either completely in G_1 or has an initial portion from C to A_1 in G_1. Without loss of generality, assume B does not separate C from A_2. Let $S_1=(G_1/\{B\})_{A_1}$ and $S_2=(G_1/\{B\})_{A_2}$. The desired path in G_1 from A_1 to A_2 containing B and C exists if and only if there is a path in S_2 from B to A_2 containing C. (The path in S_1 from A_1 to B is guaranteed by the definition of S_1.) We determine if this path in S_2 exists by solving the subgraph homeomorphism problem for pattern graph $K_{1,2}$ (Figure IV.3-3b).

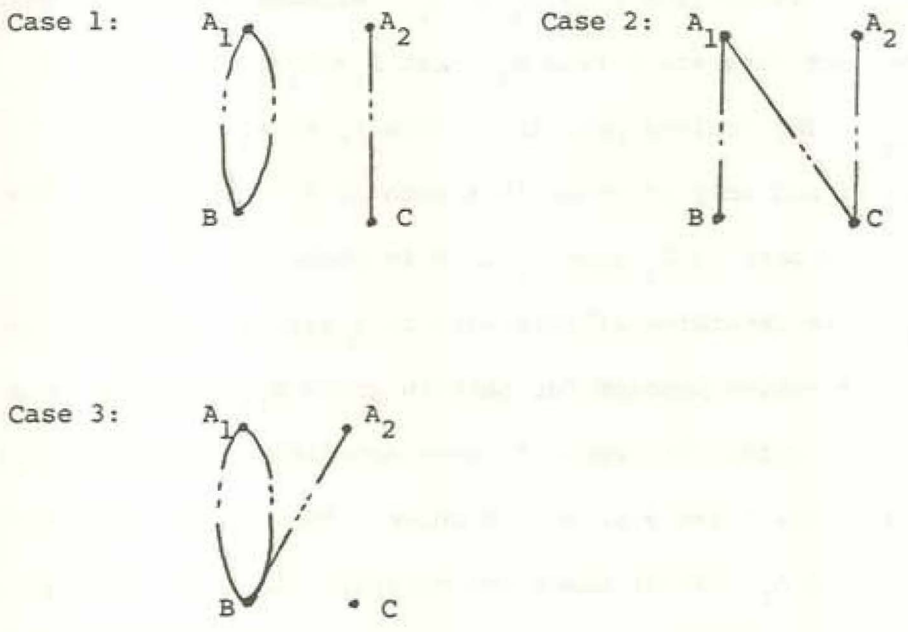
<u>Step 9</u>: Consider the sets of three node disjoint paths in G_1(BC) guaranteed by facts 7 and 8 of Step 8 above. These paths correspond to three paths in G_1. Three cases are possible, as shown in Figure IV.3-4a. Case 3 is reduced to case 1 or case 2 by finding a path from C to A_2 in G_1 which does not contain B. This path is guaranteed to exist by the biconnectivity of G_1. Some initial portion of this path from C is node disjoint from the three paths in case 3 except at its endpoint. This path is pieced together with the appropriate path of case 3 to give case 1 or case 2. (See Figure IV.3-4b)

In case 1, we use Algorithm IV.4 to find a path in G_1 from A_1 to A_2 containing B and C. In case 2, we continue to Step 10.

<u>Step 10</u>: Determine whether or not $[A_1B]$ and $[A_2C]$ are triconnected or $[A_2B]$ and $[A_1C]$ are triconnected in the graphs obtained by appropriately modifying G_1. If neither pair of merged nodes is triconnected, then neither a path of the form $\langle A_1,\ldots,B,\ldots,C,\ldots,A_2\rangle$ nor a path of the form $\langle A_2,\ldots,B,\ldots,C,\ldots,A_1\rangle$ exists in G_1, since no two of edges (A_1,B), (A_1,C), (A_2,B), (A_2,C), and (B,C) exist in G_1.

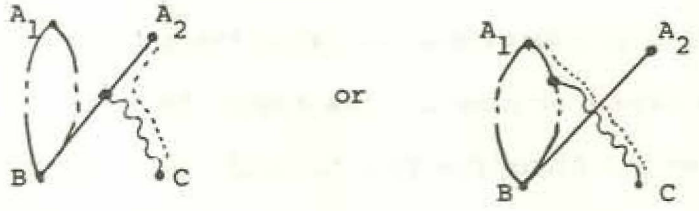
Figure IV.3-4 Cases for Step 9

a)



In all cases, A_1 and A_2 are interchangeable, and B and C are interchangeable.

b)



Reduces to case 1 or case 2 under -------

This is illustrated in Figure IV.3-5. If either pair of merged nodes is triconnected, continue to Step 11.

Step 11: Without loss of generality, assume the three node disjoint paths found in Step 10 are from $[A_1B]$ to $[A_2C]$. We know the following facts:

(1) G is biconnected.

(2) (A,B), (B,C), $(C,A) \notin E_G$.

(3) A_1 and A_2 separate A from B and C in G.

(4) $(G/\{A_1,A_2\})_B = (G/\{A_1,A_2\})_C = G_1$, a biconnected graph.

(5) There is a path in G from A_1 to A_2 containing A which lies outside G_1.

(6) There are two node disjoint paths from B to C in G_1, neither of which contains A_1 or A_2.

(7) There is a set of three node disjoint paths in G_1, with one path from A_1 to B, one path from A_1 to C, and one path from A_2 to one of B or C.

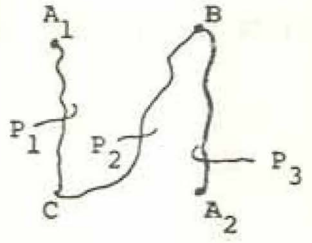
(8) There is a set of three node disjoint paths in G_1, with one path from A_2 to B, one path from A_2 to C, and one path from A_1 to one of B or C.

(9) No two of edges (A_1,B), (A_1, C), (A_2,B), or (A_2,C) are in G_1.

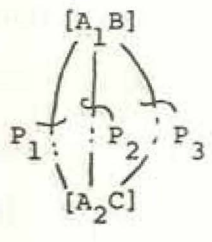
Consider the two disjoint paths from B to C which are guaranteed by fact 6. Since $[A_1B]$ and $[A_2C]$ are triconnected, there exists at least one augmenting path in a network constructed from G_1 with flow corresponding to the two node disjoint paths between B and C. This augmenting path results in three node disjoint paths, each from A_1 or B to A_2 or C in G_1. Note 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

Figure IV.3-5 Illustration for Step 10

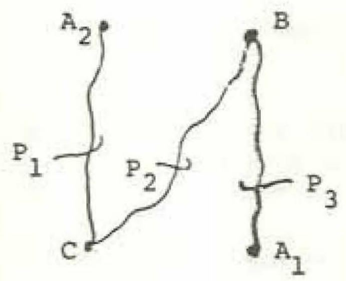
If:

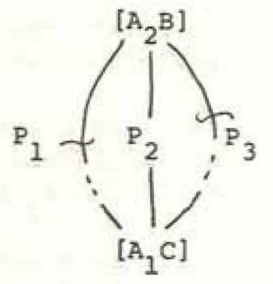


then:



or: exist with at least two of P_1, P_2, and P_3 of length ≥ 2 or:





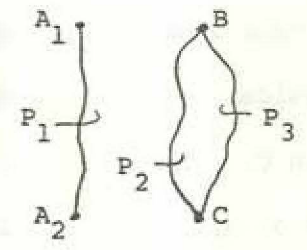
endpoints of flow. The possible configurations of these paths are shown in Figure IV.3-6. Cases 1, 2, and 3 are those which must be examined.

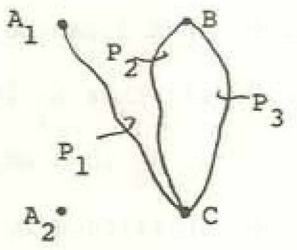
Cases 2 and 3 can be reduced to either case 1 or case 5 as follows. Recall that G_1 is biconnected. Consider case 2. Find a path in G_1 from A_2 to B which does not contain C. The existence of this path is guaranteed by the biconnectivity of G_1. Some initial portion of this path, say $\langle A_2,\ldots,v\rangle$, is node disjoint from the paths P_1, P_2, and P_3 shown in case 2 except at endpoint v. Note that v may equal A_1 or B, but not C. If v is an interior node of path P_1 or equal to A_1, we have reduced case 2 to case 1 as shown in Figure IV.3-7a. If v is an interior node on one of paths P_2 or P_3 or equal to B, we have reduced case 2 to case 5 as shown in Figure IV.3-7b. Case 3 is handled similarly.

We now must deal with case 1. By fact 7 above, we have three node disjoint paths of the form : path Q_1 from A_1 to B; path Q_2 from A_1 to C; path Q_3 from A_2 to C or from A_2 to B. Without loss of generality, we assume that Q_3 goes from A_2 to C. We will use these three node disjoint paths in conjunction with the three node disjoint paths of case 1: P_1 from A_1 to A_2 ; P_2 from B to C; P_3 from B to C. (See Figure IV.3-8.)

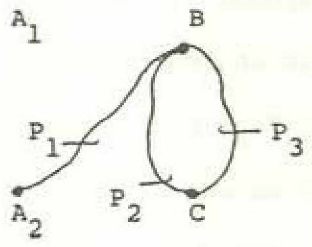
Let: p_i be the vertex closest to A_1 on Q_i which is also on P_2 or P_3, $i = 1$ or 2. Node p_1 may equal B; node p_2 may equal C, but both p_1 and p_2 are distinct and different from A_1.

Figure IV.3-6 Configurations for Step 11.

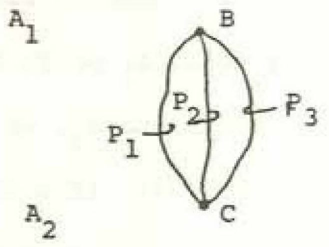
1) 

2) 

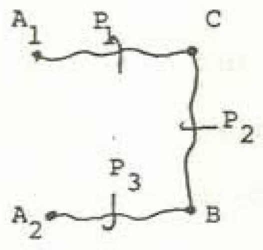
Reduces to (1) or (5)

3) 

Reduces to (1) or (5)

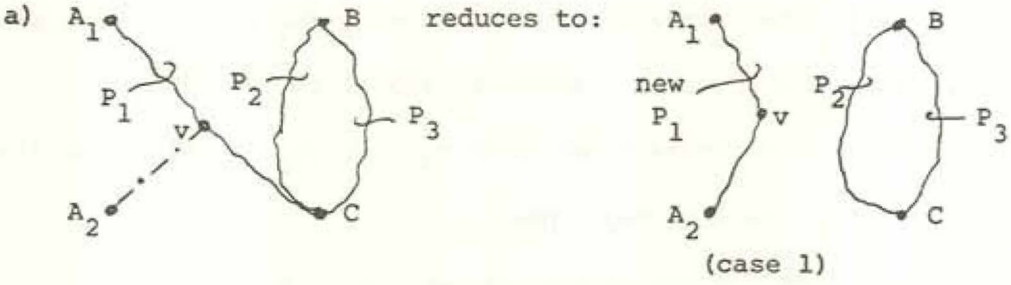
4) 

Eliminated at Step 3

5) 

Desired path

Figure IV.3-7: Reduction of Cases 2 and 3 to 1 or 5.



