

University of Montana

ScholarWorks at University of Montana

Graduate Student Theses, Dissertations, &
Professional Papers

Graduate School

1997

Analysis of road network accessibility

Dale A. Hamilton

The University of Montana

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

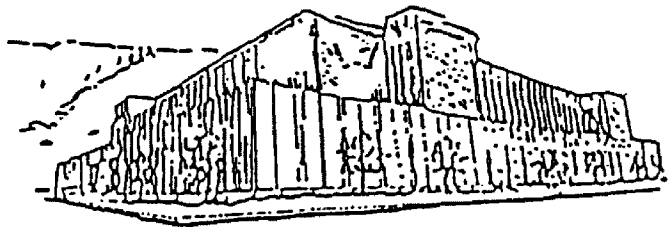
Let us know how access to this document benefits you.

Recommended Citation

Hamilton, Dale A., "Analysis of road network accessibility" (1997). *Graduate Student Theses, Dissertations, & Professional Papers*. 6593.

<https://scholarworks.umt.edu/etd/6593>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.



Maureen and Mike
MANSFIELD LIBRARY

The University of **MONTANA**

Permission is granted by the author to reproduce this material in its entirety,
provided that this material is used for scholarly purposes and is properly cited in
published works and reports.

*** Please check "Yes" or "No" and provide signature ***

Yes, I grant permission _____
No, I do not grant permission _____

Author's Signature Doc A Hamilton

Date 6/2/97

Any copying for commercial purposes or financial gain may be undertaken only with
the author's explicit consent.

ANALYSIS OF ROAD NETWORK ACCESSIBILITY

by

Dale A. Hamilton

B.S. Seattle Pacific University, 1987

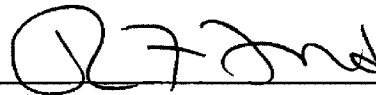
A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science - Computer Science

The University of Montana

1997

Approved by:



Chairperson



Dean, Graduate School

6-5-97

Date

UMI Number: EP37394

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP37394

Published by ProQuest LLC (2013). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



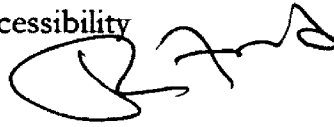
ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Hamilton, Dale A., M.S., May 1997

Computer Science

Analysis Of Road Network Accessibility

Chairperson: Ray Ford



Many interesting problems in natural resource management involve the assessment of the impact of vehicle traffic moving into, through, and out of a designated area via a road network. In order to exactly determine the extent of the influence of the road network, it is necessary to determine which segments of the network are accessible to vehicle traffic, when they are accessible, and to what classes of vehicles they are accessible. These determinations can be made using a derivative of a classical "connected components" graph algorithm, taking as inputs the road network represented as a graph and an associated set of "barriers" which constrain travel across the network. In practice, barriers are real constraints, such as gates, kelly humps, revegetation, etc. Algorithmically, barriers are represented as nodes on the graph, attributed with values that define the nature of the constraint. The algorithm groups together "connected arcs" as those which are accessible to each other without passing through any barriers. The algorithm next determines the effects of the restrictions imposed by the barriers on each set of connected arcs. Algorithm output is a road network represented as a graph with arcs attributed according to the nature of the travel restriction imposed by the set of barriers.

This effort will focus on the algorithm, and explains how it complements the analysis available using only the facilities embedded in a typical commercial GIS, such as ESRI's Arc/Info. Examples show how the algorithm can be used to record the arc restriction attributes for use with other types of analysis. The examples also show how the algorithm can be utilized to display which portions of the road network are restricted for a given set of dates. Additional types of analysis that will utilize arc restriction attributes as input are discussed.

TABLE OF CONTENTS

List of Figures	ii
List of Algorithms	iii
Acknowledgments	iv
Chapter 1 - Problem Definition and Motivation	1
Chapter 2 - Problem Background	6
Relevant Graph Theory Concepts	6
Relevant Geographic Information System Concepts	15
Arc/Info Specific GIS Concepts	17
Analysis Related Work - Graph Algorithms and GIS Functions	20
Chapter 3 - A Graph Theoretic Algorithm For Road Closure Analysis	25
A Classical “Connected Components” Graph Algorithm	25
A Derivative Algorithm That Can Handle Seasonal Barriers	29
NAD Attribute Identification	34
Analysis of the CONDITIONAL_CONNECTED_COMPONENTS Algorithm	44
Chapter 4 - Comparison Of Solutions	48
Road Closure Analysis Solution Comparisons	48
Analysis Results Visualization Comparisons	51
Chapter 5 - Integration Of Graph Theoretic Solution With A GIS	53
Access Point Management	53
Barrier Management	55
Running The Analysis	55
Analysis Results Visualization	57
Summary	58
Appendix A - Network Accessibility Analyzer Instructions	59
References	66

LIST OF FIGURES

<i>Number</i>	<i>Contents</i>	<i>Page</i>
2.1	Pictorial Representation Of A Graph	7
2.2	A Simple Road Network	11
2.3	Graph Representation Of Road Network Within Study Area	12
2.4	Barrier and Access Point INAD and BAD Attribute Values	13
2.5	Connected Element Sets	14
3.1	Connected Graph Components	26
3.2	Condensed Graph of Connected Sets and Barriers	34
3.3	Condensed Graph Prior To NAD Attribute Determination	37
3.4	Condensed Graph After First Breadth First Search	40
3.5	Condensed Graph After Second Breadth First Search	43

LIST OF ALGORITHMS

<i>Number</i>	<i>Algorithm</i>	<i>Page</i>
2.1	Single-Source Shortest Path (Dijkstra's Algorithm)	23
3.1	CONNECTED-COMPONENTS	28
3.2	CONDITIONAL-CONNECTED-COMPONENTS	33
3.3	NAD Attribute Determination Algorithm	36

ACKNOWLEDGMENTS

The author wishes to acknowledge the guidance and encouragement provided by the graduate committee chairperson, Dr. Ray Ford.

A thank you is extended to Mr. Bill Tanke and others at the United States Forest Service Northern Region for providing the road closure analysis problem and being of assistance in learning Arc/Info.

Thanks also to Jonathan and Nicholas Hamilton for their patience while Daddy was writing this thesis.

A special thanks to my wife, Leslie Hamilton, for her patience and encouragement in addition to covering many of my family responsibilities during this research effort.

PROBLEM DEFINITION AND MOTIVATION

In driving through National Forest land one often finds locked gates that block through traffic on a road. Interestingly enough, one occasionally finds that the road on both sides of the gate is accessible to vehicle traffic. It is possible the road manager intended just to stop through traffic, but more often the intent was to restrict road access to an area. When both sides of a locked gate are accessible, the Forest Service obviously did not succeed.

This is the motivation for the problem investigated here: finding a way to identify which portions of a road network are restricted by a set of barriers, and in addition, determining the nature of the restrictions imposed by the barriers (e.g. when are the road segments restricted, to which vehicle types are the restrictions applied.) A road may be closed by placing a gate on the road or by a more permanent means such as a kelly hump or revegetation. If any road leading into a subset of the network has not been closed, access to the subnetwork has not been adequately restricted.

The determination of which road segments are restricted is dependent on the restrictions imposed by the barriers. When determining whether access past a barrier is possible, the answer may not be “yes” or “no”, but the answer might also be “maybe”. Maybe a person can get past the gate if they are in the right vehicle class on the right day. It is necessary

to not just identify where the barriers are located, but also when the barriers are closed and to which vehicle classes it is closed. For example, two gates may be placed on a road, intending to close the road between the gates. If the first gate is permanently closed and the second gate is only closed seasonally, the restricted portion of the road between the gates is only closed seasonally, regardless of the fact that the first gate is permanently closed. An additional aspect of the seasonal nature of gates occurs when gates with different closure dates close the same subnetwork of the road network. When the closure dates of the gates do not coincide, the effective closure period of the affected road subnetwork is the intersection of the closure periods of the associated gates. Gates may also impose different restrictions on different classes of vehicles. A gate may specify that automobiles may not proceed past the gate at any time while All Terrain Vehicles (ATVs) may go past the gate for a portion of the year (say May 1 through October 15) and snowmobiles may proceed past the gate during a different part of the year (say December 1 through March 30.)

Many organizations which are responsible for natural resource management rely on data indicating when road segments are accessible and when they are not. This data assists in critical decision support issues they are faced with concerning the area impacted or serviced by the road network. Even though road network information is becoming available in digital form, the analysis is still most often done manually. One manager who is responsible for finding these restriction attributes reports that the effort to perform the analysis manually generally takes many person-months of effort. Other factors make a manual analysis less than desirable besides the labor cost involved in performing this

analysis manually. The road systems many organizations manage resembles a plate of spaghetti. When attempting to determine the restrictions imposed on a particular road segment, it is hard to assure that the person manually performing the analysis has identified every possible path which affects the accessibility of that road segment. Another factor which makes manual attribution undesirable is the fact that few road systems are static. Barriers are routinely added or removed or the barrier closure schedules are changed. In a complex network, barrier change can have dramatic and widespread effects, so the complete analysis should be repeated prior to the change. The addition of a road also dictates that the restriction analysis be repeated. Maintaining a complete and consistent analysis is thus much harder and more error-prone than simply attributing the road network once when digital information first becomes available.

Ideally, an automated analysis tool can be used to perform the analysis, allowing the restricted road segments to be identified and attributed in a matter of minutes. Besides requiring significantly lower labor cost to perform the analysis, this eliminates the possibility of human error. An automated tool also allows the managing organization to run simulations to determine the effects that additional sets of gates can have on the road network. Without the availability of a tool of this nature, it might not be possible or cost effective to *a priori* determine the effects of large sets of additional gates.

In addition to labor cost savings, the data generated by this effort can be utilized as input by many additional types of analysis. For example, many types of analysis involve determining road closure alternatives that ensure that access into an area has been

restricted. The restricted area can range from a sensitive wildlife habitat to a recreation area that has suffered ill effects from excessive human use. This analysis can focus on finding “a simple solution”, “an optimal solution” or a set of solutions for public comment. A simple solution would involve determining the placement of a set of gates (or other closure mechanisms) that ensure that every path into the area is closed. It would not attempt to quantify or optimize the placement of these gates in any way, but would just make sure all routes into the area are closed. An optimal solution would try to accomplish area closure and meet some sort of quantitative criteria. For example, an optional solution might select a set of gates to close that minimizes the number of new gates that need to be installed or minimizes the distance gates are from a main road while also minimizing the area outside the closed area that is affected by the road closures. A set of possible solutions might try to find all solutions that satisfy some specified cost constraints. Other similar types of analysis involve identifying nonaccessible or roadless areas which lie within the region serviced or impacted by the road network, or within a given distance of an accessible road segment on the network.

Other analyses would utilize data generated by this effort as input. These analyses include but are not limited to the following topics: identifying which parts of a road system are closed to traffic when a specific gate at a specific point of a road network is closed or open; determining an area’s resulting road density (distance of accessible roads in the polygon on a given date divided by the size of the area) when a portion of the road subnetwork in the area is closed; determining the effect on travel distance between points

when gates are opened or closed; and finding good or optimal routes that allow staff to manually close a set of gates.

This thesis defines a class of problems termed *road closure analysis problems*, and looks at specific problem instances, solutions possible using commercial GIS packages, and more ambitious solutions possible by combining data from a GIS with custom algorithms from the study of graphs.

PROBLEM BACKGROUND

A variety of issues are relevant to understanding the *road closure analysis problem*. These issues include concepts from graph theory, geographic information systems (GIS) concepts, and other related work.

2.1 Relevant Graph Theory Concepts

The main element in the road closure analysis problem is the road network that defines access to an area. A road network can be thought of as a graph. Formally, a *graph*, G , is a pair (V,E) where V is a finite, nonempty set whose elements are called *vertices*, and E is a set of pairs from V called *edges* (Baase, 1988). An example is shown in Figure 2.1. Vertices are used to represent intersections, as well as other points of interest on the road network such as barriers, bridges and changes in road surface. Intuitively, edges represent bi-directional uninterrupted, homogeneous road segments. Vertices can have one or more associated edges. The number of edges that are associated with a vertex is referred to as the *degree*. Figure 2.1 represents a graph where $V = \{1,2,3,4,5,6,7,8,9\}$ and $E = \{(1,2), (2,3), (2,4), (2,8), (4,5), (4,6), (5,7), (5,8), (8,9)\}$. Note that vertex 5 has degree 3, vertex 9 has degree 2, and vertex 7 has degree 1.

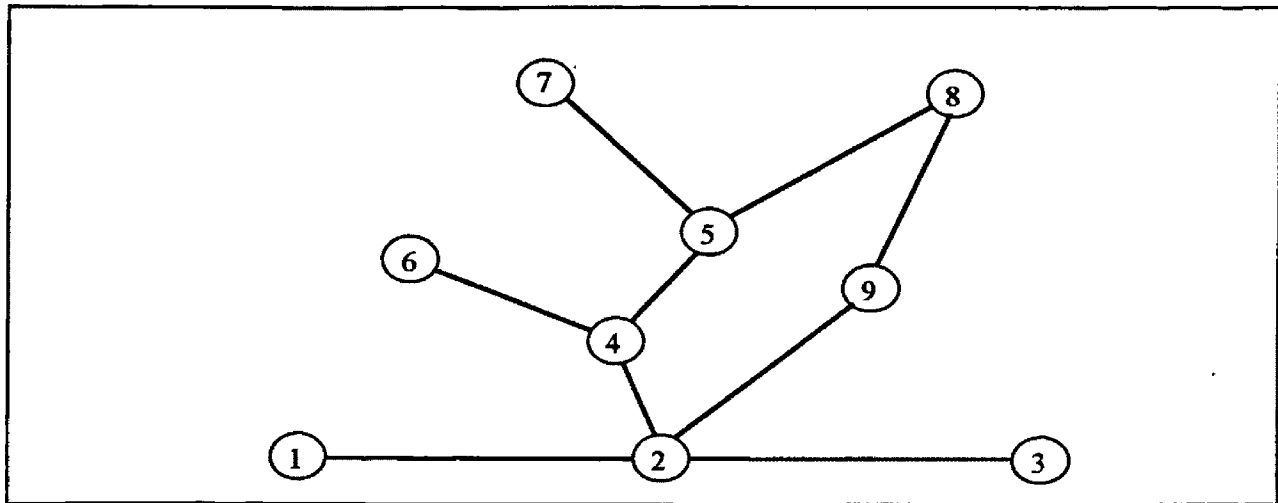


Figure 2.1. Pictorial Representation of a Graph

Both vertices and edges can be *attributed* by having special information, or data values attached to them. In road closure analysis this will assist in identifying to which class a particular vertex or edge belongs, or to record other data specific to the vertex or edge. To reflect the times when vehicle traffic access is restricted on parts of the graph, it is assumed that all vertices and edges have a *not accessible dates* attribute (NAD). This attribute defines the set of days when different vehicle classes are known to be restricted from accessing the road network at that location. Initially, this attribute has the value “undefined” for each member of G .