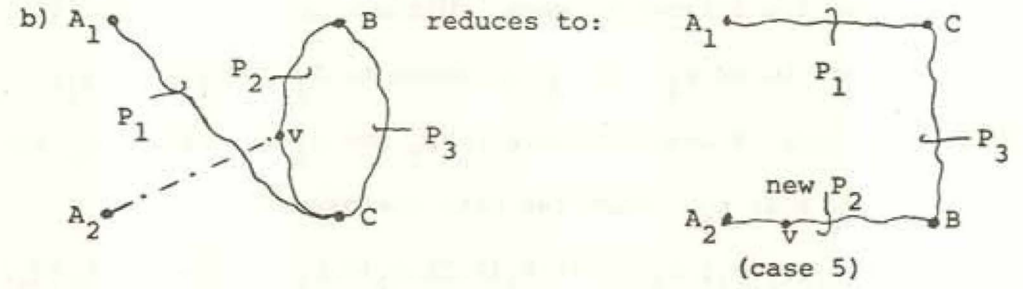
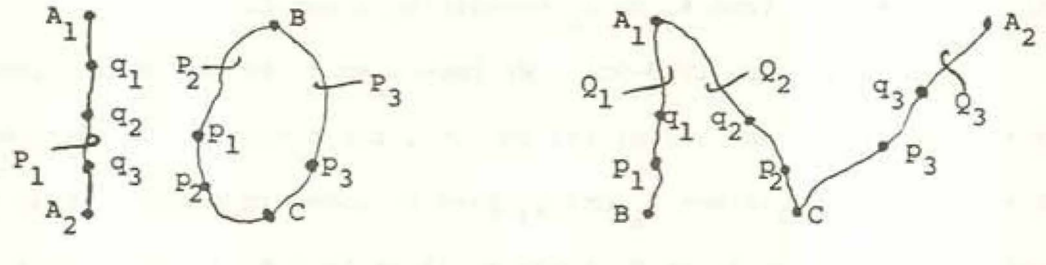


Figure IV.3-8: Paths P_1,P_2,P_3 and Q_1,Q_2,Q_3.



Let: p_3 be the vertex closest to A_2 on Q_3 which is also on P_2 or P_3 . Node p_3 may equal C.

Let: q_i be the vertex closest to p_i on $Q_i[A_1,p_i]$ which is also on P_1, i = 1 or 2. Nodes q_1 and q_2 may equal A_1.

Let: q_3 be the vertex closest to p_3 on $Q_3[A_2,p_3]$ which is also on P_1. Node q_3 may equal A_2.

Figure IV.3-8 shows one configuration for the p_i's and q_i's.

Noting that two of p_1,p_2, and p_3 must both be on P_2 or both be on P_3, we have the following cases. Without loss of generality, let p_i and p_j both be on P_2; let q_i be closer to A_1 on P_1 than q_j.

Case a (Figure IV.3-9a):We have $p_i \neq p_j$ and $q_i \neq q_j$. Also, p_i is closer to B on P_2 than p_j. Then the path composed of:

$$P_1[A_1,q_i],Q_i[q_i,p_i],P_2[p_i,B],P_3[B,C],P_2[C,p_j],Q_j[p_j,q_j],P_1[q_j,A_2]$$

is a simple path from A_1 to A_2 containing B and C.

Case b (Figure IV.3-9b): We have $p_i \neq p_j$ and $q_i \neq q_j$, but p_j is closer to B on P_2 than p_i. The path:

$$P_1[A_1,q_i],Q_i[q_i,p_i],P_2[p_i,C],P_3[C,B],P_2[B,p_j],Q_j[p_j,q_j],P_1[q_j,A_2]$$

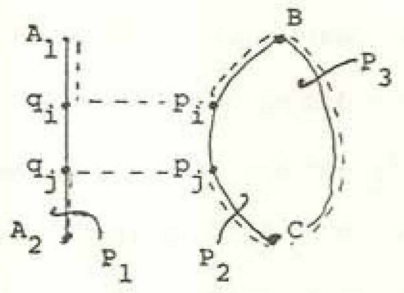
is a simple path from A_1 to A_2 containing B and C.

Case c (Figure IV.3-9c): We have $p_i=p_j$. By our definition, $p_i=p_j$ only if i=3 and j=2 or i=2 and j=3, and $p_2=p_3=C$. In this case, $p_1 \neq p_3$ and $q_1 \neq q_3$, since Q_1 and Q_3 have no nodes in common. Also p_1 and p_3 are both on P_2 or P_3 since p_3 is on both P_2 and P_3. Thus, case c reduces to case a or case b.

Case d (Figure IV.3-9d): We have $q_i=q_j$. By our definition, $q_i=q_j$ only if i=1 and j=2 or i=2 and j=1, and $q_1=q_2=A_1$. We know $p_1 \neq p_2, p_1 \neq p_3$, $p_3 \neq B$, and $q_3 \neq A_1$, by construction. Therefore, p_3 does

Figure IV.3-9: Configurations of p_i's and q_i's.

a)



$q_i \neq q_j$

$p_i \neq p_j$

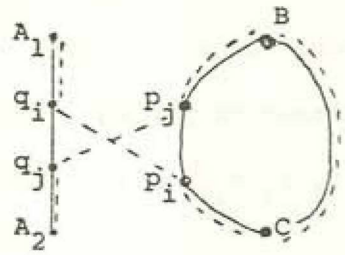
p_i may equal B

p_j may equal C

q_i may equal A_1

q_j may equal A_2

b)



$q_i \neq q_j$

$p_i \neq p_j$

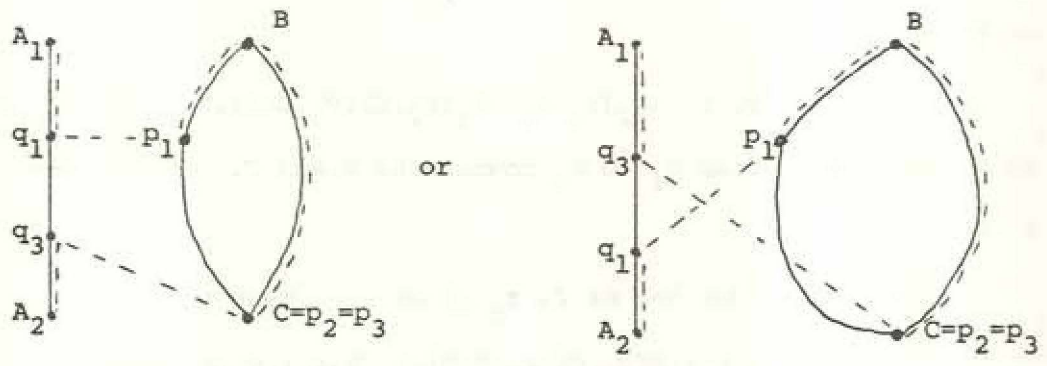
p_i may equal C

p_j may equal B

q_i may equal A_1

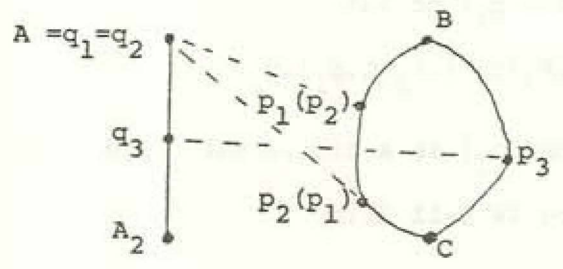
q_j may equal A_2

c)



q_1 may equal A_1; q_3 may equal A_2; p_1 may equal B.

d)



$p_3 \neq$ B or C

$p_1 \neq$ B

$p_2 \neq$ C

q_3 may equal A_2

----- indicates portions of P_i and Q_i paths used.

not equal C, since then p_1 and p_3 would both be on P_2 and case a would apply. Similarly, neither p_1=B nor p_2=C since then one of p_1 or p_2 and p_3 would be on P_3 and case a or case b would apply.

By fact 8 above, there are two node disjoint paths from A_2 to [BC] in $G_1[BC]-\{A_1\}$. Call these paths R_1 and R_2. Let r_i be the closest vertex to A_2 on R_i which is also on $P_2, P_3, Q_1[A_1,p_1]$, or $Q_2[A_1,p_2]$; let s_i be the closest vertex to r_i on $R_i[A_2,r_i]$ which is also on P_1, i=1 or 2. (See Figure IV.3-10.) Nodes s_i may equal A_2; nodes r_i may equal B or C. The following cases can occur. Without loss of generality, assume p_1 is closer to B than p_2 on P_2.

Case (i): For t = 1 or 2, r_t is on $Q_1[A_1,p_1]$ or r_t is on $Q_2[A_1,p_2]$. Then, path:

$P_1[A_2,s_t], R_t[s_t,r_t], Q_1[r_t,p_1], P_2[p_1,B], P_3[B,C], P_2[C,p_2], Q_2[p_2,A_1]$

or path:

$P_1[A_2,s_t], R_t[s_t,r_t], Q_2[r_t,p_2], P_2[p_2,C], P_3[C,B], P_2[B,p_1], Q_1[p_1,A_1]$

is a simple path from A_1 to A_2 containing B and C. (See Figure IV.3-11i.)

Case (ii): For t=1 or 2, r_t is on P_2 . Then the path

$P_1[A_2,s_t], R_t[s_t,r_t], P_2[r_t,C], P_3[C,B], P_2[B,p_1], Q_1[p_1,A_1]$

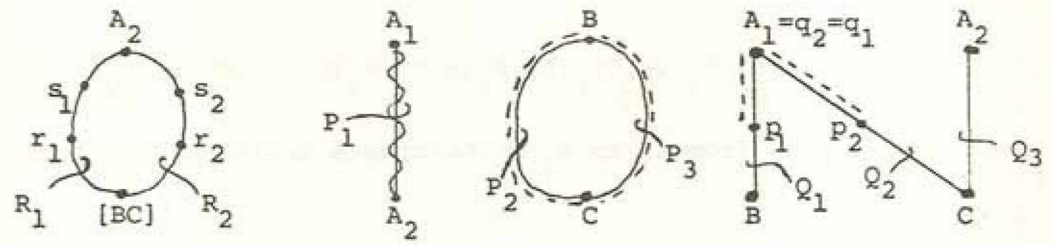
(if r_t is closer to C on P_2 than p_1) or path:

$P_1[A_2,s_t] R_t[s_t,r_t], P_2[r_t,B] P_3[B,C], P_2[C,p_1], Q_1[p_1,A_1]$

(if r_t is closer to B on P_2 than p_1) is a simple path from A_1 to A_2 containing B and C. (See Figure IV.3-11 ii.)