The objective of the road analysis problem from a graph theory perspective is to represent a road network with a particular type of graph, assign initial attribute values to some vertices, then compute other key attribute values for other member edges and vertices. Any type of graph analysis is sensitive to the size of the graph. The road closure analysis only needs to consider the portion of the road network which is local to the analysis area. For example, if road closures are being analyzed in the Bitterroot National Forest, it is not

necessary to include all the roads in the United States in the graph. Including more of the road network than necessary in the graph will hamper performance of the analysis. Intuitively, including the roads that lie “within” or “near” the analysis area in the analysis graph will suffice, but it may be difficult to determine just which roads to include or exclude before doing the analysis.

In some instances the size of the graph needed to include the surrounding loop of highways can be quite large. It may not be feasible to expect the organization doing the analysis on the study area to translate the whole road network into a graph so as to achieve this goal, especially if a large portion of the road network lies outside their jurisdiction. As a result, the main road or highway may or may not be included in the graph. In the event that the main road or highway is not included in the graph, special *access point* vertices of degree equal to one are introduced to summarize the accessibility. The edge from such a point leads “into” the road network of interest; as explained below, data attributes at this point abstract the access to/from the outside world. Typically, access points are identified when the graph is first reduced from a larger network.

As an interface between the study graph and the outside world, each access point has an *initial not accessible dates* (INAD) attribute, which lists the set of days when different vehicle classes are known to be restricted from accessing the road network from the outside at that location. For example, if an access point can only be reached by passing through a gate which is closed January 1 through May 15, the access point’s INAD has the value {Jan 1, Jan2, ..., May 15}. If the access point can be reached at any time in the

year its INAD is empty. If there are multiple paths off the graph from the highway to the access point, its INAD will be the intersection of the restrictions on accessibility for each of the paths. For example, if one path to an access point to a highway is closed January 1 through June 15 and another path is closed April 1 through July 31, the access point's INAD is {Apr 1, ..., June 15} If a single path to an access point includes more than one barrier, the dates vehicle traffic can obtain access along that path is the intersection of the dates each barrier along that path restricts traffic.

Another class of vertices which is of interest in road closure analysis are *barriers*, which record activities on the road network which have the ability to restrict traffic flow. Examples of barriers include gates, washouts, kelly humps, revegetation, and travel management signs. In graph terms, a barrier is represented as a vertex with degree equal to two with a *barred access dates* attribute (BAD) which lists the set of dates the barrier restricts traffic flow.

In review, there are four steps that must be taken to represent a road network as a graph. The steps are as follows.

1. Create an initial graph which consists of a set of vertices and edges, such that vertices represent intersections and edges represent road segments that connect the intersections. Restrict the graph to an analysis area representing the portion of the road network of interest by drawing a line that intersects only edges and that eventually encloses the analysis area.

2. Create and add new access point vertices and corresponding edges, such that access vertices are placed on every edge which is intersected by the study area boundary. The new edges simply replace those whose termination points fall now outside the study boundary. That is, an edge $\langle X, Y \rangle$ which is intersected by the study boundary with Y on the outside, is replaced by $\langle X, A \rangle$ with A being a new access point.
3. Add barrier vertices and corresponding new edges, such that the barrier vertex is placed on every edge on which a barrier is found. That is, an edge $\langle X, Y \rangle$ which includes a barrier is replaced by $\langle X, B \rangle$ and $\langle B, Y \rangle$, and the characteristics of the barrier are defined in the values of the attributes for barrier node B .
4. Determine initial-not-accessible-dates attribute for each access point and the barred-access-dates attribute for each barrier.

As an example, consider the simple road network illustrated in Figure 2.2, which shows both highways, secondary roads, and barriers. The study area is represented as a dashed rectangle. A schedule of barrier closures specifies when each of the barriers is closed.

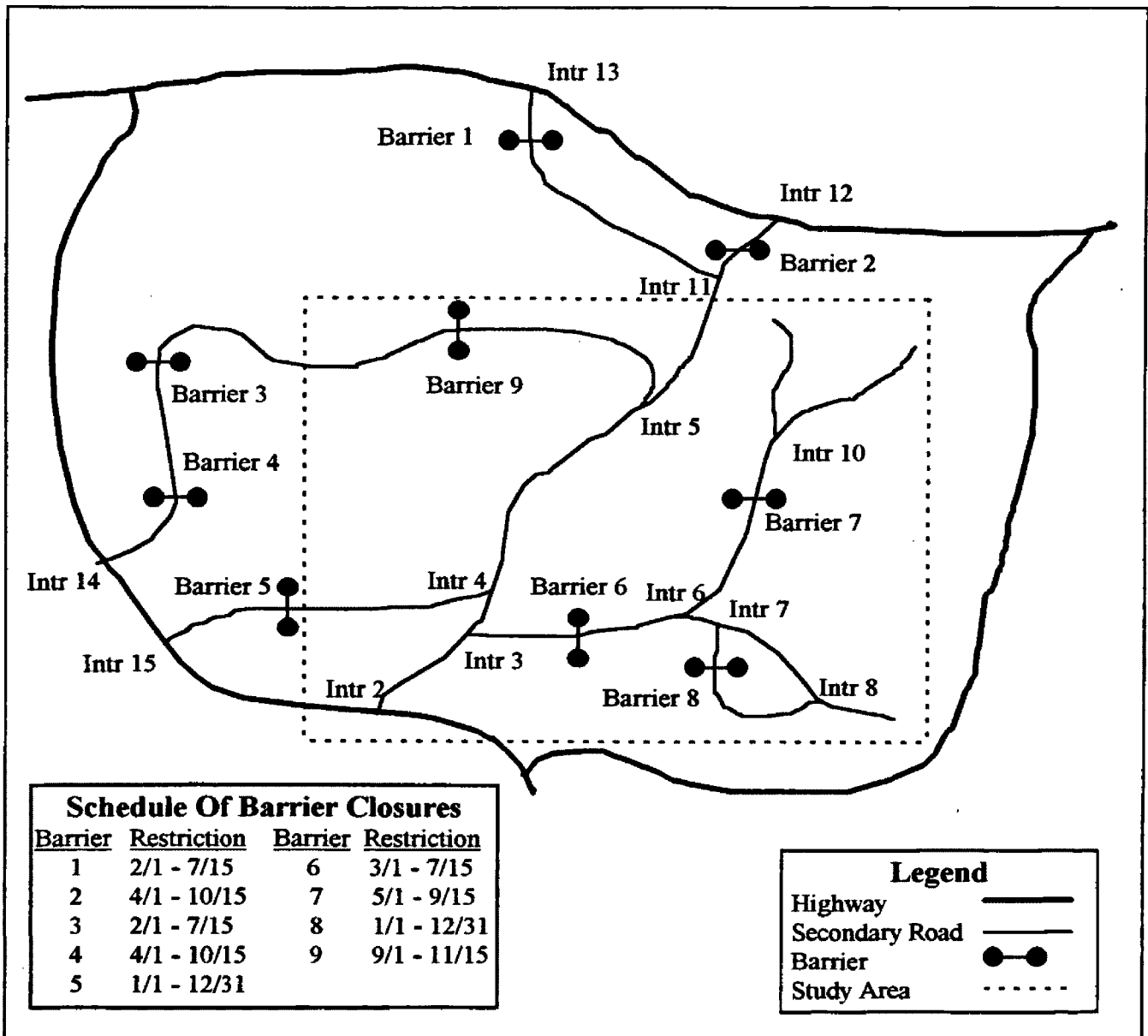


Figure 2.2 A Simple Road Network

The graph representation of the same road network, restricted to the study area, is illustrated in Figure 2.3. The graph contains three classes of vertices. Vertices with a label “A” represent access point vertices, vertices with a label “B” represent barrier vertices, and vertices which are only labeled numerically represent intersection points in the road network (i.e. vertices that are neither access points or barriers.)

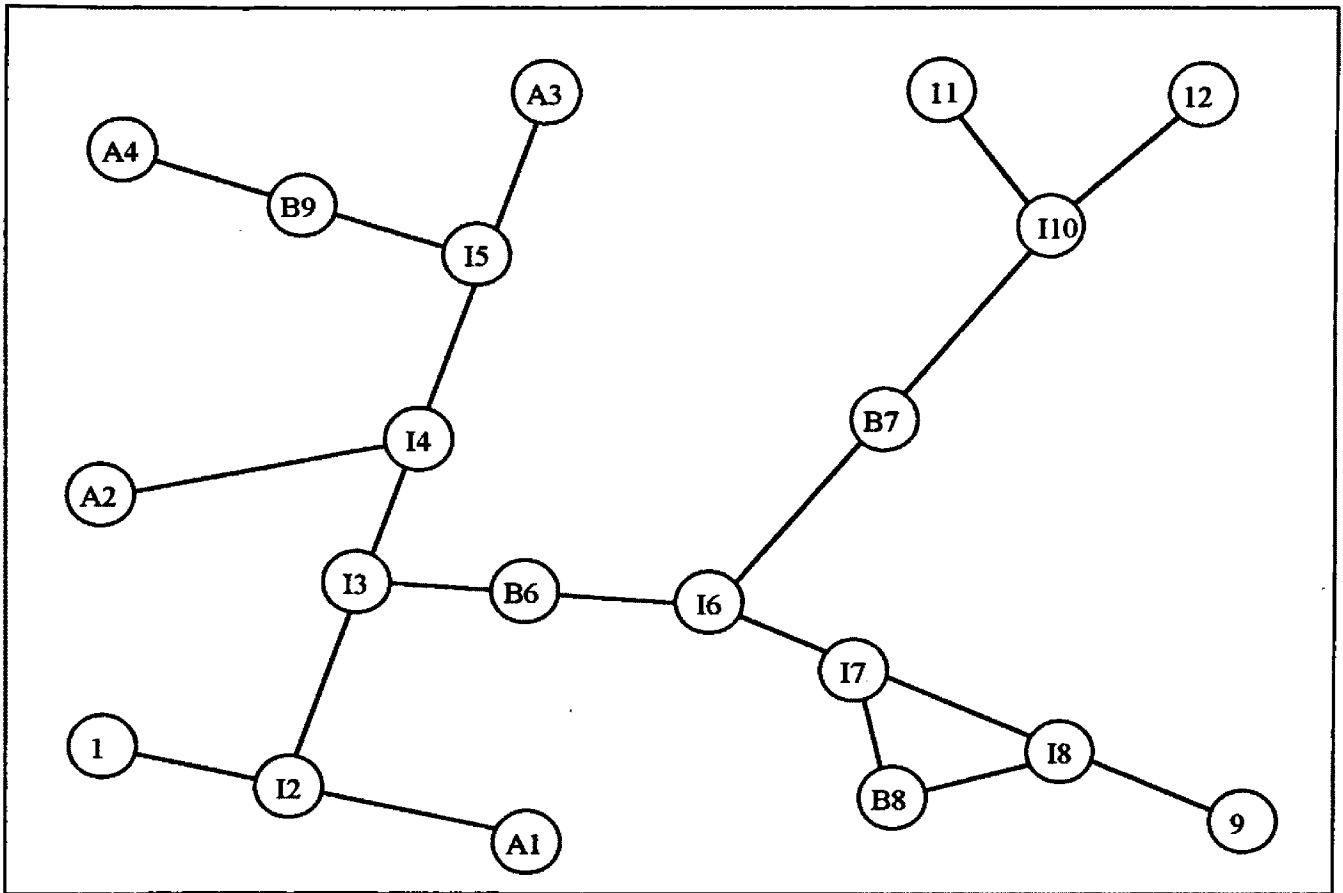


Figure 2.3 Graph Representation Of Road Network Within Study Area

Figure 2.4 shows a table of INAD and BAD values for graph access point and barrier vertices. For example, access point $A3$ corresponds to the point where the path from the highway which passes through Barrier 1 and the path from the highway which passes through Barrier 2 enter the study area. Barrier 1 is closed February 1 through July 15 and Barrier 2 is closed April 1 through October 15. $A3$'s resulting INAD attribute is {April 1, ..., July 15} which is the intersection of the INADs for the paths to the access point (i.e. $\{2/1..7/15\} \cap \{4/1..10/15\} = \{4/1..7/15\}$). Barrier vertex $B6$ has a BAD attribute value of

{3/1..7/15} which is derived from Barrier 6's entry in the schedule of closures. Figure 2.4 lists the INAD and BAD attribute values for the barrier and access point vertices of the graph shown in Figure 2.3.