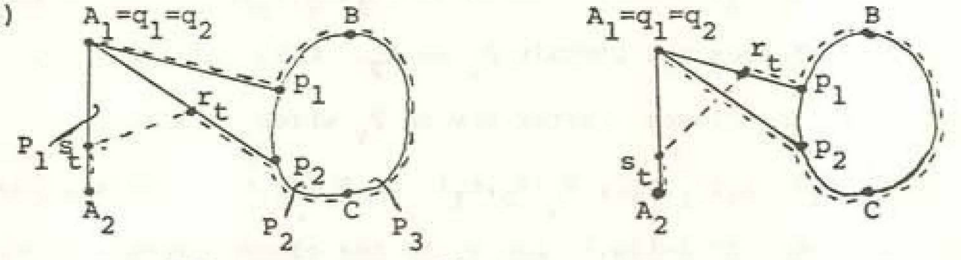
Case (iii): Both r_1 and r_2 are on P_3 and only P_3 (i.e. r_1, and r_2 are not equal to B or C), and $s_1 \neq s_2$. Without loss of generality, assume r_1 is closer to B than r_2 on P_3. Then path:

Figure IV.3-10: Definition of r_1, r_2, s_1, and s_2.

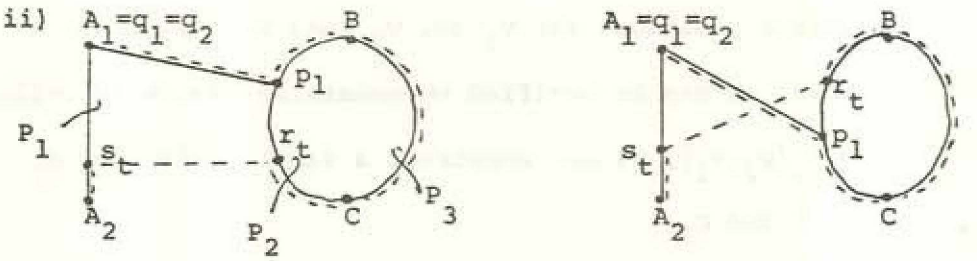


- - - - indicates may contain r_1 or r_2.
〰 indicates may contain s_1 or s_2.

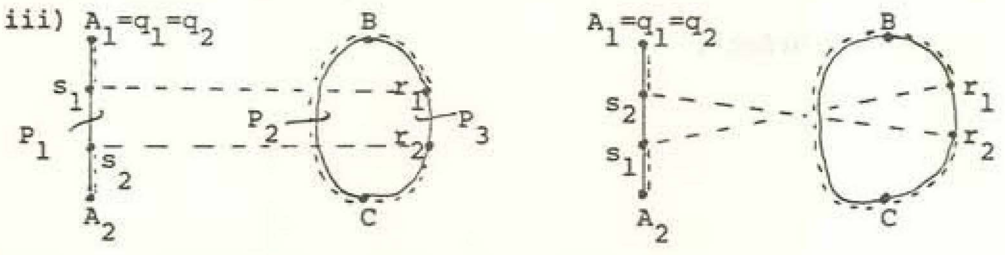
Figure IV.3-11: Possible configurations of r_1, r_2, s_1, and s_2.
------ indicates path used.

i)


s_t may equal A_2; r_t may equal p_1 or p_2.

ii)


s_t may equal A_2; r_t may equal B or C.

iii)


$s_1 \neq s_2$; s_2 may equal A_2; $r_1 \neq$ B or C; $r_2 \neq$ B or C

$P_1[A_2,s_2],R_2[s_2,r_2],P_3[r_2,C],P_2[C,B],P_3[B,r_1],R_1[r_1,s_1],P_1[s_1,A_1]$

or path:

$P_1[A_2,s_1],R_1[s_1,r_1],P_3[r_1,B],P_2[B,C],P_3[C,r_2],R_2[r_2,s_2],P_1[s_2,A_1]$

is a simple path from A_1 to A_2 containing B and C. (See Figure IV.3-11 iii.)

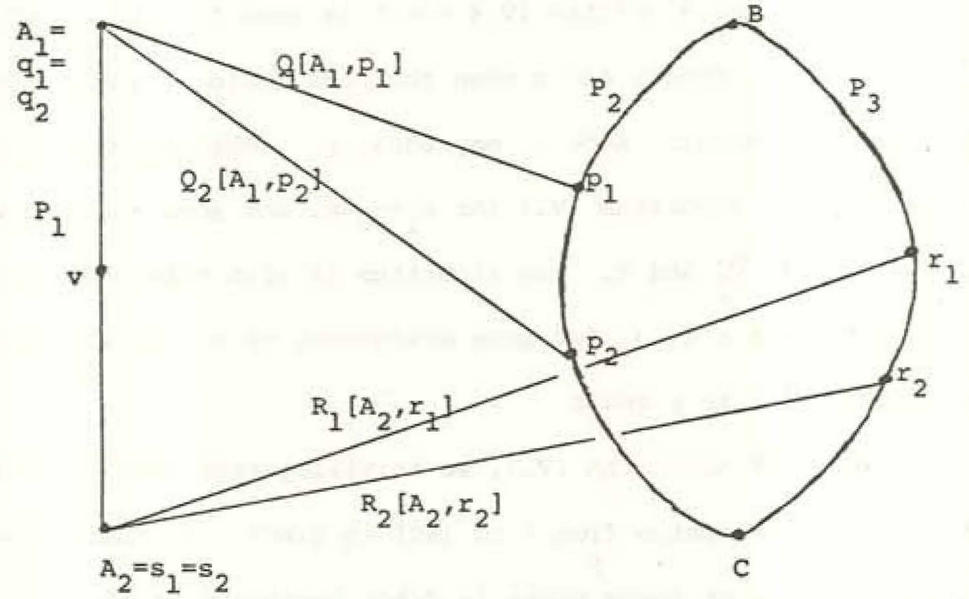
Case (iv): Both r_1 and r_2 are on P_3 and only P_3, and $s_1=s_2$. Then we know $r_1 \neq r_2$ (since $r_1=r_2$ implies $r_1=r_2=B$ or $r_1=r_2=C$), and $s_1=s_2=A_2$. By definition of $(G/\{A_1,A_2\})_B = G_1$, edge (A_1,A_2) is not in G_1. Therefore path P_1 is of length at least two. Choose any interior node v on P_1. Since v is in $(G-\{A_1,A_2\})_B$, there is a path from v to B which does not contain A_1 or A_2. Call this path P_v.

Let v_1 be the closest vertex to v on P_v which is also on P_2, P_3, $Q_1[A_1,p_1]$, $Q_2[A_1,p_2]$, $R_1[A_2,r_1]$, or $R_2[A_2,r_2]$. (All paths are shown in Figure IV.3-12a.) Let v_2 be the closest vertex to v_1 on $P_v[v,v_1]$ which is also on P_1. Note that v_2 does not equal A_1 or A_2. All possible positions for v_1 and v_2 lead to one of the cases solved above, as can be verified by examining Figure IV.3-12b. Therefore using $P_v[v_2,v_1]$, we can construct a simple path from A_1 to A_2 containing B and C.

This completes Algorithm IV.3. All cases have been resolved. If Step 11 is reached, a path is found from A_1 to A_2 containing B and C (in either order).

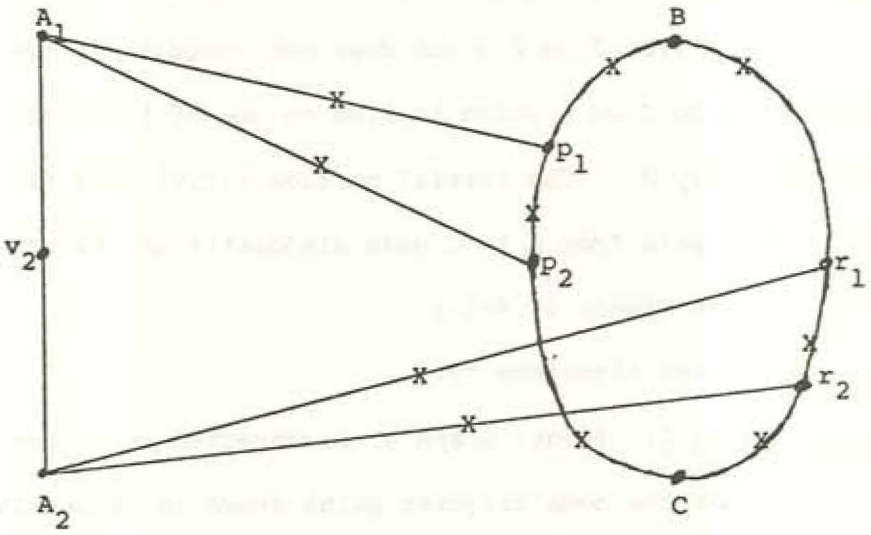
Figure IV.3-12: Final stage of Step 11--case iv.

a) All node disjoint paths found before last stage of Step 11:



p_1 and p_2 not equal to B or C; r_1 and r_2 not equal to B or C.

b) Possible positions for v_1.



X indicates a possible position for v_1. In addition, v_1 may equal B, C, p_1, p_2, r_1, or r_2.

IV.4 Algorithm IV.4

We now present Algorithm IV.4 which is used to find a path from x_1 to x_2 containing y and z when the node disjoint paths shown in Figure IV.4-1 exist. Node x_1 may equal x_2. This algorithm is used in Step 3 of Algorithm IV.3 for $x_1=x_2=x$, and some assignment of A, B, and C to x, y, and z. The algorithm is also used in Step 9, case 1, of Algorithm IV.3 for some assignment of A_1 and A_2 to x_1 and x_2, and B and C to y and z.

In Step 3 of Algorithm IV.3, we initially know that there are three node disjoint paths from A to [BC] in G(BC). We must show that the existence of these paths in G(BC) implies that the paths shown in Figure IV.4-1 exist in G. Suppose the three paths in G(BC) correspond to three paths from A to B (or C) in G. Without loss of generality, assume the three paths go to B. Since G is biconnected, there is a path, P, from C to A which does not contain B. Let v be the closest node to C on P which is also on one of the three node disjoint paths, say P'. The initial portion P[C,v] plus the subpath P'[v,A] yields a path from A to C node disjoint from the other paths from A to B. (See Figure IV.4-2.)

We now present Algorithm IV.4.

Algorithm IV.4: Input: Graph G, biconnected and nodes x_1, x_2, y, and z such that the node disjoint paths shown in Figure IV.4-1 exist. It may be that $x_1=x_2=x$. If x_1 is distinct from x_2, we are also guaranteed that a simple cycle containing y and z exists which does not contain x_1 or x_2.

Figure IV.4-1: Node disjoint paths for Algorithm IV.4.

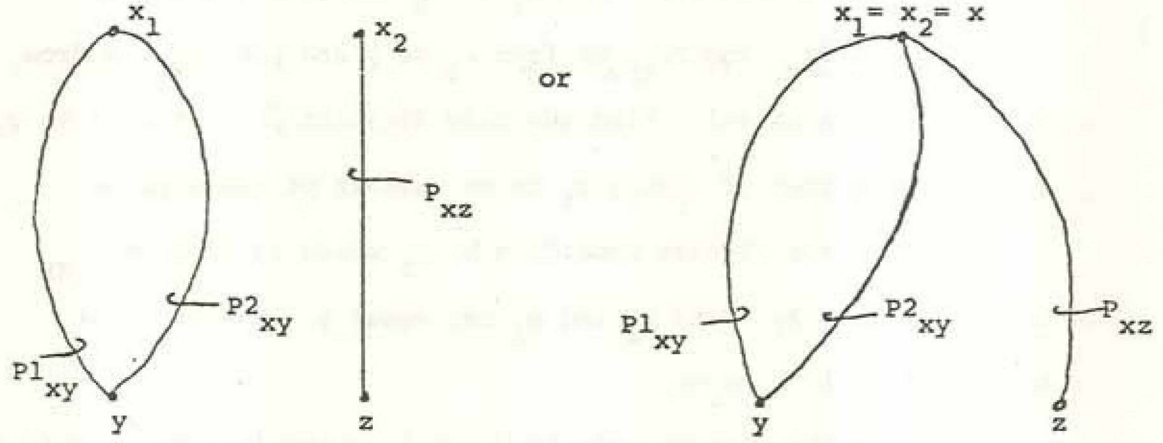
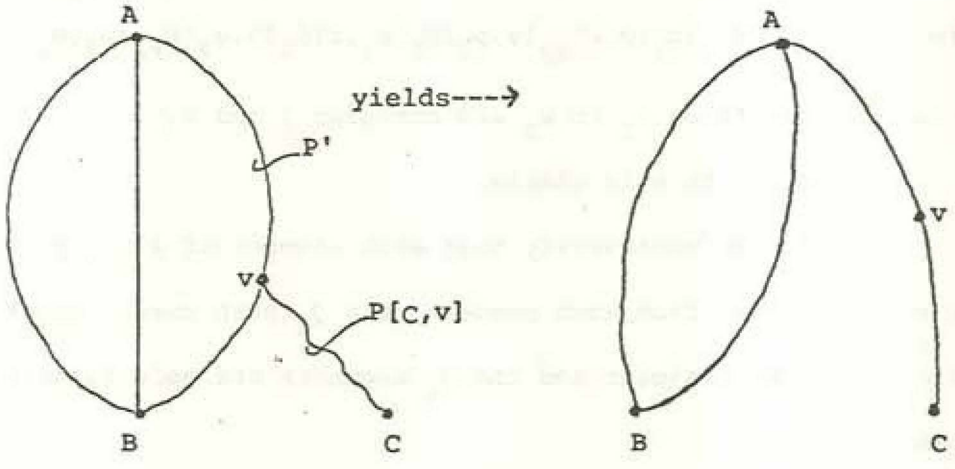


Figure IV.4-2: Construction to generate desired paths for Step 3 of Algorithm IV.3.



Node v may equal A, but v not equal to B.

Output: A simple path from x_1 to x_2 containing y and z.

Let paths $P1_{xy}$ and $P2_{xy}$ go from x_1 to y and path P_{xz} go from x_2 to z in Figure IV.4-1. Find two node disjoint paths from z to y, Q_1 and Q_2, such that if $x_1 \neq x_2$, x_1 is on neither of these paths.

Let p_i be the closest node to z on Q_i which is also on $P1_{xy}$ or $P2_{xy}$, i = 1 or 2. Both p_1 and p_2 may equal y. One of p_1 or p_2 may equal x, if $x_1=x_2=x$.

Let q_i be the closest node to x_2 on P_{xz} which is also on $Q_i[z,p_i]$. Both q_1 and q_2 may equal z; one of q_1 or q_2 may equal x_2. Note that $p_i=x$ implies that $q_i=x$. Without loss of generality, suppose that q_2 is closer to x_2 on P_{xz} than q_1. Then $q_1 \neq x_2$, and $p_1 \neq x$, if $x_1=x_2=x$.

Without loss of generality, suppose p_1 is on $P1_{xy}$. Consider the path $P = P2_{xy}[x_1,y]P1_{xy}[y,p_1]Q_1[p_1,z]Q_2[z,q_2]P_{xz}[q_2,x_2]$. This path goes from x_1 to x_2 and contains y and z.

Claim: Path P is simple.

Proof: We must verify that each segment of a P_{uv} path used is node disjoint from each segment of a Q_i path used. The P_{uv} segments are node disjoint and the Q_i segments are node disjoint by construction.

Compare $P2_{xy}[x_1,y]P1_{xy}[y,p_1]$ and $Q_1[p_1,z]$. Since p_1 is the closest node to z on Q_1 which is also on $P1_{xy}$ or $P2_{xy}$, the subpath $P2_{xy}[x_1,y]P1_{xy}[y,p_1]$ is node disjoint from $Q_1[p_1,z]$ except at p_1, where the two subpaths join. If $p_1=y$, then $P1_{xy}[y,y]$ is a null path.

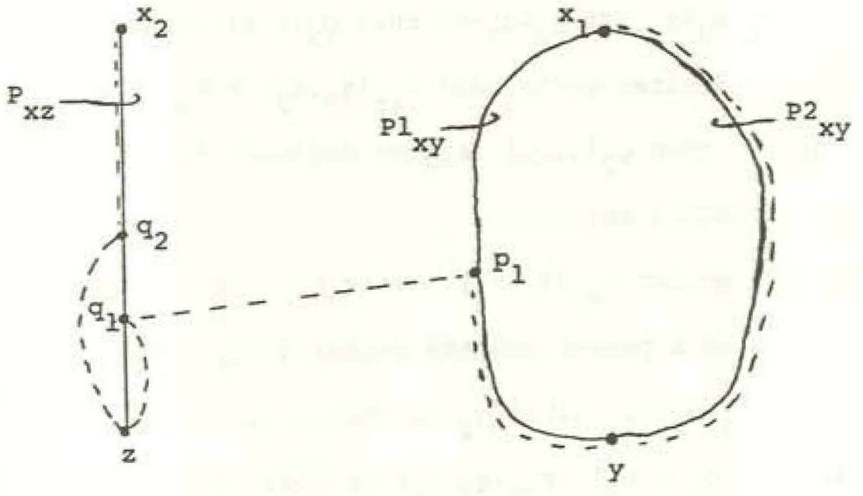
Now compare $P2_{xy}[x_1,y]P1_{xy}[y,p_1]$ with $Q_2[z,q_2]$. The path $Q_2[z,q_2]$ is a subpath of $Q_2[z,p_2]$. Since p_2 is the closest vertex to z on Q_2 also on $P2_{xy}$ or $P1_{xy}$, $Q_2[z,p_2]$ is node disjoint from $P2_{xy}[x_1,y]P1_{xy}[y,p_1]$ except possibly at p_2. However, $q_2 \neq p_2$ unless $q_2=p_2=x$, for $x_1=x_2=x$. If $q_2=p_2=x$, then $Q_2[z,x]$ joins $P2_{xy}[x,y]$ to complete the desired cycle, and $P_{xz}[q_2,x_2] = P_{xz}[x,x]$ is a null path. If $q_2 \neq p_2$, then $Q_2[z,q_2]$ is node disjoint from $P2_{xy}[x_1,y]$ $P1_{xy}[y,p_1]$, including endpoints.

Finally consider $P_{xz}[q_2,x_2]$. This path is a subpath of $P_{xz}[q_1,x_2]$. It is a proper subpath unless $q_1=q_2$. Note that $q_1=q_2$ if and only if $q_1=q_2=z$. Since q_2 is the closest node to x_2 on P_{xz} which is also on $Q_2[z,p_2]$, $P_{xz}[q_2,x_2]$ is node disjoint from $Q_2[z,q_2]$ except at q_2, where the two paths are intended to join. Since $P_{xz}[q_2,x_2]$ is a subpath of $P_{xz}[q_1,x_2]$, and q_1 is the closest node to x_2 on P_{xz} which is also on $Q_1[z,p_1]$, $P_{xz}[q_2,x_2]$ and $Q_1[z,p_1]$ are node disjoint, including endpoints unless $q_1=q_2=z$. But in this case, $Q_2[z,q_2]$ is a null path, and $P_{xz}[q_2,x_2]$ and $Q_1[p_1,z]$ join at z to form part of the cycle.

$\square$

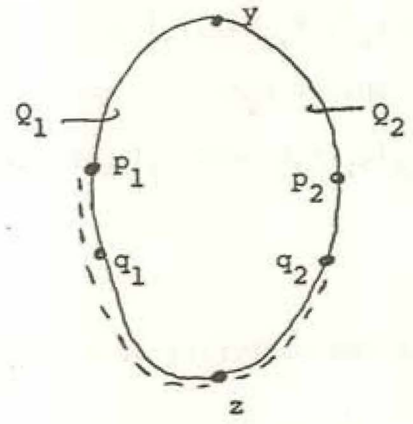
Figure IV.4-3 illustrates the positions of p_1, q_1 and q_2.

Figure IV.4-3: Positions of p_1, q_1, and q_2, and resulting path.



Nodes q_1 and q_2 may equal z; q_2 may equal x_2.

Where we have:



Nodes p_1 and p_2 may equal y; p_2 may equal x, if x_1=x_2=x.

----- indicates the desired path from x_1 to x_2 containing y and z.

IV.5 Timing of the Algorithms

The algorithms presented in Sections IV.3 and IV.4 rely heavily on Dinic's network flow algorithm. However, we never need to find more than a flow of three between any two nodes in a network. In particular, to determine if two nodes are triconnected requires finding a flow of three, and solving the subgraph homeomorphism problem for pattern graph $K_{1,2}$ requires finding a flow of two. Upon examination of the algorithm [Ta 1974, Ev 1975], we note that this implies that we need to find at most three augmenting paths in the network, and the time taken is $\mathcal{O}(|V_G|+|E_G|)$. In Section IV.2, we have shown how to find two cutpoints in $\mathcal{O}(|V_G|+|E_G|)$ operations. Note that the algorithm for biconnected components due to Hopcroft finds the articulation points of a graph and only takes $\mathcal{O}(|V_G|+|E_G|)$ operations. By appropriate bookkeeping, we can find an articulation point separating two given nodes while increasing the time taken by the algorithm by only a constant factor. We first consider Algorithm IV.4.

In Algorithm IV.4, we are given as input the three node disjoint paths from x_1 and x_2 (or x) to y and z. To find the two node disjoint paths from y to z takes $\mathcal{O}(|V_G|+|E_G|)$ operations using the network flow algorithm. To find p_1, p_2, q_1, and q_2 requires comparing nodes on the two sets of path. Suppose we first tabulate which nodes are on which paths. This will take $\mathcal{O}(|V_G|)$ steps. To find the closest node, u, to node w on a path P which is also on a set of paths, S, we process each node on path P in order, starting with w, by looking up in our table whether the node is also on a path in S. Since the number of paths in S is at most two, each node can be processed using at most two look ups. Thus, we can find u in time $\mathcal{O}(|V_G|)$. Once p_1, p_2, q_1, and q_2 are found in this manner, the construction of the desired path is immediate. We add up all the times above to conclude that Algorithm IV.4 can be executed in $\mathcal{O}(|V_G|+|E_G|)$ operations.

To determine the time taken by Algorithm IV.3, we will consider the algorithm step by step. We first note that each step is executed at most once. When the time taken by a step is $\mathcal{O}(|V_{G'}|+|E_{G'}|)$, where G' is a subgraph of G, we will use $\mathcal{O}(|V_G|+|E_G|)$ as an upper bound.