<u>Vertex</u>	<u>Vertex Class</u>	<u>INAD Attribute</u>	<u>BAD Attribute</u>
A1	Access Point	{}	
A2	Access Point	{Jan 1, ..., Dec 31}	
A3	Access Point	{Apr 1, ..., Jul 15}	
A4	Access Point	{Feb 1, ..., Oct 15}	
B6	Barrier		{Mar 1, ..., Jul 15}
B7	Barrier		{May 1, ..., Sept 15}
B8	Barrier		{Jan 1, ..., Dec 31}
B9	Barrier		{Sept 1, ..., Nov 15}

Fig 2.4 Barrier And Access Point INAD And BAD Attribute Values

Given a graph with access point and barrier information, the goal is to determine the affects of the barrier and access point INAD attributes on the NAD attributes of the graph's member edges. To do this, we note that a *connected element set* (CES) is a set of vertices and edges which can be reached from X without passing through any barrier. These sets include edges that are associated with a barrier, but do not include any barrier vertices. It is assumed that each connected element set has a *not-accessible-dates* (NAD) attribute, which has the same value as the NAD attribute of all members of that connected element set. Each set also has an INAD attribute, which is computed as the intersection of the INAD attributes for each of the set's member access points. If none of the member vertices are access points, the set's INAD attribute value is unknown.

The graph shown in Figure 2.3 has four connected element sets. The table in Figure 2.5 shows their members and INAD attributes.

<u>Set</u>	<u>Members</u>	<u>INAD</u>
CES_1	$\langle \{A4\}, \{ \langle A4, B9 \rangle \} \rangle$	{Feb 1, ..., Oct 15}
CES_2	$\langle \{1, A1, A2, A3, I2, I3, I4, I5\},$ $\{ \langle 1, I2 \rangle, \langle I2, A1 \rangle, \langle I2, I3 \rangle, \langle I3, B6 \rangle, \langle I3, I4 \rangle,$ $\langle I4, A2 \rangle, \langle I4, I5 \rangle, \langle I5, B9 \rangle, \langle I5, A3 \rangle \} \rangle,$	{}
CES_3	$\langle \{16, 17, 18, 9\},$ $\{ \langle B6, I6 \rangle, \langle I6, I7 \rangle, \langle I7, B8 \rangle, \langle I7, I8 \rangle, \langle I8, B8 \rangle,$ $\langle I8, 9 \rangle \} \rangle$	unknown
CES_4	$\langle \{110, 11, 12\},$ $\{ \langle B7, I10 \rangle, \langle I10, 11 \rangle, \langle I10, 12 \rangle \} \rangle$	unknown

Figure 2.5 Connected Element Sets

Note that the edges on either side of barrier vertex $B8$ are in the same connected element set (CES_3). This is a result of the path which exists between the edges associated with $B8$ (i.e. $\langle I7, B8 \rangle \rightarrow \langle I7, I8 \rangle \rightarrow \langle I8, B8 \rangle$).

The objective of this effort is to develop an algorithm that takes as input a graph, finds the connected element sets and their INAD values, then ripples the effects of the barriers'

BAD attribute values across the connected element sets to determine each element's NAD value.

2.2 Relevant Geographic Information System Concepts

Generally, abstract graph theory concepts have similar corresponding concepts in GIS. The edges in a graph correspond to *arcs* in GIS where the arcs are assumed to allow traffic flow in either direction, and the vertices in a graph correspond to *nodes* in GIS. The distinction between graph-vertex and GIS-node is important, because there is a meaning of "vertex" in GIS which does not correspond to a graph-vertex.

The key difference between an abstract graph and a graph in GIS is that of correlating member elements to real spatial locations. An abstract graph shows only connectivity, not correlation to physical space, whereas in a GIS it is essential to correlate each arc and node to physical space. For example, graph-vertices are used to represent intersections and other points of interest on the road network; however, the resulting graph does not indicate where these vertices are located spatially, just that they exist. Similarly, the graph's edges correspond to road segments on the road network, but they do not reflect the physical location of the road segments. It is possible to correlate graph vertices and edges to physical space by adding appropriate attributes, attached to both vertices and edges. A GIS graph does exactly that, correlating both the nodes and arcs to physical space by attributing them with spatial characteristics. For example, each node has attributes that indicate its physical location (e.g. latitude and longitude). Arcs are

attributed in a more complex manner to record the spatial path they follow between nodes. Many GIS attribute arcs with lists of connected vectors that approximate the arc's path through space. The endpoints of these vectors represent special new types of nodes, *GIS vertices*, which approximate real road paths with sequences of straight line segments. A GIS graph which shows both nodes and vertices arranged in a coordinate system according to spatial values can be thought of as a digital map which approximates a road network. As with abstract graphs, additional attributes can be added to record events that occur along the associated road network.

Algorithmically, GIS-resident barriers and access points are conceptually no different than they are in graph theory. Both are represented as nodes that are attributed to identify them as either a barrier or an access point. They also have an attribute which reflects the impact the node has on vehicle access on the network at that point (i.e. BAD for barriers and INAD for access points).

When discussing road closure analysis with computer scientists and GIS professionals, it becomes apparent that the two communities use different terminology to describe the same components of the analysis. What a computer scientist calls edges and vertices, a GIS professional refers to as arcs and nodes. A computer scientist thinks of an edge as a linear object which connects two vertices. The GIS community understands that an arc is an undirected linear feature that connects two nodes. However, when comparing vertices and nodes, a discrepancy in terms becomes apparent, stemming from whether edges/arcs are "atomic" objects. In graph theory edges are atomic, i.e. not divisible into smaller

units. In GIS, arcs are divided into smaller segments to map a course through physical space. Thus, arcs represent a series of connected line segments where each segment is connected at each end to either a node or another line segment; vertices are the points where the line segments connect to each other. This discrepancy makes it necessary to avoid using the term vertices in order to avoid confusion between the two disciplines. In GIS terms, a road network is represented as an undirected graph $G = (N, A)$, where N is the set of nodes on the road network, A is the set of undirected arcs on the graph that connect graph nodes, each node is attributed to reflect its spatial coordinates, and each arc is subdivided into a set of line segments connected by vertices with spatial coordinates.

From an abstract GIS perspective, one objective of road closure analysis is to take a road coverage with the associated sets of access points and barriers, and determine which dates each road node and/or arc is accessible. In addition, the spatial information allows the analysis to be extended to answer questions such as “On what days is it possible for a vehicle to pass within distance D of point $\langle X, Y \rangle$?”

2.3 Arc/Info Specific GIS Concepts

Arc/Info was developed by Environmental Systems Research Institute (ESRI) in the 1970's, and has grown to be the industry leading GIS. General market dominance plus the fact that Arc/Info is the GIS of choice for the U.S. Forest Service, which is the agency with primary interest in road closure analysis, led to the selection of Arc/Info as the

“implementation GIS” for this project. Arc/Info is also available in the Computer Science labs at the University of Montana, which made it convenient to use.

In Arc/Info, a *line coverage* can be used to represent a simple digital map which shows a network’s road segments and events that occur along the associated road network. The line coverage is a series of $\langle X, Y \rangle$ coordinate pairs representing nodes. The coordinate pairs are assumed to be ordered, such that arcs are assumed to connect each successive coordinate pair. Internally, each arc is represented as a sequence of coordinate pairs which represent vertices, with line segments assumed to join each pair of vertices (ESRI, 1995A). In contrast, a *point coverage* is a geographic data set consisting of a series of $\langle X, Y \rangle$ coordinate pairs representing points (ESRI, 1995A), but without any arcs assumed between the pairs. A point coverage can be thought of as a digital map which shows only the geographic location of points (e.g. house, well).

Barrier data are currently represented in at least two forms by the Forest Service. Some analysts represent barriers as events in a point coverage. Utilizing a point coverage representation, a barrier is represented by a single point which has an associated $\langle X, Y \rangle$ coordinates. These point coverage events have attributes describing information about the nature of the restriction imposed by the barrier. Barriers stored in this manner must be correlated, or “coregistered” somehow on the separate road network as nodes. Arc/Info’s SPLIT command (ESRI, 1994) can be used to correlate members of a point coverage onto a line coverage. SPLIT takes as input a line coverage and a point coverage. The point coverage is processed on a one-by-one basis by taking each member of the point coverage,

identifying the closest arc to that member point, and adding a node at the location where the arc is closest to the member point along with corresponding new arcs associated with the new node to replace the original arc. That is, an arc $\langle A, B \rangle$ with a point X' which is closest to point coverage member X is replaced by $\langle A, X' \rangle$ and $\langle X', B \rangle$. This effectively approximates X as a new member of the line coverage.

Other forests store barrier data in an Oracle database system called the *Route Management System* (RMS). Each entry in the RMS specifies with which road a barrier is associated, in addition to where on that road the barrier is located. Arc/Info's *dynamic segmentation* models linear features using *routes* and *events*. A route represents a non-circular series of connected arcs, along with measures which can be used to calculate distance. Events are locations of interest along a route (e.g. barriers, bridges, signs, etc.). Distance measures can also be used to identify the location of events along a route (ESRI, 1995B). Dynamic segmentation is a process based on events and routes which allows arcs which contain specified events (e.g. barriers) to be identified along with where on the route (i.e. which arc) the event is located. Using dynamic segmentation, barriers in the RMS can be co-registered onto the associated road network as nodes.

Although dynamic segmentation could have definite applicability to road closure analysis in the future, its use needs to be augmented with point representation of barriers to produce an analysis tool capable of allowing users to determine the effects of new barriers or modified closure schedules on existing barriers. Currently, no forests have an RMS

mature enough to reliably represent the existing set of barriers, and the RMS that are available are not generally accessible outside the Forest Service.

The work here focuses on simple point/line coverages. Barriers and access points are modeled as line coverage nodes. Each line coverage has a *node attribute table* (NAT) which is a tabular datafile containing one *row* of data for each node in the line coverage. Barrier nodes are identified via the road network coverage's NAT by utilizing a "Node Class" column on the NAT, setting that column to a specified "barrier value" for each row in the table which represents a barrier node. Each barrier's BAD is stored in the "Barred Access Dates" column on that barrier's row. Access points are modeled as nodes much like barriers are. The "Node Class" column in the NAT is set to an "access point value" for each row corresponding to an access point. Each access point's INAD is stored in the "Initial Not Accessible Dates" column on that access point's row.

From an Arc/Info perspective, the objective of this effort is to take a road coverage with the associated sets of access points and barriers (both represented as point coverages which are then coregistered onto the corresponding line coverage as nodes), and determine which dates each road coverage arc is accessible.

2.4 Analysis Related Work - Graph Algorithms and GIS Functions

A well known classical graph algorithm that can be used to solve certain types of road closure analysis problems is Dijkstra's algorithm (Kumar, 1994). This algorithm takes as

input a graph $G = (N,A)$, then finds the shortest paths from a node $n \in N$ to all other nodes in N . It is assumed that if a “shortest path” is found from a node n to another node $n' \in N$, then it is possible for vehicle traffic to follow that path from $n \rightarrow n'$, thus n' is accessible from n .

Environmental Systems Research Institute (ESRI) provides functionality within Arc/Info which can also perform a type of road closure analysis. TRACE is a command which “Creates a selected set of arcs and nodes that flow into and/or out of an *origin node(s)*”. (ESRI, 1995B) An origin node (which corresponds to an access point node) is the node that TRACE determines which arcs flow into and out of. It is assumed that there exists for any arc or node which is found to flow into or out of an origin node a path between that arc or node and the origin node. The path follows other nodes and arcs which flow into or out of the origin node.

In using the TRACE command, the portion of the network coverage to be analyzed is specified by selecting the set of arcs and nodes in the coverage on which TRACE operates. Prior to using TRACE, access point nodes need to be specified. After TRACE runs, there exists a path from at least one access point node to each selected arc and node which does not pass through any unselected nodes or arcs.

TRACE allows the user to determine which portions of the coverage are analyzed by selecting the nodes and arcs to be included in the analysis. To accomplish the road closure analysis, all arcs and nodes in the road coverage are selected. The nodes representing closed barriers are then deselected, which prevents TRACE from considering paths that

include that barrier node when tracing the coverage. Since roads are assumed to be undirected (at least outside cities they are) TRACE needs to be executed specifying a direction of BOTH, allowing TRACE to ignore arc direction of flow. When TRACE runs, it identifies all the selected nodes and arcs for which a path exists to an access point node which does not pass through a unselected barrier node. As a result of not selecting nodes, arcs and nodes behind those unselected barrier nodes will not be traced unless there is another path from an access point node to those arcs and nodes which does not pass through any unselected barrier nodes.

According to ESRI, TRACE is implemented using Dijkstra's Algorithm to compute a "single source shortest distance" on a graph. That is, given an access point x , the algorithm identifies the shortest path to all other nodes reachable from x . The algorithm constructs a set S of vertices whose shortest distance from the source is known. At each step, a node n whose distance from the source is shortest of the remaining nodes is identified. That node n is added to S . Since all edges have nonnegative costs, the path from the source to n passes only through nodes already in S . Therefore it is only necessary to record for each node n the shortest distance from the source to n along a path that passes only through nodes of S . Aho (1974) describes an implementation of Dijkstra's algorithm using as input a directed graph $G = (N, A)$, a source $n_0 \in N$, and a function l from edges to nonnegative reals. For all n_i and n_j in N where $n_i \neq n_j$, $l(n_i, n_j)$ is $+\infty$ if (n_i, n_j) is not an edge; $l(n_i, n_j) = 0$. As output the algorithm produces for each $n \in N$, the minimum over all paths P from n_0 to n of the sum of the labels of the edges (i.e. arcs) of P . To do so, the algorithm constructs a set $S \subseteq N$ such that the shortest path

from the source to each node n in S lies wholly in S . The array $D[n]$ contains the cost of the current shortest path from n_0 to n passing only through nodes of S . A pseudo-code description of the implementation appears in Algorithm 2.1.