Step 1: The time to test if an edge is in E_G depends on the representation of E_G. Since the biconnected components and network flow algorithms require an adjacency list representation, we will assume this representation for E_G. Then, to find an edge, we must search the adjacency list of one of the endpoints of the edge. This takes $\mathcal{O}(|V_G|)$ operations. To find a path from A to B containing C, we solve the subgraph homeomorphism problem for pattern graph $K_{1,2}$. This requires $\mathcal{O}(|V_G|+|E_G|)$ operations. Thus, Step 1 takes $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 2: Breaking G into biconnected components requires $\mathcal{O}(|V_G|+|E_G|)$ operations. The test of whether A, B, and C are in the same biconnected component can be incorporated in the algorithm for finding biconnected components. The component containing A, B, and C, if it exists, will be part of the output of the algorithm. Thus, Step 2 requires $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 3: Merging nodes B and C requires $\mathcal{O}(|E_G|)$ operations to modify the adjacency lists. The network flow algorithm on N(BC) requires $\mathcal{O}(|V_G|+|E_G|)$ operations, since we need only find a flow of three. Since we execute these node merging and flow algorithms at most three times, this portion of Step 3 requires $\mathcal{O}(|V_G|+|E_G|)$ operations. If Algorithm IV.4 is then used, another $\mathcal{O}(|V_G|+|E_G|)$ operations are required.

Step 4: Finding cutpoints A_1 and A_2 and constructing $(G-\{A_1,A_2\})_B$ each require $\mathcal{O}(|V_G|+|E_G|)$ operations. (See Section IV.2) Determining if C is a node in $(G-\{A_1,A_2\})_B$ can be accomplished while $(G-\{A_1,A_2\})_B$ is being constructed. Thus, Step 4 takes $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 5: $G_1 = (G/\{A_1,A_2\})_B$ is constructed from $(G-\{A_1,A_2\})_B$ in $\mathcal{O}(|V_G|)$ operations. We apply Hopcroft's biconnected components algorithm to $(G-\{A_1,A_2\})_B$. This requires $\mathcal{O}(|V_G|+|E_G|)$ operations. If B and C are not biconnected in $(G-\{A_1,A_2\})_B$, an articulation point, x, separating B and C, is part of the output of the algorithm when appropriate bookkeeping is added. Constructing S_B and S_C requires $\mathcal{O}(|V_G|+|E_G|)$ operations. The fixed subgraph homeomorphism problem with pattern graph $K_{1,2}$ is solved at most four times, taking $\mathcal{O}(|V_G|+|E_G|)$ operations each time. Thus, Step 5 takes $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 6: Determining if any two of edges (A_1,B), (A_1,C), (A_2,B), and (A_2,C) are in G_1 requires $\mathcal{O}(|E_{G_1}|)$, which is also $\mathcal{O}(|E_G|)$, operations to search the adjacency lists. When two of these edges exist, we solve at most two instances of the fixed subgraph homeomorphism problem with pattern graph $K_{1,2}$. This requires $\mathcal{O}(|V_G|+|E_G|)$ operations. Thus, Step 6 requires at most $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 7: Determining if there are augmenting paths to A_1 and A_2 should be done by saving the information from Step 3, when the network flow algorithm is used on $N(BC)$. Which of the edges (A_1,B), (A_1,C), (A_2,B), and (A_2,C), if any, are in G_1 can be saved from Step 6. We use the algorithm for the fixed subgraph homeomorphism problem with pattern graph $K_{1,2}$ at most once. Thus, Step 7 takes at most $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 8: The biconnected components algorithm requires $\mathcal{O}(|V_G|+|E_G|)$ operations and will determine if either B or C is an articulation point in G_1. With appropriate bookkeeping, the biconnected components algorithm will also output S_1 and S_2, and the desired path in S_1, if G_1 is not biconnected. To find the desired path in S_2, we use the algorithm for the fixed subgraph homeomorphism problem with pattern graph $K_{1,2}$. This requires $\mathcal{O}(|V_G|+|E_G|)$ operations. Thus, Step 8 requires $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 9: Each set of node disjoint paths in $G_1(BC)$ is defined by the flow from [BC] to A_1 and A_2 in N(BC) merged with an augmenting path to A_1 or A_2. The paths in $G_1(BC)$ corresponding to the new flow in N(BC) can be found in $\mathcal{O}(|V_G|+|E_G|)$ operations by tracing the paths of flow in $G_1(BC)$. The paths in G_1 can be found by determining the correspondence between edges from B and C in G_1 and edges from [BC] in $G_1(BC)$ used by the paths. This requires $\mathcal{O}(|V_G|)$ operations to search the adjacency lists of B and C. Once we know the paths in G_1, we can determine which of the cases occurs. If case 3 occurs, we find a path from C to A_2 which does not contain B, and use this path to reduce case 3 to case 1 or case 2. This requires $\mathcal{O}(|V_G|+|E_G|)$ operations. If case 1 occurs, we use Algorithm IV.4, which requires another $\mathcal{O}(|V_G|+|E_G|)$ operations. We conclude that Step 9 requires $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 10: Merging the nodes A_1 and B, A_2 and C, A_1 and C, and A_2 and B requires $\mathcal{O}(|E_G|)$ operations. Determining if the pairs of merged nodes are triconnected requires $\mathcal{O}(|V_G|+|E_G|)$ operations using the network flow algorithm. Thus, Step 10 requires $\mathcal{O}(|V_G|+|E_G|)$ operations.

Step 11: To find the two disjoint paths from B to C containing neither A_1 nor A_2 requires $\mathcal{O}(|V_G|+|E_G|)$ operations. To find the augmenting path requires $\mathcal{O}(|V_G|+|E_G|)$ operations. Reducing cases 2 and 3 to case 1 or 5 requires $\mathcal{O}(|V_G|+|E_G|)$ operations to find a path from A_2 to B which does not contain C or a path from A_1 to C which does not contain B. The merging of this path with the set of node disjoint paths can be done in $\mathcal{O}(|V_G|)$ operations using the method described in our timing discussion of Algorithm IV.4. Here, the set S of disjoint paths contains three paths.

To process case 1, we need the sets of three node disjoint paths examined in Step 9. We assume these paths have been saved. To find the p_i's and q_i's using the method described for Algorithm IV.4 then takes $\mathcal{O}(|V_G|)$ operations. Finding the configuration of the p_i's and q_i's will require $\mathcal{O}(|V_G|)$ operations if the proper bookkeeping is done so that we always know on which paths a given node appears. Similarly, nodes r_1, r_2, s_1, and s_2 can be found and their positions on various paths determined in $\mathcal{O}(|V_G|)$ operations. Finally, path P_v can be found in $\mathcal{O}(|V_G|+|E_G|)$ operations, and v_1 and v_2 can be found in $\mathcal{O}(|V_G|)$ operations. We conclude that in the worst case of Step 11, a path from A_1 to A_2 containing B and C (in either order) can be found in $\mathcal{O}(|V_G|+|E_G|)$ operations.

To compute an upper bound on the time taken by Algorithm IV.3, we add up the worst case times for each step. Since each step takes at most $\mathcal{O}(|V_G|+|E_G|)$ operations, we conclude that Algorithm IV.3 requires at most $\mathcal{O}(|V_G|+|E_G|)$ operations. We cannot expect any algorithm which finds a cycle containing A, B, and C in G to take less than $\mathcal{O}(|V_G|+|E_G|)$ operations in the worst case, since it takes $\mathcal{O}(|V_G|+|E_G|)$ operations just to examine G. Thus, the time taken by Algorithm IV.3 is linear in the size of G.

IV.6 Conclusion

In this chapter, we have presented a linear time algorithm for the fixed subgraph homeomorphism problem when the pattern graph H is an undirected cycle of length three. This problem and the fixed subgraph homeomorphism problem when H is a tree of depth one are the only fixed subgraph homeomorphism problems which we know how to solve in polynomial time. In Chapter V, we discuss the most basic open problem for undirected graphs-- the fixed subgraph homeomorphism problem when H consists of two disjoint edges. We will show that this problem is fundamental to all other fixed subgraph homeomorphism problems which we cannot solve in polynomial time.

V The Two Disjoint Paths Problem

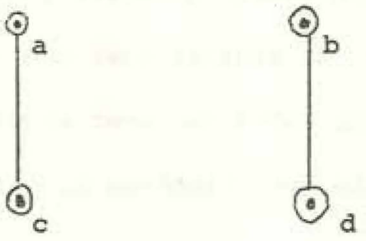
V.1 Introduction

In this chapter, we discuss the fixed subgraph homeomorphism problem when the pattern graph H consists of two disjoint undirected edges. This problem is called the two disjoint paths problem for undirected graphs since, in G, we are looking for two disjoint paths between given pairs of nodes. (See Figure V.1-1.) It is easily seen that the two disjoint paths problem for undirected graphs is reducible to the two disjoint paths problem for directed graphs. If H is undirected, $H=<\{a,b,c,d\},\{(a,c),(b,d)\}>$, and undirected graph G is input, we form directed graph H_d by making edges (a,c) and (b,d) directed. We form directed graph G_d by replacing each undirected edge (u,v) by directed edges (v,u) and (u,v). Then $H \leq_N G$ if and only if $H_d \leq_N G_d$, since each path in G_d corresponds to a path in G containing the same nodes, and each path in G corresponds to two paths in G_d, one in each direction, containing the same nodes.

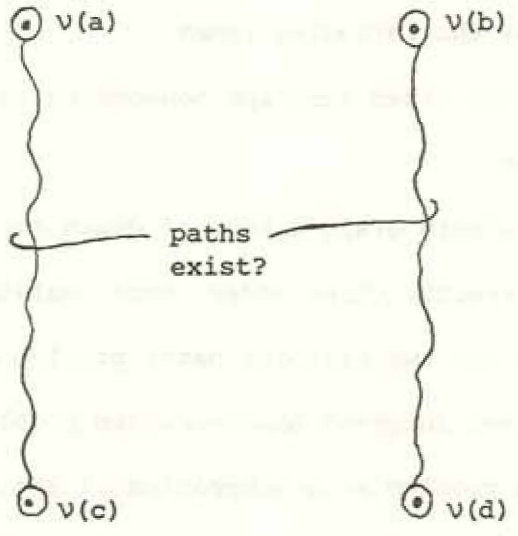
Recall that in Chapter III we showed that the two disjoint paths problem for directed graphs is equivalent to the two other basic open probems for directed graphs. In each problem, the pattern graph contains exactly two edges, and the mapping ν is specified. These three open problems represent the only fixed subgraph homeomorphism problems for directed graphs which we cannot solve in polynomial time when the pattern graph contains only two edges. The fact that the two disjoint path problem is reducible to these problems implies that, at worst, it is as hard to solve as these

Figure V.1-1: The Two Disjoint Paths Problem

pattern graph H:



in input graph G:



⊙ indicates a node whose image under ν or inverse image is specified.

problems. It is also possible that the two disjoint paths problem for undirected graphs is much easier to solve than the problems for directed graphs. Therefore, we suggest that the two disjoint paths problem for undirected graphs holds the most promise for solution. The relationships between the open problems discussed above are summarized in Figure V.1-2.

The fact that the two disjoint paths problem promises to be the "simplest" open problem to solve is not the only reason we focus on it. This problem is also the most fundamental of the open problems which are fixed subgraph homeomorphism problems for undirected graphs, as shown by the following lemma.

Lemma V.1.1 Any fixed subgraph homeomorphism problem for undirected graphs either

(a) has as pattern graph a tree of depth one or a cycle containing exactly three nodes once isolated nodes are removed

or (b) contains the two disjoint paths problem as a subproblem.