SINGLE-SOURCE-SHORTEST-PATH

```

Begin
     $S \leftarrow \{n_0\}$ 
     $D[n_0] \leftarrow 0$ 
    for each  $n$  in  $N - \{n_0\}$  do
         $D[n] \leftarrow l(n_0)$ 
    while  $S \neq N$  do
        begin
            choose a node  $w$  in  $N - S$  such that  $D[w]$  is a minimum
            add  $w$  to  $S$ 
            for each  $n$  in  $N - S$  do
                 $D[n] \leftarrow \text{MIN}(D[n], D[w] + l(w, n))$ 
        end
    End

```

Algorithm 2.1. Single-source shortest path (Dijkstra's algorithm).

Given that ESRI's TRACE command uses this algorithm, the assumption that all barriers are either open or closed represents a major deficiency. In reality, many barriers are gates that are closed on a seasonal basis. Whether access is allowed past a gate is date dependent. The same problem exists with access points. Some access points correspond to locations on roads that are always accessible, but this is not always the case. As stated previously, an access point may correspond to a point on a road that is known to only be

accessible on a seasonal basis. To determine the actual restrictions on segments of the road network utilizing the TRACE command, it is necessary to run TRACE for every possible nonoverlapping date range and vehicle type combination. The set of possible nonoverlapping date ranges can be determined by looking at every date specified in the schedule of closures applicable to the road network. Each date found on the schedule of closures is the beginning of one range and the end of another date range such that no data exists on the schedule of closures which lie between any given date and either date which that date forms a date range. For example if a schedule of closures included February 15, June 30, August 15 and October 1 the nonoverlapping date ranges would be February 15 to June 30, June 30 to August 15, August 15 to October 1 and October 1 to February 15. This results in four date ranges.

In addition to not handling the seasonal nature of gates, there is also the issue of whether utilizing Dijkstra's Algorithm (as TRACE does) is the best way to determine which arcs are connected to the access points. Aho (1974) states that "if we only wish to know to which (nodes) there exists a path from the source, the problem is trivial and can be solved by a number of algorithms which operate in $O(e)$ on an e -edge (arc) graph." He goes on to state that Dijkstra's Algorithm is $O(n^2)$, not $O(e)$.

Chapter Three describes a graph algorithm that more efficiently performs the analysis than Arc/Info's TRACE function or any version of Dijkstra's algorithm. The algorithm accomplishes this by handling the seasonal nature of barriers and using a design which improves on the time complexity of $O(n^2)$ for Dijkstra's Algorithm.

Chapter 3

A GRAPH THEORETIC ALGORITHM FOR ROAD CLOSURE ANALYSIS

The graph theoretic solution developed for this project utilizes two phases to solve the road closure analysis problem. The first phase groups together graph members which can be reached from each other without passing through a barrier. This phase is based on a classical “connected components” graph algorithm which takes into account that some gates are closed on a seasonal basis. The second phase of the solution calculates the NAD attributes of each connected set of graph members based on access point INAD attribute values and barrier BAD attribute values.

3.1 A Classical “Connected Components” Graph Algorithm

Determining which portions of the road network have accessibility restrictions imposed by barriers is accomplished with a *connected components algorithm* (Cormen, 1990). A classical connected components algorithm identifies groups of nodes in a graph which are connected by arcs. Figure 3.1 shows an example of a graph with three groups of connected components {1,2,3}, {4,5,6,7} and {8}.

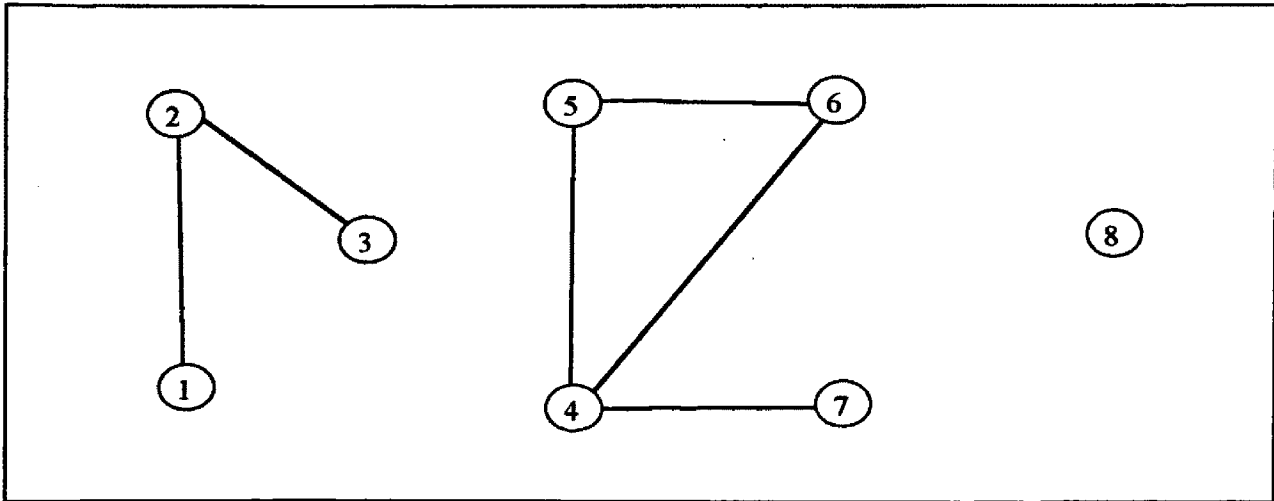


Figure 3.1 - Connected Graph Components

This section describes a classic connected components algorithm from which an algorithm capable of handling gates closed on a seasonal basis is derived. The algorithm takes a standard graph as input. Each node on the graph is initially put into a *connected component set* by itself. The algorithm examines each arc in the graph. As each arc is processed, the nodes to which it is connected via an arc are identified. If both source and target nodes are contained in different connected component sets, the two sets are merged. If both nodes are already in the same connected component set, no further action is required for that arc. When the algorithm completes, each node in any connected component set may be reached from any other node in the same connected component set by following a path of arcs in the graph.

The classic `CONNECTED_COMPONENTS` algorithm described by Cormen utilizes a disjoint-set data structure which maintains a collection $T = \{S_1, S_2, \dots, S_k\}$ of disjoint dynamic sets. Each set is identified by a representative, which is some member of the set.

Letting x denote one of these sets, the following operations are utilized by the `CONNECTED_COMPONENTS` procedure shown as Algorithm 3.1.

- `MAKE_SET(x)` creates a new set whose only member (and thus representative) is pointed to by x . Since the sets in T are disjoint, we require that x not be in a set.
- `UNION(x, y)` unites the dynamic sets that contain x and y , S_x and S_y respectively, into a new set that is the union of these two sets. The two sets are assumed to be disjoint prior to the operation. The sets S_x and S_y are destroyed following the union, removing them from the collection T . The representative of the resulting set is some member of $S_x \cup S_y$.
- `FIND_SET(x)` returns the representative of the set containing x .

The procedure `CONNECTED_COMPONENTS` uses these operations to group together components of the graph which are connected. It takes as input a graph G . As output it produces a set of connected components, in the form of the graph nodes in the connected components. Once `CONNECTED_COMPONENTS` has been run with graph G as input, `FIND_SET` will return the same value for nodes that are in the same set of connected components.

```
CONNECTED_COMPONENTS( $G$ )
  for each node  $n \in N[G]$ 
    do MAKE_SET( $N$ )
  for each arc  $(u, v) \in E[G]$ 
    do if FIND_SET( $u$ )  $\neq$  FIND_SET( $v$ )
       then UNION( $u, v$ )
```

Algorithm 3.1 - CONNECTED_COMPONENTS

CONNECTED_COMPONENTS puts nodes from G into a disjoint-set data structure, grouping connected nodes into the same set. A simple way to implement a disjoint-set data structure is to represent each set with a linked list. The first object in each linked list serves as its set's representative. Each object in the linked list contains a set member, a pointer to the object containing the next set member and a pointer back to the representative. Within each linked list, the objects may appear in any order (subject to our assumption that the first object in each list is the representative).

Using this linked list representation, both MAKE-SET and FIND_SET are easy, requiring $O(1)$ time (Cormen, 1990). To carry out MAKE_SET(x), create a new linked list whose only object is x . For FIND_SET(x), return the pointer from x back to the representative.

3.2 A Derivative Algorithm That Can Handle Seasonal Barriers

The classical `CONNECTED_COMPONENTS` algorithm described by Cormen (1990) does not explicitly indicate which arcs connect the nodes in a connected component set. In addition, it is difficult to adapt to handle barriers, especially ones that are closed seasonally. As mentioned previously, a node representing the barrier is attributed to indicate when traffic access is restricted. The classic algorithm assumes that each node will end up in one and only one connected component set, and does not check the attributes on barrier nodes to adjust connectivity. Although a barrier is reachable from the two connected component sets on either side of the barrier, the barrier is not a member of either connected set.

To overcome these deficiencies, a derivative algorithm `CONDITIONAL_CONNECTED_COMPONENTS`, shown as Algorithm 3.2, keeps the node representing the barrier in a connected set by itself. This connected set is labeled as a barrier, is attributed with any restrictions the barrier imposes on traffic access, and is linked via *barrier links* to the connected sets containing the nodes on either sides of the barrier. These barrier links allow the derivative algorithm to identify the impact of the barriers restrictions on access from the neighboring connected sets. `CONDITIONAL_CONNECTED_COMPONENTS` also utilizes a construct to group connected graph members (both arcs and nodes) in a special structure referred to as a *connected set*. In order to attribute arcs with the access restrictions identified for the

same data structure. A connected set originates in the same way as a connected component set, with each node in the graph being placed in a connected set by itself.

CONDITIONAL-CONNECTED-COMPONENTS processes arcs one by one. If both nodes are in the same connected set, the arc is placed in that connected set. If neither node is a barrier and the two nodes are in different connected sets, the connected sets are merged and the arc is placed in the merged connected set. If neither node is a barrier, the arc joining them is put in the connected set with its end points. If one of the nodes represents a barrier, the arc is placed in the connected set which contains the arc's nonbarrier node and a barrier link is also established between the two connected sets. If both nodes associated with an arc are barriers, a pseudo connected set is created which contains no nodes, but has the arc and links to the connected sets containing the two nodes.

The **CONDITIONAL-CONNECTED-COMPONENTS** algorithm requires a slightly different data structure than the classic **CONNECTED-COMPONENTS** algorithm. **CONDITIONAL-CONNECTED-COMPONENTS** requires a data structure of sets where each set consists of a list of connected nodes and a list of associated arcs. Each connected set contains a list of references to the connected sets containing neighboring barrier nodes. Connected sets contain a **INAD** and **NAD** attributes. Calculation of values for these two attributes is addressed in Section 3.3. The derivative data structure supports the following operations.

- **MAKE_SET(x)** creates a new connected set whose only node member is node x , and which has no member arcs. If x represents a barrier, the connected set is attributed accordingly.
- **FIND_SET(x)** returns the connected set which contains node x .
- **UNION(x, y)** unites the dynamic sets that contain non-barrier nodes x and y , S_x and S_y respectively, into a new set that is the union of these two sets. S_x and S_y are assumed to be disjoint prior to the operation. The arc $\langle x, y \rangle$ is added to the new set. The representative of the resulting set is some member of $S_x \cup S_y$. The original sets S_x and S_y are destroyed following the union, removing them from the collection of connected sets.
- **MAKE_LINK(x, y)** establishes a barrier link between a non-barrier connected set x and a barrier connected set y . The arc $\langle x, y \rangle$ is added to the connected set containing node x .

As shown in Algorithm 3.2, the procedure **CONDITIONAL_CONNECTED_COMPONENTS** uses the preceding disjoint set operations to group together connected graph components with barriers. Once **CONDITIONAL_CONNECTED_COMPONENTS** produces its results, it is possible to determine whether two nodes are on the same connected set by comparing the results of running **FIND_SET** for both nodes. If the same connected set is returned by both calls, then the nodes reside on the same connected set; otherwise they are in different connected

sets. The set of nodes from graph G is denoted $N[G]$; and the set of arcs is denoted by $A[G]$. In solving the road closure analysis problem, once the procedure **CONDITIONAL-CONNECTED-COMPONENTS** produces its results, the NAD attributes for each connected set can be determined, as discussed in Section 3.3. The NAD attribute for each set can be recorded with each of its member arcs once the attribute is determined for the connected set.

CONDITIONAL-CONNECTED-COMPONENTS

Begin

```
for each node n in N[G]
    MAKE_SET(n)
for each arc(u,v) in A[G]
    set_u := FIND_SET(u)
    set_v := FIND_SET(v)
    if (set_u = set_v)
        add arc(u,v) to set_u's arc list
    else if (neither set_u or set_v are barrier sets)
        add arc(u,v) to set_u
        UNION(set_u, set_v)
    else if (set_u and set_v are barrier sets)
        create pseudo set
        MAKE_LINK(pseudo set, set_u)
        add arc(u,v) to the pseudo set
    else if (set_u is a barrier set)
        MAKE_LINK(set_v, set_u)
        put arc(u,v) to set_v
    else if (set_v is a barrier set)
        MAKE_LINK(set_u, set_v)
        put arc(u,v) to set_u
    end if
end for
```

End

Algorithm 3.2. CONDITIONAL-CONNECTED-COMPONENTS

An illustration of the graph representation of the connected sets and barriers produced by `CONDITIONAL_CONNECTED_COMPONENTS` when the road network represented by the graph in Figure 2.3 is supplied as input is shown in Figure 3.2.

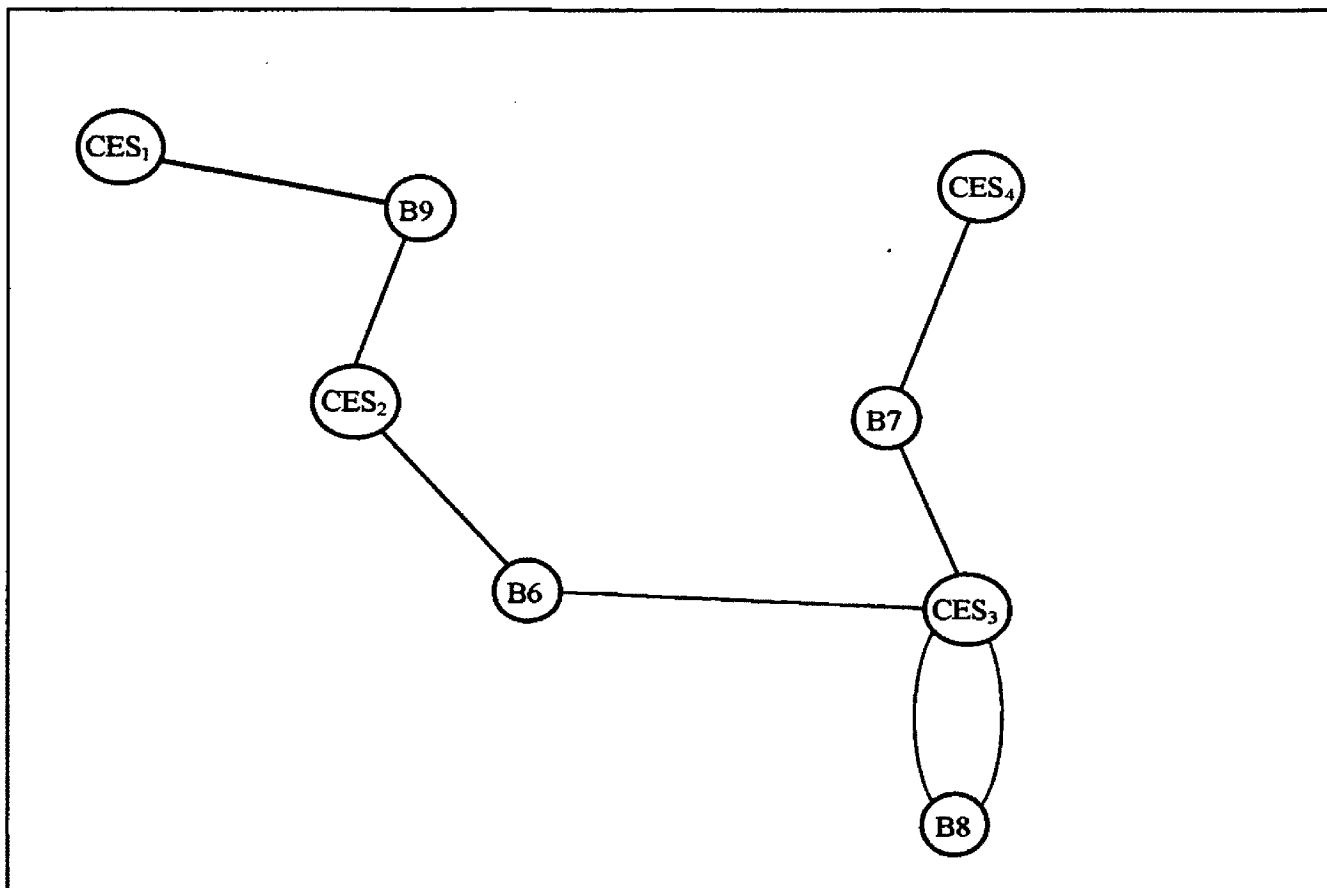


Figure 3.2 - Condensed Graph of Connected Sets and Barriers

3.3 NAD Attribute Identification

NAD attribute determination is performed using an iterative breadth first search. The NAD attributes that are calculated for any connected set apply to each of its member arcs and nodes. The first step in calculating NAD values is to identify the connected sets which

have known INAD values, based on the member access points they contain which have an INAD attribute. This is determined for each connected set by taking the intersection of each member access point's INAD attribute. It can be assumed that any connected set that has a known INAD value also has a known NAD value which equals the INAD value. Connected sets that have an unknown INAD value are assumed to have a NAD value of {1/1 .. 12/31}.

Next an iterative breadth first search is done of the collection of connected sets. Each breadth first search starts by placing each connected set which is known to have *unlimited access* onto a queue and setting its NAD attributes to null. Unlimited access is defined as allowing vehicle access to that point at any time. Unlimited access is represented by an INAD or known NAD attribute with an empty value. As each set on the queue is processed, all neighboring sets which have not been previously placed on the queue in the current search iteration are pushed onto the queue. As each set is pulled off the queue, its NAD attribute is updated by taking the intersection of its current NAD (if known), its INAD (if it has one) and the effective restrictions imposed by each neighboring barrier. The effective restriction imposed by a neighboring barrier can be determined only if the NAD of the connected set on the other side of the barrier is known. If that NAD is known, the effective restriction of the barrier is union of the barrier node's BAD attribute and the NAD attribute of the connected set lying on the other side of that barrier from the connected set currently being processed. This breadth first search is repeated until the NAD attribute for each connected set are the same as they were after the previous breadth first search. This NAD attribute determination algorithm, referred to as

NAD_ATTRIBUTE_DETERMINATION is shown in Algorithm 3.3. This algorithm takes as input a disjoint set of connected sets (denoted as DS) which is produced by CONDITIONAL_CONNECTED_COMPONENTS.

```

NAD_ATTRIBUTE_DETERMINATION ( $DS$ )
   $NAD\_CHANGED := True$ 
  while  $NAD\_CHANGED$ 
     $NAD\_CHANGED := False$ 
    clear the queue
    put connected sets that have  $NAD = \{\}$  on the queue
    while the queue is not empty
       $C := queue.pop$ 
      for all connected sets  $N$  which are neighbors of  $C$ 
         $B$  is the barrier node between  $C$  and  $N$ 
        if  $N$  has not been pushed onto the queue this iteration
          push  $N$  onto the queue
        end if
        if  $N.NAD$  is known
           $C.NAD = C.NAD \cap (B.BAD \cup N.NAD)$ 
           $NAD\_CHANGED := True$ 
        end if
      end for
    end while
  end while

```

Algorithm 3.3. NAD Attribute Determination Algorithm

To illustrate how the NAD_ATTRIBUTE_DETERMINATION algorithm works, consider a graph consisting of connected sets and barrier nodes which is produced by

CONDITIONAL_CONNECTED_COMPONENTS. The graph and element values shown in Figure 3.2 represents a road network similar to the road network shown in Figure 2.3. The example that follows only shows NAD values consisting of a single set of dates; NAD attributes can handle multiple vehicle classes by maintaining a separate set of dates for each vehicle class. While this example does not prove the correctness of using NAD-ATTRIBUTE_DETERMINATION to determine NAD values, it does show that this approach does seem to work in practice.

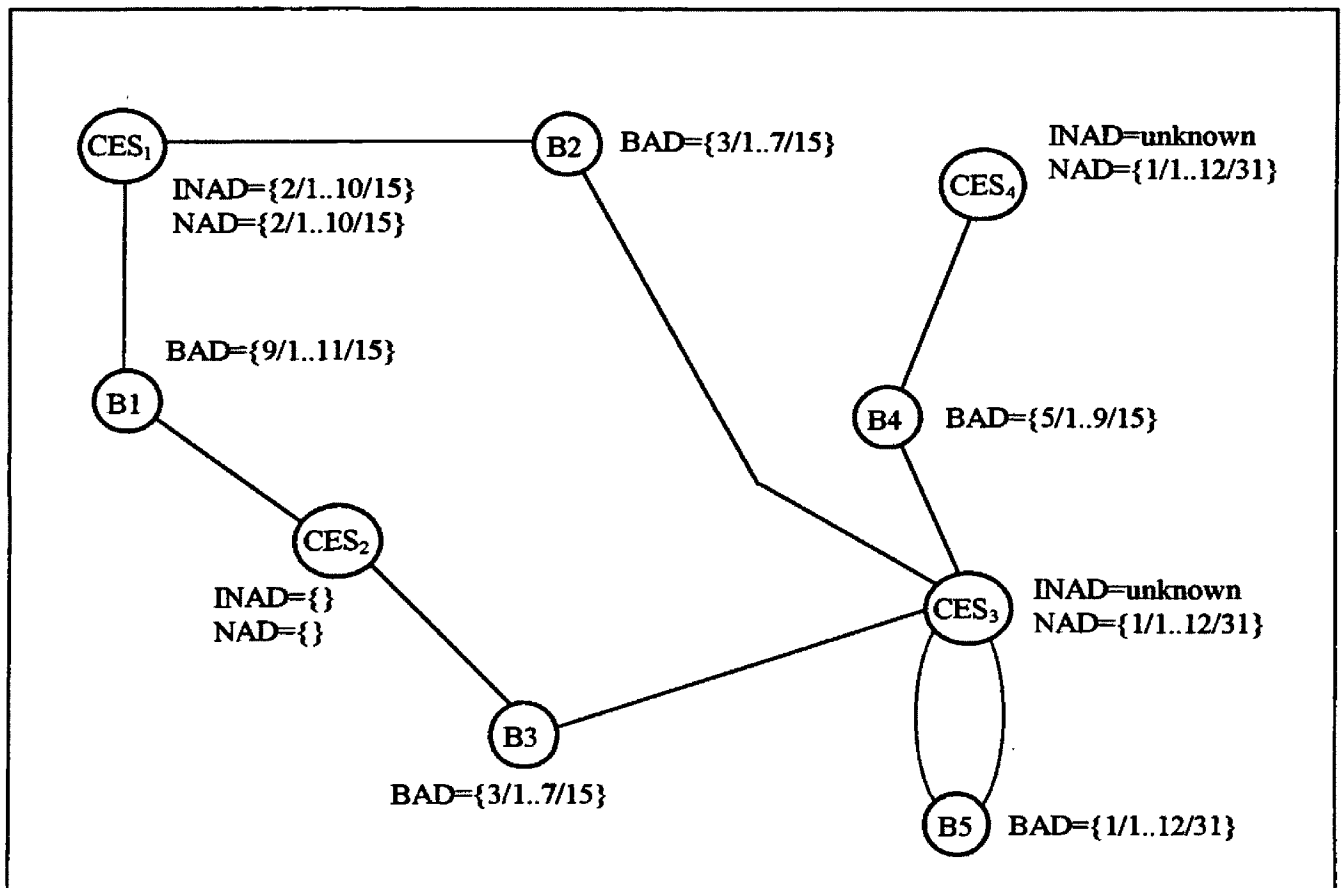


Figure 3.3 Condensed Graph Prior To NAD Attribute Determination

On the first iteration of the breadth first search, CES_2 has a null NAD value and is placed on the queue. CES_2 is then popped off the queue. CES_2 has two neighboring connected element sets, CES_1 and CES_3 , both of which are pushed onto the queue now, since neither connected set has been placed on the queue previously. $CES_2.NAD$ is then updated to reflect the effective restrictions imposed by the neighboring connected sets as follows:

First CES_1 :

$$\begin{aligned}
 CES_2.NAD &:= CES_2.NAD \cap (B1.BAD \cup CES_1.NAD) \\
 &:= \{\} \cap (\{\text{Sept 1, ..., Nov 15}\} \cup \{\text{Feb 1, ..., Oct 15}\}) \\
 &:= \{\}
 \end{aligned}$$

Then CES_3 :

$$\begin{aligned}
 CES_2.NAD &:= CES_2.NAD \cap (B3.BAD \cup CES_3.NAD) \\
 &:= \{\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\text{Jan 1, ..., Dec 31}\}) \\
 &:= \{\}
 \end{aligned}$$

Next CES_1 is pulled off the queue. Its two neighbors CES_2 and CES_3 have both been previously placed on the queue, so there is no need to do so again. $CES_1.NAD$ is updated to reflect the effective restrictions imposed by the neighboring connected sets as follows:

First CES_2 :

$$\begin{aligned}
 CES_1.NAD &:= CES_1.NAD \cap (B1.BAD \cup CES_2.NAD) \\
 &:= \{\text{Feb 1, ..., Oct 15}\} \cap (\{\text{Sept 1, ..., Nov 15}\} \cup \{\}) \\
 &:= \{\text{Sept 1, ..., Oct 15}\}
 \end{aligned}$$

Then CES_3 :

$$\begin{aligned}
CES_1.NAD &:= CES_1.NAD \cap (B2.BAD \cup CES_3.NAD) \\
&:= \{\text{Sep 1, ..., Oct 15}\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\text{Jan 1, ..., Dec 31}\}) \\
&:= \{\text{Sep 1, ..., Oct 15}\}
\end{aligned}$$

Next CES_3 is pulled off the queue. Of its three neighbors CES_1 , CES_2 and CES_4 , only CES_4 needs to be placed on the queue. $CES_3.NAD$ is updated to reflect the effective restrictions imposed by the neighboring connected sets. Incidentally, the barrier links to $B5$ are ignored because CES_3 lies on both sides of that barrier. The update of $CES_3.NAD$ is as follows:

First CES_1 :

$$\begin{aligned}
CES_3.NAD &:= CES_3.NAD \cap (B2.BAD \cup CES_1.NAD) \\
&:= \{\text{Jan 1, ..., Dec 31}\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\text{Sep 1, ..., Oct 15}\}) \\
&:= \{\text{Mar 1, ..., Jul 15, Sep 1, ..., Oct 15}\}
\end{aligned}$$

Next CES_2 :

$$\begin{aligned}
CES_3.NAD &:= CES_3.NAD \cap (B3.BAD \cup CES_2.NAD) \\
&:= \{\text{Mar 1, ..., Jul 15, Sep 1, ..., Oct 15}\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\}) \\
&:= \{\text{Mar 1, ..., Jul 15}\}
\end{aligned}$$

Then CES_4 :

$$\begin{aligned}
CES_3.NAD &:= CES_3.NAD \cap (B4.BAD \cup CES_4.NAD) \\
&:= \{\text{Mar 1, ..., Jul 15}\} \cap (\{\text{May 1, ..., Sep 15}\} \cup \{\text{Jan 1, ..., Dec 31}\}) \\
&:= \{\text{Mar 1, ..., Jul 15}\}
\end{aligned}$$

Last of all, CES_4 is pulled off the queue. Its neighbor CES_3 has been previously placed on the queue, so there is no need to do so again. $CES_4.NAD$ is updated to reflect the effective restrictions imposed by the neighboring connected set CES_3 as follows:

$$\begin{aligned}
 CES_4.NAD &:= CES_4.NAD \cap (B4.BAD \cup CES_3.NAD) \\
 &:= \{\text{Jan 1, ..., Dec 31}\} \cap (\{\text{May 1, ..., Sep 15}\} \cup \{\text{Mar 1, ..., Jul 15}\}) \\
 &:= \{\text{Mar 1, ..., Sep 15}\}
 \end{aligned}$$

Figure 3.3 shows the condensed graph along with the corresponding attribute values after the first pass through the NAD attribute determination breadth first search.