Proof: A fixed subgraph homeomorphism problem contains the two disjoint paths problem as a subproblem if the set $\alpha(E_H)$ must contain two paths which are node disjoint including their endpoints. The problems which contain the two disjoint paths problem as a subproblem are exactly those problems whose pattern graphs contain two edges with no common endpoints (two disjoint edges). We need to prove that any pattern graph, H, which does not contain two disjoint edges is a tree of depth one or a cycle containing exactly three nodes after isolated nodes have been removed. When $|E_H|$ is one or two, the result is immediately obtained by enumerating all possible graphs (up to isomorphism) with one or two edges and no isolated nodes. (See Figure V.1-3).

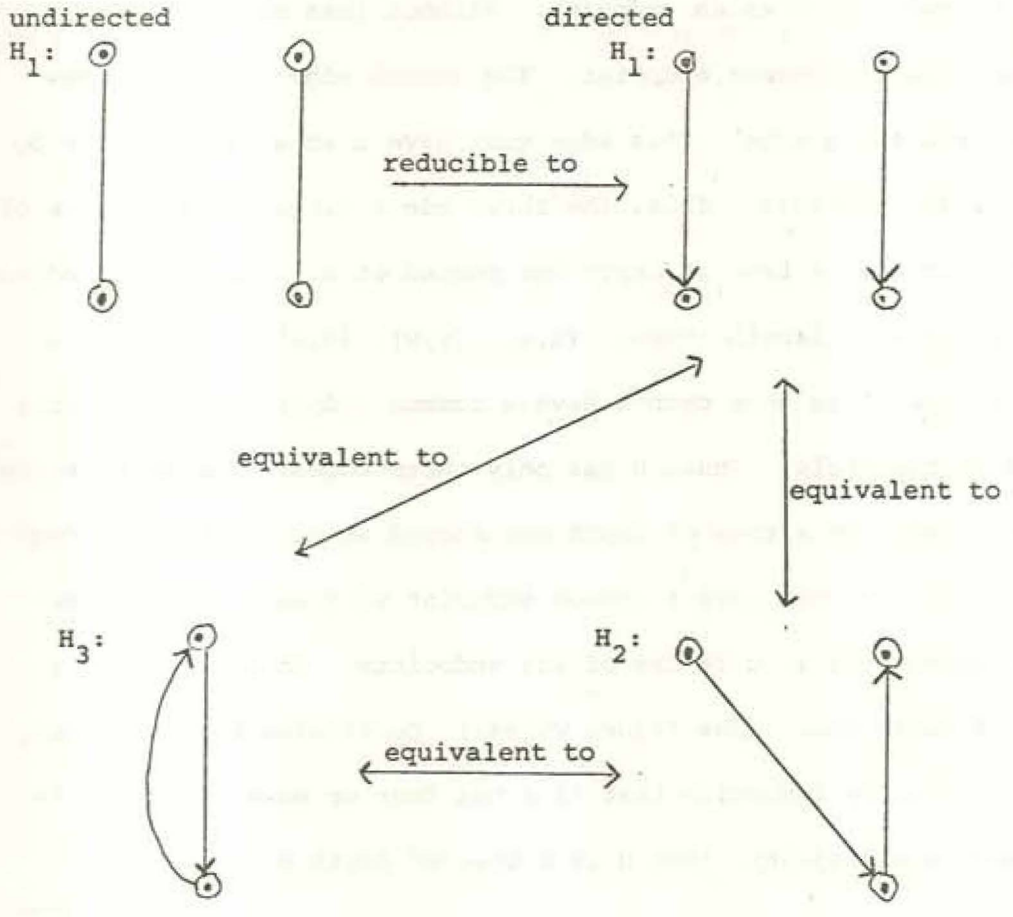
Consider any undirected graph H such that $|E_H| \geq 3$, and H contains no isolated nodes. Suppose H does not contain two disjoint edges. Choose any edge (u,v) in E_H. Now consider another edge in H. Since H does not contain two disjoint edges, this edge must have either u or v as an endpoint. Without loss of generality, assume u is the common endpoint. The second edge is (u,w). Now consider a third edge. This edge must have u as an endpoint or both v and w as endpoints. Thus, the three edges either form a cycle of length three or a tree of depth one rooted at u. Suppose the edges form a cycle of length three: <(u,v), (v,w), (w,u)>. If H has a fourth edge, this edge cannot have a common endpoint with all three edges of the cycle. Thus, H has only three edges. Now suppose the three edges form a tree of depth one rooted at u. If H has a fourth edge, this edge can have a common endpoint with each of the other three edges only if u is one of its endpoints. Thus H remains a tree of depth one. (See Figure V.1-4.) Continuing this reasoning, we can prove by induction that if H has four or more edges, no two of which are disjoint, then H is a tree of depth one.

$\square$

We see that any fixed subgraph homeomorphism problem for undirected graphs which we do not know how to solve in polynomial time contains the two disjoint paths problem as a subproblem. If we can solve any other open problem, we can solve the two disjoint paths problem by adding to G and H corresponding nodes and edges until we

Figure V.1-2:Relationships between open problems.

The fixed subgraph homeomorphism problem with pattern graph:



⊙ indicates that the node is in N_H.

Figure V.1-3: Enumeration for the proof of Lemma V.1.1.

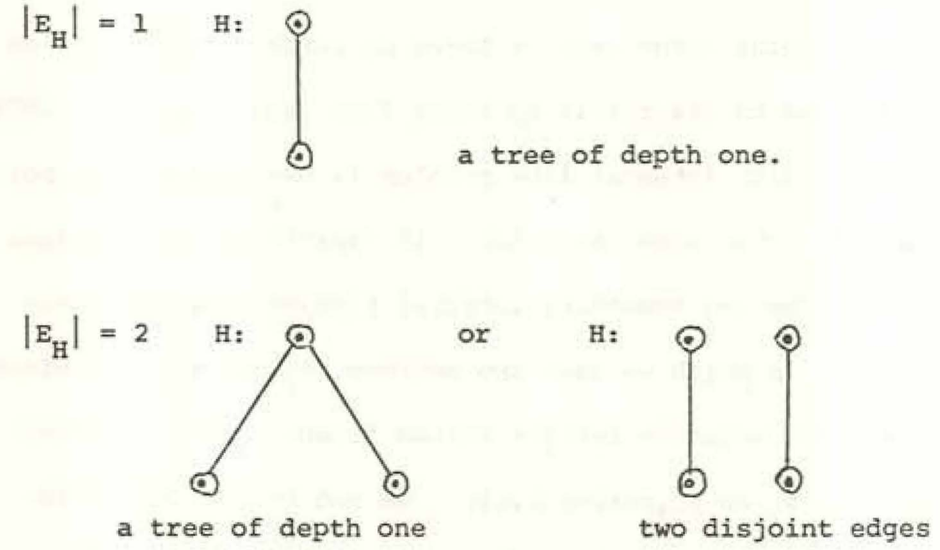
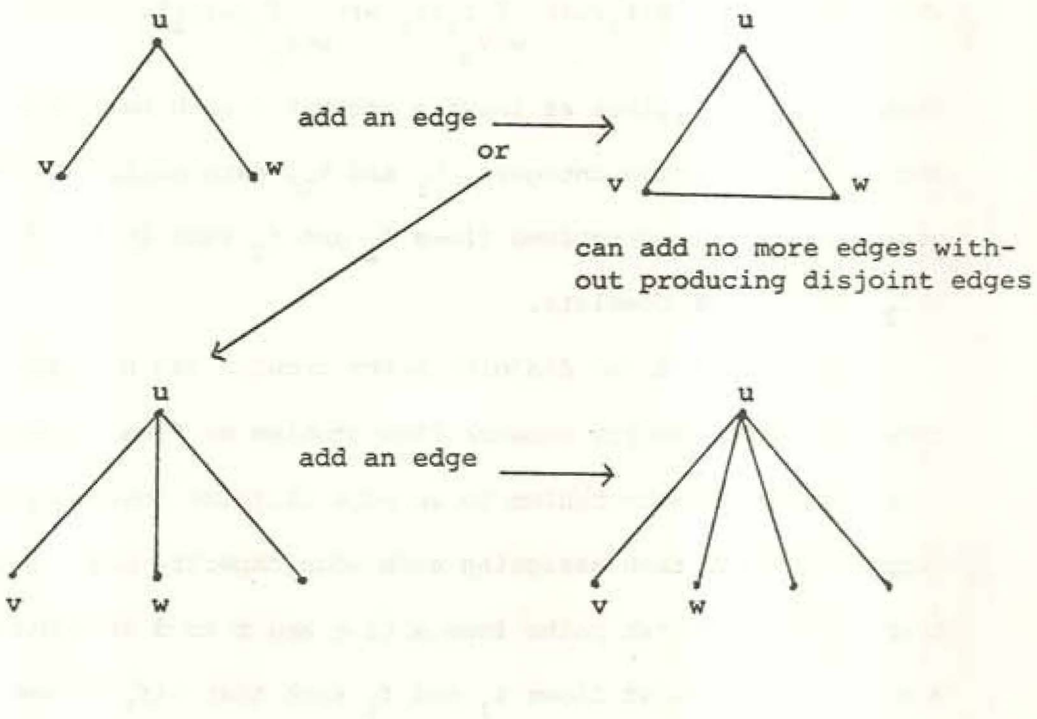


Figure V.1-4: Inductive argument for the proof of Lemma V.1.1.



have an instance of the problem we can solve.

The two disjoint paths problem takes on further importance as an open problem due to the result by Even, Itai, and Shamir [Ev 1976] that the two commodity integral flow problem is NP-complete for both directed and undirected networks, with unit capacities on the edges of the network. The two commodity integral flow problem is a network flow problem in which we have two sources, s_1 and s_2, two sinks, t_1 and t_2, and non-negative integer valued flows, f_1 and f_2, from s_1 to t_1 and from s_2 to t_2, respectively. We modify our definition of a network, N, to allow s_1 and s_2 to have incoming edges and t_1 and t_2 to have outgoing edges. We require that:

i) $f_1(u,w)+f_2(u,w) \leq c(u,w)$ for each $(u,w)\varepsilon E_N$

ii) $\sum_{w\varepsilon V_N} f_i(u,w) = \sum_{w\varepsilon V_N} f_i(w,u)$ for each $u\varepsilon V_N-\{s_i,t_i\}$ and $i=1$ or 2.

The value of f_i, $v(f_i)$, is $\sum_{w\varepsilon V_N} f_i(s_i,w) - \sum_{w\varepsilon V_N} f_i(w,s_i)$. Even, Itai, and Shamir show that, given as input a network N with unit edge capacities and two non-negative integers, k_1 and k_2, determining if there exist non-negative integer-valued flows f_1 and f_2 such that $v(f_1)=k_1$ and $v(f_2)=k_2$ is NP-complete.