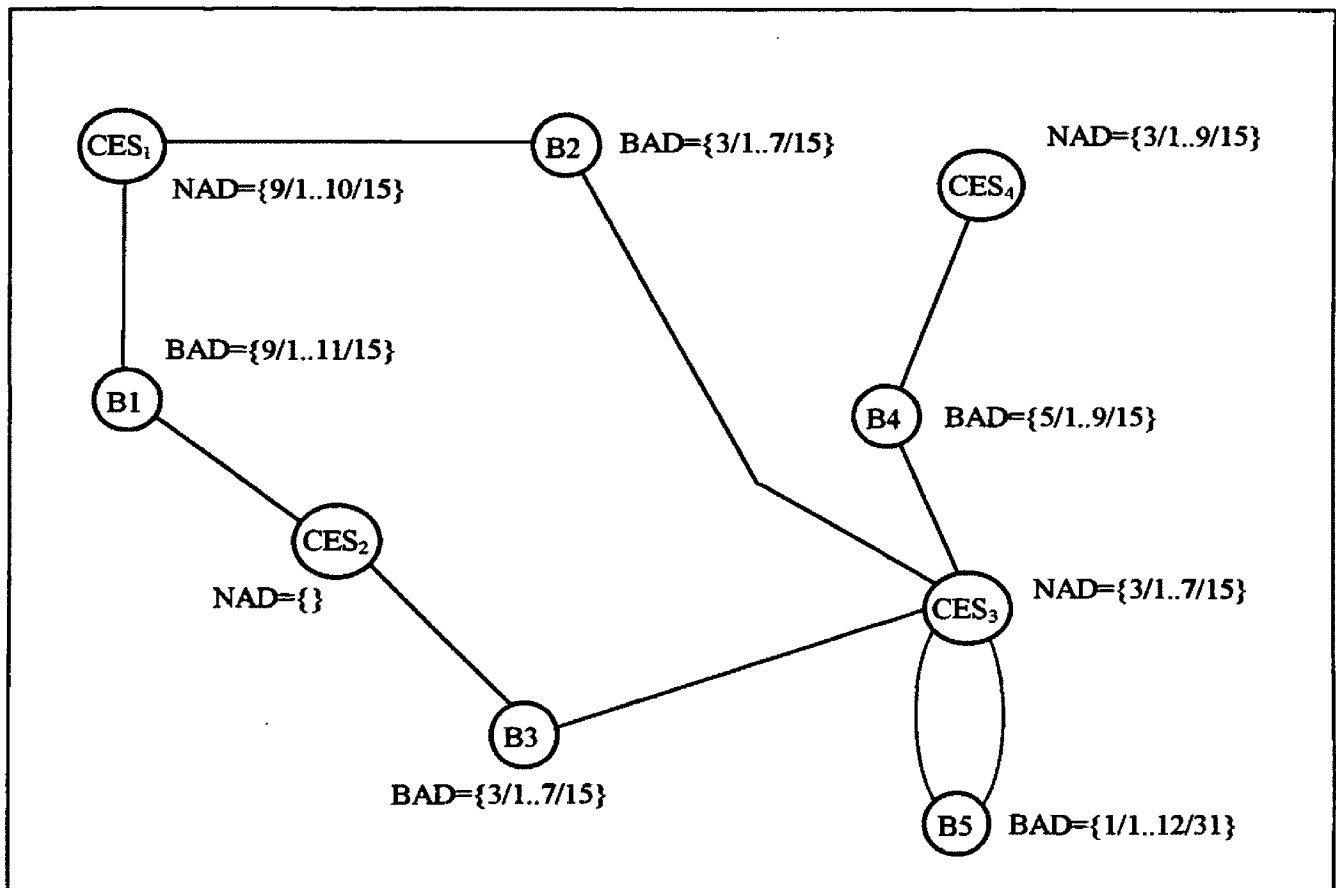


Figure 3.4 Condensed Graph After First Breadth First Search

During the first iteration of the breadth first search, the NAD values of three of the sets (i.e. CES_1 , CES_3 and CES_4) were updated, necessitating the need for at least one more pass through the breadth first search. On the second iteration of the breadth first search, CES_2 is still the only set with a null NAD value and is placed on the queue. CES_2 is then popped off the queue. CES_2 has two neighboring connected element sets, CES_1 and CES_3 , both of which are pushed onto the queue. $CES_2.NAD$ is then updated to reflect the effective restrictions imposed by the neighboring connected sets as follows:

First CES_1 :

$$\begin{aligned}
 CES_2.NAD &:= CES_2.NAD \cap (B1.BAD \cup CES_1.NAD) \\
 &:= \{\} \cap (\{\text{Sept 1, ..., Nov 15}\} \cup \{\text{Sep 1, ..., Oct 15}\}) \\
 &:= \{\}
 \end{aligned}$$

Then CES_3 :

$$\begin{aligned}
 CES_2.NAD &:= CES_2.NAD \cap (B3.BAD \cup CES_3.NAD) \\
 &:= \{\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\text{Mar 1, ..., Jul 31}\}) \\
 &:= \{\}
 \end{aligned}$$

Next CES_1 is pulled off the queue. Its two neighbors CES_2 and CES_3 have both been previously placed on the queue, so there is no need to do so again. $CES_1.NAD$ is updated to reflect the effective restrictions imposed by the neighboring connected sets as follows:

First CES_2 :

$$\begin{aligned}
 CES_1.NAD &:= CES_1.NAD \cap (B1.BAD \cup CES_2.NAD) \\
 &:= \{\text{Sep 1, ..., Oct 15}\} \cap (\{\text{Sept 1, ..., Nov 15}\} \cup \{\})
 \end{aligned}$$

$$:= \{\text{Sept 1, ..., Oct 15}\}$$

Then CES_3 :

$$\begin{aligned} CES_1.NAD &:= CES_1.NAD \cap (B2.BAD \cup CES_3.NAD) \\ &:= \{\text{Sep 1, ..., Oct 15}\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\text{Mar 1, ..., Jul 15}\}) \\ &:= \{\} \end{aligned}$$

Next CES_3 is pulled off the queue. Of its three neighbors (i.e. CES_1 , CES_2 and CES_4), only CES_4 needs to be placed on the queue. $CES_3.NAD$ is updated to reflect the effective restrictions imposed by the neighboring connected sets as follows:

First CES_1 :

$$\begin{aligned} CES_3.NAD &:= CES_3.NAD \cap (B2.BAD \cup CES_1.NAD) \\ &:= \{\text{Mar, ..., Jul 15}\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\}) \\ &:= \{\text{Mar 1, ..., Jul 15}\} \end{aligned}$$

Next CES_2 :

$$\begin{aligned} CES_3.NAD &:= CES_3.NAD \cap (B3.BAD \cup CES_2.NAD) \\ &:= \{\text{Mar 1, ..., Jul 15}\} \cap (\{\text{Mar 1, ..., Jul 15}\} \cup \{\}) \\ &:= \{\text{Mar 1, ..., Jul 15}\} \end{aligned}$$

Then CES_4 :

$$\begin{aligned} CES_3.NAD &:= CES_3.NAD \cap (B4.BAD \cup CES_4.NAD) \\ &:= \{\text{Mar 1, ..., Jul 15}\} \cap (\{\text{May 1, ..., Sep 15}\} \cup \{\text{Mar 1, ..., Sep 15}\}) \\ &:= \{\text{Mar 1, ..., Jul 15}\} \end{aligned}$$

Last of all, CES_4 is pulled off the queue. Its neighbor CES_3 has been previously placed on the queue, so there is no need to do so again. $CES_4.NAD$ is updated to reflect the effective restrictions imposed by the neighboring connected set CES_3 as follows:

$$\begin{aligned}
 CES_4.NAD &:= CES_4.NAD \cap (B_4.BAD \cup CES_3.NAD) \\
 &:= \{\text{Mar 1, ..., Sep 15}\} \cap (\{\text{May 1, ..., Sep 15}\} \cup \{\text{Mar 1, ..., Jul 15}\}) \\
 &:= \{\text{Mar 1, ..., Sep 15}\}
 \end{aligned}$$

Figure 3.4 shows the condensed graph along with the corresponding attribute values after the second pass through the NAD attribute determination breadth first search.

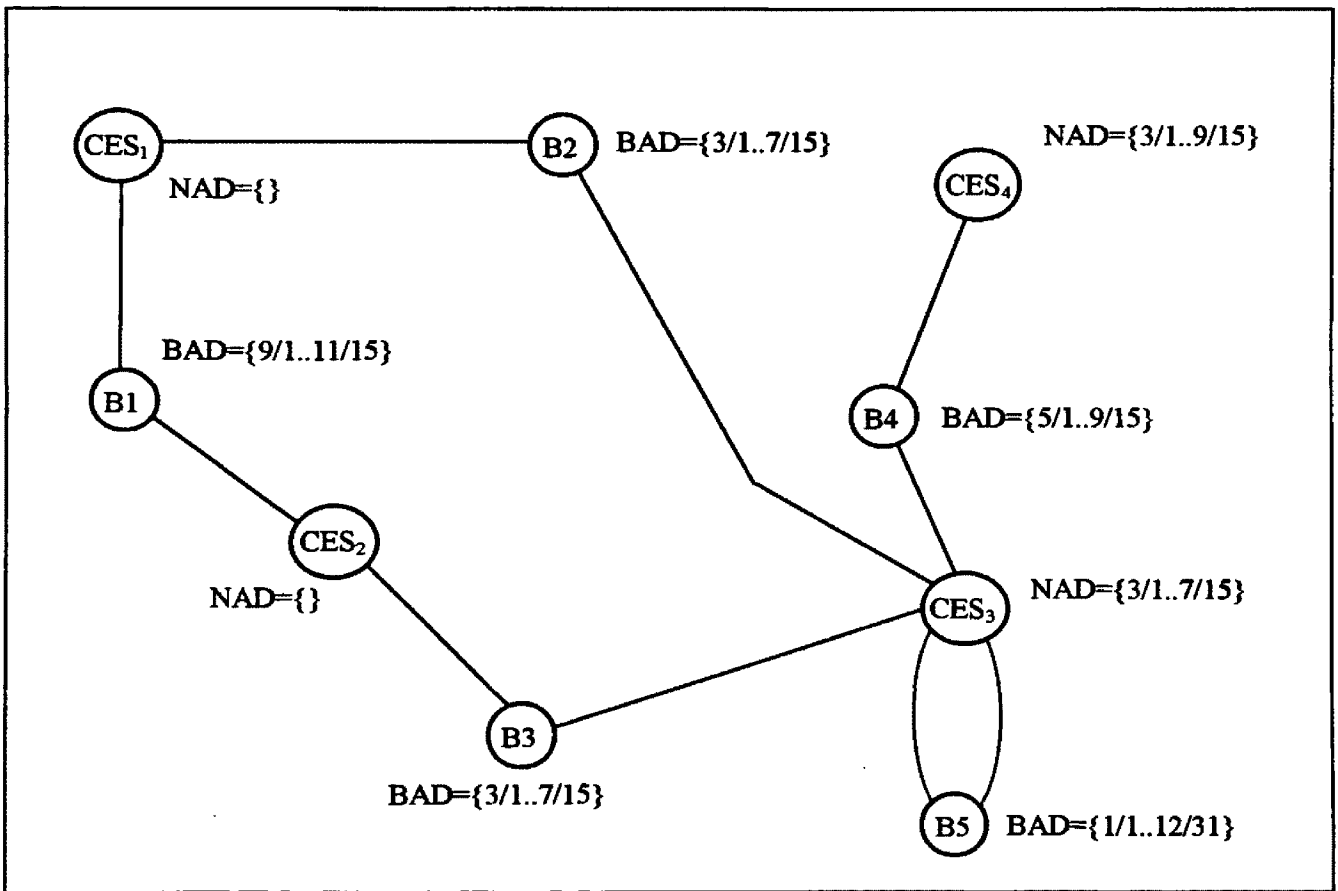


Figure 3.5 Condensed Graph After Second Breadth First Search

Since CES_i had its NAD updated during the second pass through the breadth first search, a third pass is necessary. Upon completion of the third pass, none of the NAD values have been updated, eliminating the need for a fourth pass through the breadth first search. Since the details of the third pass are the same as the second pass with one exception (i.e. the initial value of $CES_j.NAD$) the detail of the third pass are omitted.

3.4 Analysis of the `CONDITIONAL_CONNECTED_COMPONENTS` Algorithm

The performance analysis discussed in this section only pertains to the `CONDITIONAL_CONNECTED_COMPONENTS` algorithm discussed in Section 3.2. Due to the complexity of the analysis of the breadth first search algorithm which calculates the NAD attributes as discussed in Section 3.3, that analysis is not discussed in this document.