The directed two disjoint paths problem can be viewed as an integral two commodity network flow problem by first reducing the node disjoint homeomorphism to an edge disjoint homeomorphism (see Chapter II), and then assigning each edge capacity one. Asking if there exist disjoint paths from a to c and b to d is equivalent to asking if there exist flows f_1 and f_2 such that $v(f_1)=1$ and $v(f_2)=1$, for s_1 corresponding to a, s_2 corresponding to b, t_1 corresponding to c, and t_2 corresponding to d. Since the undirected two disjoint paths problem is reducible to the directed two disjoint paths problem, it is also reducible to the integral two commodity network flow problem. We would like to know if the two disjoint paths problem is also sufficiently difficult to be NP-complete. If $NP \neq P$, it is also possible that the two disjoint paths problem neither has a polynomial time algorithm nor is NP-complete [Lad]. Since proving this condition is equivalent to proving $NP \neq P$, we are not optimistic about establishing such a result. If the two disjoint paths problem does prove to have a polynomial time algorithm, we can ask for what k,if any, does the k disjoint paths problem become NP-complete, and for what k, if any, does the integral two commodity flow problem for $v(f_1)$ and $v(f_2)$ no larger than k become NP-complete when the algorithm is allowed to depend on k.

We have shown in the discussion above that the two disjoint paths problem not only has a key role in the quest for algorithms to solve fixed subgraph homeomorphism problems, but also has importance in our understanding of the hierarchy of complexities of problems. In the next section, we discuss the progress which has been made in finding polynomial time algorithms to solve the problem.

V.2 Attempts at Solution

Attempts to solve the two disjoint paths problem have resulted in polynomial time algorithms to solve the problem when the input graph G has certain properties. Perl and Shiloach [Perl] have shown that when G is a triconnected planar graph, the two disjoint paths problem can be solve by an algorithm requiring $\mathcal{O}(|E_G|)$ steps. In fact, the algorithm presented for triconnected planar graphs is applicable for a planar graph G whenever nodes a and c are triconnected and nodes b and d are triconnected, where disjoint paths from a to c and b to d are desired. Perl and Shiloach also show that in a triconnected chordal graph[1], there are two disjoint paths between any two pairs of nodes in the graph. The paths can be found using $\mathcal{O}(|E_G|)$ operations. They claim that A. Itai can solve the two disjoint paths problem for planar or chordal graphs which are not triconnected in $\mathcal{O}(|E_G|)$ operations by using the algorithm of Hopcroft and Tarjan to separate G into triconnected components [Ho 1973b], and solving instances of the problem within these triconnected components.

The paper by Perl and Shiloach also discusses the two disjoint paths problem for directed graphs and for edge disjoint, rather than node disjoint, paths. They present methods of reducing the two edge disjoint paths problem for undirected (or directed) graphs to the

[1] A chordal graph is an undirected graph, G, such that for any cycle of length greater than three in G, there is an edge in G, called a chord, connecting two nodes which are on the cycle, but not adjacent on the cycle.

two node disjoint paths problem for undirected (or directed) graphs. These methods of reduction are specific to the two disjoint paths problem and simplier than our general methods of Chapter II. It is shown that if an undirected graph is three-edge-connected (i.e. any set of edges whose removal separates the graph is of size at least three), then there are two edge disjoint paths between any two pairs of nodes in the graph. An algorithm executable in $\mathcal{O}(|V_G|\cdot|E_G|)$ steps is presented to solve the two (node) disjoint paths problem for acyclic directed graphs.

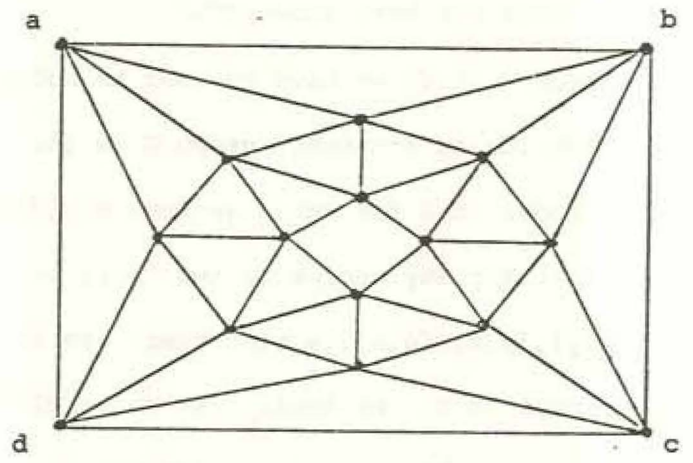
Several of the results by Perl and Shiloach discussed above give properties of G which guarantee the existence of two disjoint paths in G between any two pairs of nodes. Larman and Mani [Lar] and Watkins [Wa] also address this question. Recall from Chapter IV that triconnectivity of G was sufficient to guarantee the existence of a cycle containing any three given nodes of G. We would like a similar connectivity result for the two disjoint paths problem. However, Watkins illustrates that 5-connectivity of G does not guarantee the existence of two disjoint paths between any two pairs of nodes in G. Figure V.2-1 is the counterexample used. (This graph also appears in [Lar].) Larman and Mani do prove that if G is 6×2^{15}-connected, then there exist two disjoint paths between any two pairs of nodes in G. However, the size of the connectivity is so large as to be unusable for practical algorithms. Watkins shows that if the connectivity of G is at least four, and $K_5 \underset{N}{\leq} G$, where K_5 is the complete graph on five nodes, then there exist two disjoint paths between any two pairs of nodes in G. Note that $K_5 \leq_N G$ implies that G is not planar, by Kuratowski's result. It is conjectured by Watkins that if the connectivity of G is at least four and $K_{3,3} \leq_N G$, where $K_{3,3}$ is the complete bipartite graph on two sets of three nodes, then there exist two disjoint paths between any two pairs of nodes in G. If this conjecture is true, then any non-planar graph whose connectivity is at least four contains two disjoint paths between any two pairs of nodes. The proof of this conjecture remains an open problem. The minimum connectivity of an arbitrary graph, G, which guarantees the existence of two disjoint paths between any two pairs of nodes in G is also an open question [Perl].

Consider again the existence of two disjoint paths from a to c and from b to d, when a,b,c,and d are four known nodes in the graph G. Figure V.2-2 illustrates that no particular connectivity between a and c or b and d guarantees that the two disjoint paths exist. The particular graph shown in Figure V.2-2 is planar, and the algorithm of Perl and Shiloach can be used. However, this algorithm uses Theorem V.2.1 below to decide whether the disjoint paths from a to c and b to d exist.

Theorem V.2.1 (Theorem 4.1 of [Perl]) Let G be a planar graph. If there is an imbedding of G in the plane such that nodes a,b,c,and d are on one face of G in that cyclic order, then two disjoint paths--one from a to c and one from b to d-- do not exist in G.

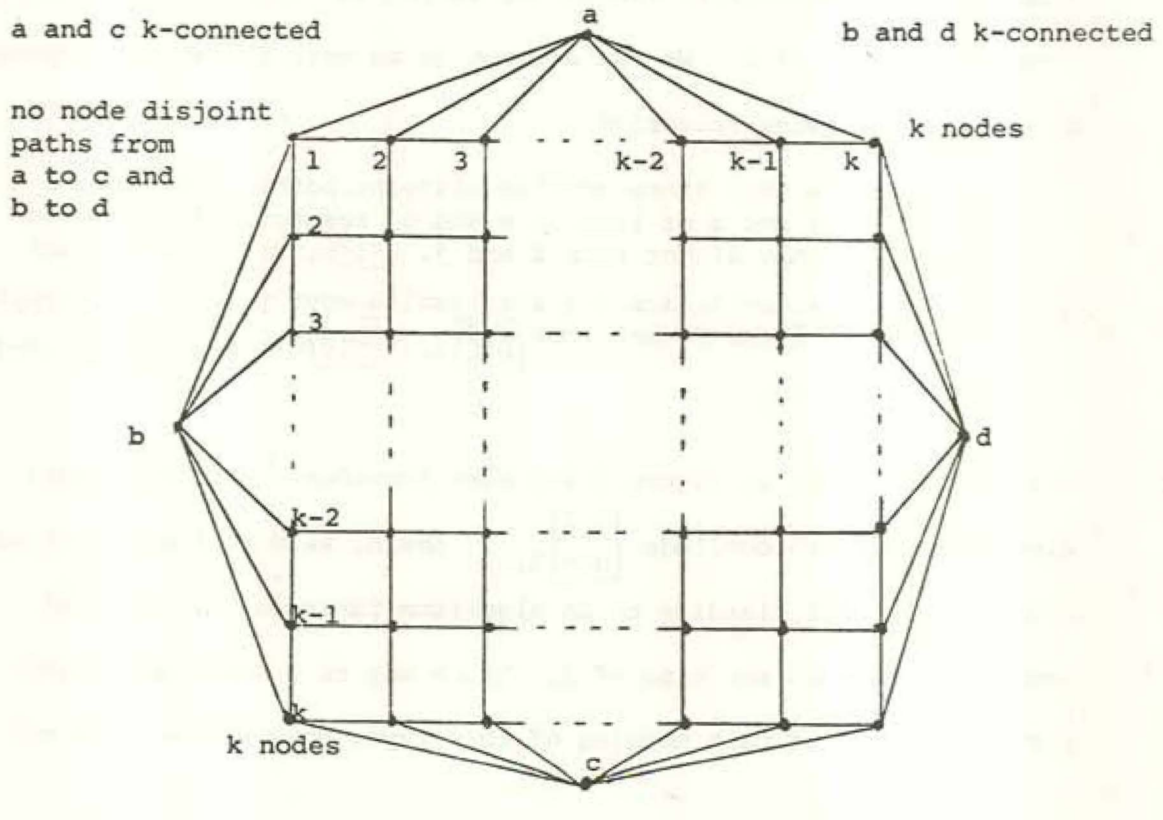
When an arbitrary graph G is considered, we have not found criteria which will allow us to decide if the two disjoint paths

Figure V.2-1: Counterexample to 5-connectivity guaranteeing two node disjoint paths between any two pairs of nodes.



There are no disjoint paths from a to c and b to d.

Figure V.2-2: Counterexample to any connectivity of a and c and b and d guaranteeing two node disjoint paths from a to c and b to d.



exist. Our attempts to devise an algorithm similar in methodology to Algorithm IV.3 have not been successful.

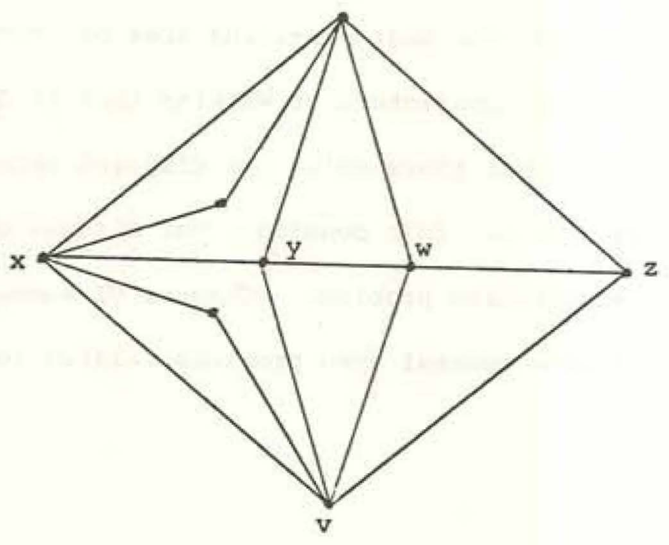
Another approach which we have pursued to solve the two disjoint paths problem for an arbitrary graph G is that of building up sets of pairs of nodes in G for which we know disjoint paths exist. Suppose we know that for any choice of two pairs of endpoints from among the set {[x,y],[y,z],[u,w],[w,v]}, there are two disjoint paths (excluding endpoints) in G. We would like to conclude that there are two node disjoint paths in G with endpoints [x,z] and [u,v]. However, Figure V.2-3 shows a graph G which provides a counterexample to this conclusion. The problem is that different paths may be used between the same endpoints when testing for different sets of disjoint paths. Keeping track of all the paths used leads to an algorithm which is essentially exhaustive search and requires exponential time in the size of G. We may attempt to be more clever and attempt to use the following criterion:

> We conclude that there are two disjoint paths in G between u and v and x and z of lengths m and n, respectively, (denoted $\begin{bmatrix} x,z \\ u,v \end{bmatrix}_{n,m}$) only if for each i and j, $0 \le i \le n$, $0 \le j \le m$, such that no two of i,j,n-i, and m-j are simultaneously equal to zero, there exist nodes ξ and ζ such that $\begin{bmatrix} x,\zeta \\ u,\xi \end{bmatrix}_{i,j}$, $\begin{bmatrix} \zeta,z \\ \xi,v \end{bmatrix}_{n-i,m-j}$, $\begin{bmatrix} x,\zeta \\ \xi,v \end{bmatrix}_{i,m-j}$, and $\begin{bmatrix} \zeta,z \\ u,\xi \end{bmatrix}_{n-i,j}$.

However, the graph in Figure V.2-3 also provides a counterexample when we attempt to conclude $\begin{bmatrix} x,z \\ u,v \end{bmatrix}_{3,2}$. Again, we must keep track of actual paths used, leading to an algorithm taking an exponential number of steps in the size of G. There may be a polynomial time algorithm based on path merging of this type. However, we have not

Figure V.2-3: Counterexample for path-joining criteria.

For:



<x,y> disjoint from <u,w>

<x,y> disjoint from <w,v>

<x,y> disjoint from <y,w,z>

<y,v,z> disjoint from <u,w>

<y,u,z> disjoint from <w,v>

<u,w> disjoint from <w,v>

However, there are no node disjoint paths from x to z and u to v.

found a criterion which allows us to do this.

In this section we have attempted to present the understanding of the two disjoint paths problem which we have developed during our attempts to solve it. Given the results of Perl, Shiloach, and Itai for planar graphs, the most promising area of investigation appears to be based on the conjecture of Watkins that if G is 4-connected and not planar, then there exist two disjoint paths between any two pairs of nodes in G. This concludes our discussion of solutions to subgraph homeomorphism problems. Chapter VI summarizes our results and presents some general open problems related to subgraph homeomorphism.

VI Conclusion

VI.1 Summary

In this thesis we have presented definitions of the subgraph homeomorphism problem for node disjoint homeomorphism, edge disjoint homeomorphism, directed graphs, and undirected graphs. After observing that the most general subgraph homeomorphism problem--when both graphs H and G are input--is NP-complete, we concentrated on finding algorithms which depend on the pattern graph H and are of polynomial time in the size of the input graph G. We showed that in all cases, an edge disjoint subgraph homeomorphism problem could be solved by solving a node disjoint subgraph homeomorphism problem. However, only in the fixed, directed case could we find a reduction allowing us to solve a node disjoint homeomorphism problem by solving an edge disjoint homeomorphism problem.

The algorithms we have presented to solve subgraph homeomorphism problems rely heavily on the polynomial time network flow algorithm discussed in Chapter III. Our algorithm for finding a cycle in graph G containing three given nodes in G, presented in Chapter IV, combines this network flow algorithm (to find node disjoint paths in G) with algorithms to break G into components. We are able, through splitting G into components and "cutting and pasting" node disjoint paths of G, to determine if the cycle exists and, if so, to construct it in linear time in the size of G. This algorithm and the algorithm for solving the fixed subgraph homeomorphism problem when the pattern graph is a tree of depth one are the only polynomial time algorithms for fixed subgraph homeomorphism problems which we know of. Combining these algorithms with the three reductions presented in Chapter III allows us to solve a number of problems when the node mapping ν is partially specified or unspecified.

VI.2 Topics for Further Investigation

We have spent Chapter V discussing what we believe to be the most alluring subgraph homeomorphism problem for which no polynomial time algorithm is known--the two disjoint paths problem. This problem not only appears to be the simpliest open problem but also is a fundamental problem for all open fixed subgraph homeomorphism problems. There are, however, other areas of research which do not involve solving a particular subgraph homeomorphism problem.

Suppose we can solve the subgraph homeomorphism problem for a pattern graph $H = <V_H,E_H>$. We would like to know if we can solve the subgraph homeomorphism problem if we add an edge to H or delete an edge from H, keeping N_H constant. It would be informative to characterize the situations in which we can solve the new problem for $H^- = <V_H, E_H-\{(u,v)\}>$ or $H^+ = <V_H,E_H \cup \{(u,v)\}>$, where $(u,v) \in (V_H)^2$. For example, if u and v are in N_H, then we can solve the subgraph homeomorphism problem for H^- in a straightforward manner by adding edge (u,v) back into H^-, adding $(\rho(u),\rho(v))$ to the input graph G, and solving the subgraph homeomorphism problem for H with the modified graph G as input. If nodes u and v are not in N_H, it is not clear how to use the solution for H to find a solution for H^-, since we do not know what nodes in G should correspond to u and v. Even less is known about the solution of the subgraph homeomorphism problem for H^+. For example, it is easy to solve the fixed subgraph homeomorphism problem when H consists of four nodes, a,b,c, and d, and one edge, (a,b). We delete the nodes $\nu(c)$ and $\nu(d)$ from G and determine if $\nu(a)$ and $\nu(b)$ are connected in G. However, if we add edge (c,d) to H, we get the two disjoint paths problem, which has so far defied solution.

Similar questions can be asked about the set N_H. In Section III.1, we showed that we can always remove nodes from N_H and solve the resulting problem in polynomial time if the original problem was solvable in polynomial time. We would like to know when we can add nodes of V_H to N_H and be able to solve the problem. For example, we can solve the subgraph homeomorphism problem for H a tree of depth two in polynomial time when the leaves of H are not in N_H. How do we solve the problem in polynomial time when we add leaves to N_H? Note that we can always add or delete <u>isolated</u> nodes from V_H and add <u>isolated</u> nodes to N_H. If we have deleted a node from V_H, we simply add the node back into V_H, add an isolated node to the input graph, and solve the original problem. If we have added a node to V_H, we solve the old problem for each graph resulting from deleting a node from the input graph. If we have added an isolated node to N_H, we delete this node from N_H, delete the corresponding node from the input graph, add an isolated node to the input graph, and solve the old problem. Given that we can add isolated nodes in this fashion, a generally applicable method for solving $H^+ \leq_N G$ once a polynomial time algorithm for $H \leq_N G$ is known would provide us with a means of solving $H \leq_N G$, for any graph H, in polynomial time. We would begin with a simple pattern graph for which a polynomial time algorithm is known and add nodes to V_H and N_H and edges to E_H until the desired graph is constructed.

Another line of investigation which may be pursued is: "Given pattern graph H and specified node set N_H, what properties of the input graph G guarantee that $H \leq_N G$ (or $H \leq_E G$) for any partial specification ρ?" We discussed this question in Chapter V in conjunction with the two disjoint paths problem. An example of the type of results we would like is the theorem by Perl and Shiloach [Perl]: Any triconnected chordal graph contains two node disjoint paths between any two pairs of nodes. In this case, the properties of G guaranteeing two disjoint paths are triconnectivity and chordality. Watkins [Wa] and Larman and Mani [Lar] have investigated this question for the n disjoint paths problem. Larman and Mani have also investigated this question for pattern graphs K_n (the complete undirected graph on n nodes) and $K_{n,n}$ (the complete undirected bipartite graph on two sets of n nodes). They consider the question for both $N_H = \emptyset$ and $N_H = V_H$. Similarly, we may ask, "What properties of G and $\rho(N_H)$ guarantee that $H \leq_N G$ (or $H \leq_E G$) for a given partial specification, ρ?" For example, we know that there is a cycle containing three given nodes of G whenever there are three disjoint paths from one of these nodes to the other two.

The ultimate goal of this research is to answer the question, "Is there a polynomial time algorithm for every subgraph homeomorphism problem when the algorithm is allowed to depend on the pattern graph, H, and the partial specification set, N_H?" Hunt and Szymanski [Hunt] have asked this question for floating subgraph homeomorphism problems in conjunction with their research on programming schema. The results presented in this thesis represent our present knowledge of the answer to this question. The research problems proposed in this section are designed to further our knowledge of the subgraph homeomorphism problem with this ultimate goal.

Bibliography

[Ah] 1. Aho, A., Hopcroft, J., Ullman, J., The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1974.

[Be] 2. Berge, Claude, The Theory of Graphs and its Applications, John Wiley & Sons, Inc., New York, 1966.

[Di] 3. 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 1976] 4. 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] 5. Even, S., Tarjan, R.E., "Network Flow and Testing Graph Connectivity," SIAM Journal on Computing, Vol. 4, No. 4, Dec. 1975, pp. 507-518.

[Fo] 6. Ford, L.R., Fulkerson, D.R., Flows in Networks, Princeton University Press, Princeton, New Jersey, 1962.

[Ge] 7. Geller, Dennis, "Forbidden Subgraphs," Proof Techniques in Graph Theory, Frank Harary, ed., Acedemic Press, New York, 1969, pp. 37-47.

[Ha 1971] 8. Harary, Frank, Graph Theory, Addison-Wesley Publishing Co., Reading, Mass., 1971.

[Ha 1973] 9. 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] 10. Hopcroft,J.E., Tarjan, R,E., "Algorithm 447: Efficient Algorithms for Graph Manipulation," Communications of the ACM, Vol. 16, No. 6, June 1973, pp. 372-378.

[Ho 1973b] 11. Hopcroft, J.E., Tarjan, R.E., "Dividing a Graph into Triconnected Components," SIAM Journal on Computing, Vol. 2, No. 3, Sept. 1973, pp. 135-158.

[Hu] 12. Hu, T.C., Integer Programming and Network Flows, Addison-Wesley Publishing Co., Reading, Mass., 1969.

[Hunt] 13. 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.

[Lad] 14. Ladner, R. "On the Structure of Polynomial Time Reducibility," Journal of the ACM, Vol. 22, No. 1, Jan. 1975, pp. 155-171.

[Lar] 15. 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.

[Lu] 16. Luckham, D.C., Park, D.M.R., Paterson, M.S., "On Formalized Computer Programs," Journal of Computer and Systems Sciences, Vol. 4, No. 3, June 1970, pp. 200-249.

[Perl] 17. 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.

[Ri] 18. Rivest, R., private communication.

[Ta 1972] 19. 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] 20. Tarjan, R.E., "Testing Graph Connectivity," Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1974, pp. 185-193.

[Wa] 21. Watkins, Mark, "On the Existence of Certain Disjoint Arcs in Graphs," Duke Mathematical Journal, Vol. 35, 1968, pp. 231-246.

Note: reference [Lu] is not cited in the text.