Even though the data structure and associated operations utilized by `CONDITIONAL_CONNECTED_COMPONENTS` have a higher cost than those used by either `CONNECTED_COMPONENTS` or Dijkstra's algorithm (i.e. TRACE) due to the more complex data structure used by `CONDITIONAL_CONNECTED_COMPONENTS`, the cost of performing the road closure analysis is lower for `CONDITIONAL_CONNECTED_COMPONENTS`. The cost savings of `CONDITIONAL_CONNECTED_COMPONENTS` results from being able to perform the analysis for multiple nonoverlapping date ranges and vehicle classes at the same time,

while `CONNECTED_COMPONENTS` must process each nonoverlapping date range separately for each vehicle class.

Cormen (1990) describes an optional representation of disjoint sets which represents sets of nodes as rooted trees, with each node containing one node and each tree representing one set of nodes. In these *disjoint-set forests*, each member points to its parent. The root of each tree contains the representative and is its own parent. Although the straightforward algorithms that use this representation are no faster than ones that use the linked-list representation, by introducing two heuristics, “union by rank” and “path compression”, the asymptotically fastest disjoint-set data structure known can be achieved.

Disjoint-set forests utilize three primary operations. A `MAKE_SET` operation simply creates a tree with just one node. `FIND_SET` chases parent pointers until the root of the tree is located. The nodes visited on this path toward the root constitute the *find path*. A `UNION` operation causes the root of one tree to point to the root of the other.

The first heuristic, *union by rank*, which can improve on the linked-list implementation makes the root of the tree with fewer nodes point to the root of the tree with more nodes. Rather than explicitly keeping track of the size of the subtree rooted at each node, a rank is maintained that approximates the logarithm of the subtree size and is also an upper bound on the height of the node. In union by rank, the root with smaller rank is made to point to the root with larger rank during a `UNION` operation.

The second heuristic, *path compression*, assists in improving the linked-list implementation during FIND-SET operations by making each node on the find path point directly to the root. Path compression does not change any ranks.

In analyzing the cost associated with solving the road closure analysis problem with `CONDITIONAL_CONNECTED_COMPONENTS` utilizing a linked list representation for the disjoint node sets, the cost associated with the individual operations it utilizes are influenced by the implementation decision to utilize linked lists for the disjoint sets. The cost associated with the operations can be reduced by utilizing disjoint-set forests data structures in place of the linked list representation of sets used in this implementation.

`MAKE_SET(X)` creates three linked lists, one for nodes, one for arcs and one for barrier links. X is then placed on the node list. This results in a cost of $O(1)$ for `MAKE_SET`.

`FIND_SET(X)` traverses the list of connected sets checking each set's node list for X . The number of node list elements that must be visited to find X is bounded by the number of graph nodes m . The cost associated with `FIND_SET` is $O(m)$.

`MAKE_LINK(X, Y)` starts by finding the sets which contain X and Y . Utilizing `FIND_SET` to find the sets has a cost of $O(m)$ each. A barrier link referring to Y is then added to X 's barrier link list at a cost of $O(1)$. The total cost associated with `MAKE_LINK` is $2 * O(m) + O(1)$ which equal $O(m)$.

`UNION(X, Y)` starts by finding the sets which contain X and Y . Utilizing `FIND_SET` to find the sets has a cost of $O(m)$ each. Next the node, arc and barrier lists in the set containing

Y is appended to the corresponding lists in the set containing X . Each of these appends have a cost of $O(m)$. The set which initially contained Y is deleted at a cost of $O(1)$. The total cost associated with UNION is $5 \cdot O(m) + O(1)$ which equal $O(m)$.

The `CONDITIONAL_CONNECTED_COMPONENTS` procedure consists of two loops. The first loop has m iterations, one for each node in the graph. Each pass through the loop calls `MAKE_SET` which has a cost of $O(1)$. This results in a cost associated with the first loop of $m \cdot O(1)$ which equals $O(m)$. The second `CONDITIONAL_CONNECTED_COMPONENTS` loop has e iterations, one for each edge in the graph. This loop utilizes a variety of operations including UNION, `FIND_SET` and `MAKE_LINK`, but does not contain any loops. All the operations inside the second loop are either $O(1)$ or $O(m)$, resulting in a cumulative cost of $O(m)$ for a single pass through the second loop. The total cost associated with the second loop is $e \cdot O(m)$ which equals $O(em)$. Since $O(em)$ is greater than $O(m)$, the cost associated with `CONDITIONAL_CONNECTED_COMPONENTS` is $O(em)$.

By contrast, Aho (1974) states Dijkstra's algorithm (i.e. `TRACE`) has a cost of $O(m^2)$. However, since it must be run for each nonoverlapping date range and vehicle class combination, the cost is $O(dvm^2)$ where d is the number of nonoverlapping date ranges and v is the number of vehicle classes.

COMPARISON OF SOLUTIONS

Two solutions to the road closure analysis problem have been presented. One solution is an implementation of the Conditionally Connected Component Algorithm from Chapter Three. The other solution utilizes the Arc/Info TRACE command mentioned in Chapter Two. When comparing the performance of the two solutions, both execution time and disk space utilization are issues that need to be examined. This chapter also examines two approaches to visualizing the results of the road closure analysis.

4.1 Road Closure Analysis Solution Comparisons

To compare the two algorithms, I use as benchmarks their performance in the analysis of a road coverage representing the Rocky Mountain Division of the Lewis and Clark National Forest. This coverage consists of approximately 3100 arcs, and represents an “average sized” coverage. Forest Service personnel have indicated an interest in performing additional analyses on study sizes ranging from individual Bear Management Units (some of which are less than 1000 arcs) to whole ecosystems comprising multiple national forests (the road coverage for the Kootenai National Forest alone contains over 21000 arcs). However, the analysis here is limited to that of the Rocky Mountain Division of the Lewis and Clark National Forest.

The execution times compared for each of the solutions include the time needed to take an input road coverage with barriers and access points represented as nodes on the road network coverage, determine which arcs have vehicle access restricted by barrier nodes, and record the NAD attributes of those arcs in the Arc Restriction Table (ART). The ART is an "Info" RDBMS table consisting of an *arc_id* column with a width of 4 bytes, a *vehicle type* column with a width of 6 bytes, a *closure beginning date* column with a width of 4 bytes and a *closure end date* column with a width of 4 bytes. Each row requires a total of 18 bytes.

The solution utilizing the TRACE command requires that the command be run multiple times. As discussed in Chapter Two, it must be run for each nonoverlapping date range and vehicle combination. For each arc that is found to be restricted, a row is placed in the ART for each nonoverlapping date range / vehicle type combination. An arc's NAD attribute comprises the set of rows recorded in the ART for that AAT row. This solution records all the arcs that are restricted for a single date range / vehicle type combination before looking at the next date range / vehicle type combination. As a result, ART is not sorted by *arc_id*. The unsorted order of the ART rows becomes a factor when an application tries to use the ART to determine which arcs are restricted for a given set of dates and vehicle types. If the rows are sorted by *arc_id*, an application can make one pass through the table. All the rows for a given arc are located together. When a row with a different *arc_id* is found, the assumption can be made that all the rows for that arc have been examined and it is possible to determine whether the arc is restricted for the specified criteria. However, if the ART is not ordered by *arc_id*, either the table must be

sorted by `arc_id` (using the Tables module SORT command) or the application must search through the whole table for rows for a particular arc.

Section 3.3 describes how connected sets determined by `CONDITIONAL-CONNECTED-COMPONENTS` can be used to calculate NAD attribute values for the arcs on a road network. This solution places one row in the ART for each contiguous range of dates an arc is closed to a particular vehicle type. This solution places rows in the ART sorted by `arc_id`.

In comparing execution times, both solutions were run on a IBM RS/6000 model 220 (named Tincup) in one of the Computer Science labs at the University of Montana. The solution utilizing TRACE ran in 22:26 minutes, creating an ART containing 59,831 rows requiring 1,076,958 bytes of disk space. The solution utilizing the Conditional Connected Components algorithm ran in 2:48 minutes, creating an ART containing 4710 rows requiring 84,780 bytes of disk space. The improvement in ART disk space utilization obtained by the Conditional Connected Components algorithm results from its ability to identify and merge multiple contiguous date range values for NAD values. The TRACE solution does not have this ability, resulting in more rows and as a consequence, higher disk space usage.

4.2 Analysis Results Visualization Comparisons

Two methods of visualizing the results of the accessibility analyses were investigated. Both approaches allow the user to specify visualization criteria that include a date range and multiple vehicle types. Arcs that have access restricted for one or more specified classes of vehicles for the entire date range are identified and are given a *Restricted Vehicle Classes* (RVC) attribute value which specifies how many of the specified vehicle classes have access restricted on that arc. The RVC is used to assist in color coding the arcs to display how many of the specified vehicle classes have restricted access for the entire date range. Arc/Info's ArcPlot module is used to display the road network coverage, utilizing the RVC attribute to color code the arcs to show how many of the specified vehicle classes have restricted access on the arcs.

ESRI recommends implementing this visualization tool with an AML script using nested cursors. The outer cursor visits each row in the Arc Attribute Table (AAT). The inner cursor visits each row in the ART with the same arc_id as the row currently pointed to by the outer (AAT) cursor. To handle the one-to-many relation between the ART and the AAT, the inner cursor is implemented as a relate cursor. Each row in the AAT that is found to have rows in the ART which indicate that the arc meets restriction date criteria for one or more specified vehicle types has a value reflecting the number of restricted vehicle classes placed in the RVC column in the AAT. The value is used to assist in color coding arcs to display how many of the specified vehicles are restricted on the arc for the specified date range.

Another approach to visualizing the results of the analysis is to simply write a C++ program to read the AAT and ART to accomplish the same objectives as the AML cursor based solution.

Both visualization approaches were implemented and applied to the road coverage representing the Rocky Mountain Division of the Lewis and Clark National Forest. The coverage's ART had been previously created by an implementation of the Conditionally Connected Components Algorithm. The C++ approach was able to read the ART, calculate the RVC attribute values, and display the results in 2:27 minutes. The AML cursor approach was terminated after failing to finish running after 3.5 hours.

INTEGRATION OF GRAPH THEORETIC SOLUTION WITH A GIS

This project focuses on the development of a road network analysis tool based on the **CONDITIONALLY_CONNECTED_COMPONENTS** algorithm. The tool provides a means of specifying access points and barriers, determines the extent of the restrictions imposed on the road network by the associated barriers via an implementation of the algorithm, and provides a visualization mechanism to display the travel restrictions identified by the analysis. The implementation and visualization component help establish the validity and practical performance characteristics of the algorithm. The analysis tool operates in an Arc/Info environment, taking a road coverage as input and allowing the user to specify the location and attributes of barriers and access points. This chapter describes from a technical standpoint how the analysis tool manages the data to be analyzed for travel restrictions. For a functional description of how to operate the tool described in this chapter, refer to the Network Accessibility Analyzer Instructions in Appendix A.

5.1 Access Point Management

The first phase of the analysis involves access point management. Access points are locations on the road network that are known to be accessible to vehicle traffic. It is

possible for there to be restrictions on when certain vehicles may have access at the specified location. These points may correspond to highways or other roads which are designated as being open to vehicles.

The analysis tool provides a way for the user to add access points to a point coverage and specify accessibility attributes to be associated with the access points. The accessibility attributes specify when vehicle traffic is known to be able to have access to the road network at the access points. The portion of the toolset which is used for this part of the analysis is implemented as an Arc Macro Language (AML) script. The script utilizes Arc/Info's ArcEdit module to add user specified access points to a point coverage which stores access points. ArcEdit displays the road coverage and allows the user to graphically specify the location of access points to be stored in the corresponding point coverage with the ADD command and attributes the access point with its accessibility attribute via the MOVEITEM command.

The Forest Service does not have a standard rule system or data set format which specifies information about access points, nor do any of their standard GIS databases have these access points recorded. It is assumed that vehicles can always gain access to any location on a highway, other main road, or a node on the edge of the coverage which is known to be accessible to a highway by a road which is not on the road coverage.

5.2 Barrier Management

The second phase of the analysis involves barrier management. Barriers are entered in a manner similar to how access points are added. The analysis tool provides a mechanism allowing the user to add barriers to a barrier point coverage and allows for associating restriction attributes with the barriers. The portion of the toolset which is used for this part of the analysis utilizes an AML script which is similar to the script used for the management of access points. The script utilizes Arc/Info's ArcEdit module to add user specified barriers to a point coverage which stores the user specified barriers. ArcEdit displays the road coverage and allows the user to specify the location of barriers to be stored in the corresponding point coverage with the ADD command and attributes the barriers with the nature of the restriction imposed on traffic flow by the barrier with the MOVEITEM command.

5.3 Running The Analysis

The third phase of the analysis process involves co-locating the access points and barriers specified by the user onto the road coverage, then running an implementation of CONDITIONALY_CONNECTED_COMPONENTS on that road coverage. The process for merging elements of the access point and barrier coverages are the same. Both point coverages are merged into the road network's line coverage via Arc/Info's SPLIT command. An AML script processes the elements of the point coverages on an individual

basis using an AML cursor. The road coverage arc which is closest to the barrier or access point feature being processed is identified. That arc is then split into two arcs where the arc is closest to the feature being processed. The node connecting the two parts of the split arc is given the attributes of the feature being processed.

The `CONDITIONALLY_CONNECTED_COMPONENTS` analysis program is run on the road coverage, which now includes the barriers and access points represented as attributed nodes. The program is implemented in C++. It accesses the road coverage by reading ASCII copies of the road coverage's AAT and NAT, which have been dumped as ASCII files from Arc/Info via the UNLOAD command. The analysis program produces an Arc Restriction File as an ASCII output file, which records the arcs found to have restrictions and the nature of those restrictions. This file can be loaded into a newly created ART via the Arc/Info ADD command.

The `CONDITIONALLY_CONNECTED_COMPONENTS` analysis program can achieve additional performance gains by modifying the implementation to directly access tables within Arc/Info, rather than exporting and importing ASCII files. The tables can be directly accessed by using facilities in either Arc Software Developers Library (ArcSDL) or Infolib. ArcSDL is a C application programming interface developed by ESRI which allows C/C++ applications to directly access Arc/Info data structures and invoke Arc/Info commands from within an application. While a copy of ArcSDL is available at the University of Montana Computer Science Department, it is not used in this effort because distribution and support for ArcSDL is highly restricted. Another means of directly

accessing the Arc/Info tables is using Infolib. Infolib is an unsupported C library which allows applications written in C/C++ to directly access the Arc/Info tables. Infolib was developed by ESRI and is available free at many sites on the Internet. However, ESRI makes it clear that this library is not officially supported, so its use in any long term project is also questionable. This, any development of a more robust means to interface the CCC program and Arc/Info is dependent on ESRI's ability and willingness to provide standard interface libraries to their data formats.

5.4 Analysis Results Visualization

The last phase of the analysis entails utilizing the analysis tool's analysis results visualization script to view what impact the barriers have on the road network as determined by the algorithm implementation. This crude visualization mechanism is implemented as an AML script. The script allows the user to specify visualization criteria including begin date, end date and vehicle class. The script calls a C++ program which examines the ART to identify which arcs are restricted from the specified begin date through the specified end date for one or more of the vehicle classes specified. The script then utilizes Arc/Info's ArcPlot module to graphically display the road network. The color used to display an arc signifies how many of the specified vehicle classes are restricted for that arc. Which colors are used is dependent on the local symbol file being used by Arc/Info. When using the default symbol table, white signifies arcs that are not restricted for any specified vehicle class, red arcs are restricted for one specified vehicle

class, green for two vehicle classes and blue for three vehicle classes. The script also identifies arcs which are found to not be on any path to any access point. This visualization script may be run multiple times for different sets of dates and vehicle classes.

5.5 Summary

This analysis tool which utilizes the `CONDITIONAL_CONNECTED_COMPONENTS` is effective in solving the road closure analysis problem. It allows a person to perform in a matter of minutes an analysis that currently must be performed manually at a cost of multiple person-months. As the performance benchmarks indicate, this approach to solving the road closure analysis problem is the most effective approach to automating the analysis when the alternative options for automation are considered. Although this approach is the most effective solution to the road closure analysis problem, additional performance gains could be achieved with a more direct access to the Arc/Info data via standard interface libraries.

Network Accessibility Analyzer Instructions

Overview

The Network Accessibility Analyzer performs an analysis of a road network, determining what effect barriers have on the accessibility of the arcs on the road network. The analysis is performed with the following steps:

- Determination of which restriction references are effective for the road network.
- Specification of access points.
- Specification of barriers.
- Analysis of the effects of the gates on the road network.
- Visualization of the analysis results.

The Network Accessibility Analyzer operates in an Arc/Info environment. The intended audience for this document are individuals who have some familiarity with Arc/Info and their organization's GIS data this process utilizes as input. The creation of road network and polygon coverages mentioned in this document is beyond the scope of this effort.

Restriction Reference Identification

A Restriction Reference File must be created which lists the Restriction Schedule (or Schedule of Closure) which is in affect for the road network to be analyzed. The file shall have a set of records for each Restriction Schedule Code. Each set of records for a given Restriction Reference Code shall have one record for each vehicle type affected by the restriction. Each record shall be on a line of its own. The data included in each record shall be delimited with commas and is defined as follows:

Restriction Code	Code identifying a particular Restriction Schedule. This code must be placed inside single quotes.
Vehicle Type	A type of vehicle the Restriction Schedule applies to. This value must be placed inside single quotes. The Vehicle type shall not exceed 6 characters.

Restriction Begin Date Date the restriction begins for the specified vehicle. This date value must be specified in MM.DD format. (e.g. 09.15 for September 15.)

Restriction End Date Date the restriction ends for the specified vehicle. This date value must be specified in MM.DD format. (e.g. 09.15 for September 15.)

An example of a Schedule of Closures and the associated Restriction Reference File follows.

Restriction Code	Automobile	All-Terrain Vehicle	Snowmobile
A	Yearlong	Yearlong	Yearlong
B	10/1 - 6/30	10/1 - 12/1	No Restriction

Fig 1. Schedule of Closures

'A', 'Auto',01.01,12.31
'A', 'ATV',01.01,12.31
'A', 'Snom',01.01,12.31
'B', 'Auto',10.01,06.30
'B', 'ATV',10.01,12.01

Fig 2. Restriction Reference File

Note that if a Restriction Reference does not apply to a particular vehicle type, no record for that vehicle type is included with that Restriction Reference's set of records. Restriction Reference B from above illustrates this point. (i.e. Restriction Reference B does not include a restriction for snowmobiles.)

Access Point Identification

Access Points are locations on the road network that are known to be accessible to vehicles. It is possible for there to be restrictions on when certain vehicles may have access at the specified location. These points may correspond to Highways or other roads which are designated as being open to vehicles.

- Step 1:** Identify the road coverage corresponding to the road network which is to have barriers added.
- Step 2:** Identify a polygon coverage which will assist in giving reference when viewing the road coverage. Some sort of polygon coverage representing jurisdictional boundaries is often useful as reference when viewing a road coverage. The usage of such a polygon coverage is optional.
- Step 3:** Start AP_MGR aml script. The script can be invoked by typing the following at the ARC command prompt:

```
<AML_dir>/ap_mgr <Rd_Cover> <AP_Cover> <Back_Poly_Cover>
```

- AML_dir is the directory where the AML scripts for the network analyzer have been installed. If you do not know the path name for that directory, ask your system administrator.
- Rd_Cover is the road coverage identified in Step 1.
- AP_Cover is the name of the point coverage that will contain the access points specified by the user while running this script.
- Back_Poly_Cover is the polygon coverage identified in Step 2. This argument to the script is optional. Only include this coverage if the user desires to see a polygon coverage in addition to the road coverage when entering Access Points.

As the script starts, a ArcEdit session is started and the road coverage is displayed.

- Step 4:** Once the script starts, the user is prompted for a Restriction Reference. If the first Access Point to be entered does not have any restrictions on when any vehicle type may have access to that point, hit a return at this prompt. If the first Access Point does have restrictions on when it is accessible, enter the Restriction Reference Code corresponding to that restriction.
- Step 5:** Move the cursor onto the ArcEdit window over the point on the road coverage where an access point is located and click the left mouse button.
- Step 6:** To add additional access points with the same restriction reference move the cursor to the points where additional access points are located and click the left mouse button where each additional access points is to be added.
- Step 7:** To add access points with a different restriction reference, move the cursor to where the first access point with the new restriction reference

To add additional access points with this same restriction reference move the cursor to where the additional access points are located and click the left mouse button (as in Step 6.)

Step 8: To stop adding access points click the right mouse button. As the script is exiting ArcEdit the user is prompted as to whether they want to save their work. Answer yes to this query (both times.)

Note: It is recommended that the user save their work periodically. This is accomplished by exiting, then restarting the script.

Barrier Identification

Barriers are locations on the road network where restrictions are imposed on the travel of vehicles. These points may correspond to Gates, Kelly Humps, Berms, Washouts or other means of restricting further access on a road or trail. Gates may be permanently closed or may only be closed seasonally. Barriers are entered in a manner similar to how access points are added.

Step 1: Identify road coverage corresponding to the road network which is to have gates added.

Step 2: Identify a polygon coverage which will assist in giving reference when viewing the road coverage. Some sort of polygon coverage representing jurisdictional boundaries is often useful as reference when viewing a road coverage. The usage of such a polygon coverage is optional.

Step 3: Start GATE_MGR aml script. The script can be invoked by typing the following at the ARC command prompt:

```
<AML_dir>/gate_mgr <Rd_Cover> <Barrier_Cov> <Back_Poly_Cov>
```

- AML_dir is the directory where the AML scripts for the network analyzer have been installed. If you do not know the path name for that directory, ask your system administrator.
- Rd_Cover is the road coverage identified in Step 1.
- Barrier_Cov is the name of the point coverage that will contain the barriers specified by the user while running this script.
- Back_Poly_Cov is the polygon coverage identified in Step 2. This argument to the script is optional. Only include this coverage if the user desires to see a polygon coverage in addition to the road coverage when entering Access Points.

As the script starts, a ArcEdit session is started and the road coverage is displayed.

- Step 4:** Once the script starts, the user is prompted for a Restriction Reference. Enter the Restriction Reference Code corresponding to the restriction imposed by the first barrier to be entered.
- Step 5:** Move the cursor onto the ArcEdit window over the point on the road coverage where the first barrier of the session is located and click the left mouse button.
- Step 6:** To add additional barriers with the same restriction reference move the cursor to the points where additional barriers are located and click the left mouse button where each additional barrier is to be added.
- Step 7:** To add barriers with a different restriction reference, move the cursor to where the first barrier with the new restriction reference is located and click the middle mouse button. The user is then queried for the restriction reference for the barrier that was just entered. To add additional barriers with this same restriction reference move the cursor to where the additional access points are located and click the left mouse button (as in Step 6.)
- Step 8:** To stop adding barriers click the right mouse button. As the script is exiting ArcEdit the user is prompted as to whether they want to save their work. Answer yes to this query (both times.)
- Note:** It is recommended that the user save their work periodically. This is accomplished by exiting, then restarting the script.

Network Analysis

Network Analysis entails merging the barrier and access point coverages created in the preceding phases into the road coverage corresponding to the road network to be analyzed. A merged road coverage with nodes representing the gates and attributes is the result of this merger. After the merger, a program is run on the merged road coverage determining what effect the gates have on the road network. Each arc that is found to be restricted by one or more gates is attributed with the restrictions imposed by those gates.

- Step 1:** Start NW_ANLZR aml script. The script can be invoked by typing the following at the ARC command prompt on one line:

```
<AML_dir>/nw_anlzl <Rd_Cov_In> <Mrg_Rd_Cov> <Barrier_Cov>
<AP_Cov> <Restr_Ref_File> <Search_Dist>
```

- AML_dir is the directory where the AML scripts for the network analyzer have been installed. If you do not know the path name for that directory, ask your system administrator.

- **Rd_Cov_In** is the road coverage identified in Steps 1 of *Access Point Identification* and *Barrier Identification*
- **Mrg_Rd_Cov** is the merged road coverage containing the barrier (**Barrier_Cov**) and access points (**AP_Cov**) represented as nodes on the input road coverage (**Rd_Cov_In**). This coverage also has a Arc Restriction Table which contains each arcs restriction attribute if it has any.
- **Barrier_Coverage** is the name of the point coverage that contains the barriers to be merged into the road network as nodes.
- **AP_Coverage** is the name of the point coverage that contains the access points to be merged into the road network as nodes.
- **Restr_Ref_File** is the file discussed above in *Restriction Reference Identification*
- **Search_Dist** is an optional argument which specifies how far the script needs to look to when trying to merge a barrier or access point onto the road network. This has been defaulted a 1000, but a higher value may be specified if necessary.

Analysis Results Visualization

The Analysis Results Visualization script allows the user to see the effect the barriers have on the road network. This crude visualization mechanism allows the user to specify visualization criteria including begin date, end date and vehicle types. The script will color code arcs which are restricted from the specified begin date through the specified end date for one or more of the vehicle types specified, the color of the arc specifying how many of the vehicle types are restricted for that arc. Which colors are used is dependent on the local symbol file being used by Arc/Info. The script will also specify arcs which are found to not be on any path to any access point. This process may be run multiple times for different sets of dates and vehicle types.

Step 1: Start DISP_RES aml script. The script can be invoked by typing the following at the ARC command prompt on one line:

```
<AML_dir>/disp_res <Road_Cov> <Gate_Cov> <Begin_Date>
<End_Date> <Vehicle1> ... <VehicleN>
```

- **AML_dir** is the directory where the AML scripts for the network analyzer have been installed. If you do not know the path name for that directory, ask your system administrator.

- **Gate_Cov** is the name of the gate coverage created in *Barrier Identification*.
- **Road_Cov** is the resulting coverage (**Mrg_Rd_Cov**) from *Network Analysis*.
- **Begin_Date** is the beginning date of the search criteria.
- **End_Date** is the ending date of the search criteria.
- **Vehicle** is the vehicle type associated with the search criteria. Up to 6 vehicle types may be specified.

References

Aho, Alfred V., John E. Hopcroft and Jeffrey D. Ullman (1974) *The Design and Analysis of Computer Algorithms*. Reading, Massachusetts: Addison-Wesley Publishing Company

Baase, Sara. (1988) *Computer Algorithms - Introduction to Design and Analysis*. Reading, Massachusetts: Addison-Wesley Publishing Company

Cormen, Thomas H. , Charles E. Leiserson and Ronald L. Rivest (1990). *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press

Environmental Systems Research Institute, Inc. (1995A) *Understanding GIS - The ARC/INFO Method*. Environmental Systems Research Institute, Inc. Redlands, California

Environmental Systems Research Institute, Inc. (1995B) *ARC Help*. Environmental Systems Research Institute, Inc. Redlands, California

Environmental Systems Research Institute, Inc. (1994) *ARC Macro Language - Developing ARC/INFO Menus and Macros with AML*. Environmental Systems Research Institute, Inc. Redlands, California

Kumar, Vipin, Ananth Grama, Anshul Gupta and George Karypis (1994) *Introduction to Parallel Computing - Design and Analysis of Algorithms*. The Benjamin/Cummings Publishing Company, Inc. Redwood City, California

United States Forest Service (1994) *Route Management System v1.03 User's Guide*