ARO Report 91-1

TRANSACTIONS OF THE EIGHTH ARMY

CONFERENCE ON APPLIED MATHEMATICS

AND COMPUTING

Sponsored by

The Army Mathematics Steering Committee

on behalf of

THE ASSISTANT SECRETARY OF THE ARMY FOR

RESEARCH, DEVELOPMENT, AND ACQUISITION

A Logic Programming Approach to Network Flow Algorithms.

Andrew W. Harrell

U.S. Army Engineer Waterways Experiment Station,

Vicksburg, Mississippi

Abstract

The well-known Ford-Fulkerson algorithm and most of the more recent approaches to solving the network maximal flow and minimal-cost flow problems use a labeling procedure. Labeling involves using nodes in the network which have values and are updated by adding a series of augmenting flows or edges until the optimal solution is reached. In this paper, an alternative approach is examined using the full ordered list of flow paths without cycles. This list is generated by a Prolog-based depth-first search with backtracking of the type described by Winston [13],[14]. This approach keeps track of the full queue of partial search paths and is easier to use to examine the solution for weak-links or critical nodes. If a modeler is creating a network to represent a real situation it is reasonable to assume that the number of ingoing and outgoing edges to a vertex are limited. Time bounds are presented to demonstrate that the above approach is under these conditions as efficient as the n-cubed algorithms explained in Tarjan [12] which use a method of labeled preflows.

Key words - network algorithm, depth-first search, maximal flow, min-cost flow, logic programming, backtracking.

1. Introduction.

With the development of computer graphics techniques for displaying digital map information, new ways of representing unit movement and aircraft or ship routing have been developed. However, the use of digital map data presents problems as for example representing the effects of various types of on- and off-road obstacles, underwater mines, bridge interdiction on the movement rates, and routing possibilities. Programs must be written to define and store route movement networks and arrays of obstacles.

This research was supported in part by Headquarters, US Army Corps of Engineers, Washington, DC 20314-1000.
Author's Address: Dr. Andrew Harrell, US Army Engineer Waterways Experiment Station, 3909 Halls Ferry Road, Vicksburg, MS 39181-0631 Internet: h4enaah0@vicksbrg.army.mil

For this paper, we will assume this already exists along with avenues of approach or movement corridors and their corresponding traverse speeds across the map. Harrell [7],[8],[9] gives a partial description of some current techniques of doing this. The following short glossary defines some of the basic terms that will be used:

Backtracking - An algorithmic search scheme which in order to compute all the ways to satisfy a given goal computes one solution through following a series of branching points and then retraces its steps to the last previous decision in order to compute another possible solution.

Cycle - A path with the same starting and ending node.

Dead end - A node in a network from which no edges proceed.

Edge - The line connecting two nodes in a network. Each edge in a network usually has associated with it a traverse time, vehicular speed, or a flow-rate, and represents a given portion of the overall map.

Flow- An assignment of of flow-rates to some or all of the edges in a given network. Each flow-rate has to be less than or equal to the flow-capacity of its edge.

Flow-capacity- The largest allowable flow-rate for a particular edge.

Flow-rate - The number of vehicles per hour that can pass over a given edge in the network. As explained in the text this can be calculated as [1/(time it takes a group of vehicles to traverse the edge)]*number of vehicles in the group.

Maximal flow problem - The problem of determining what is the greatest number of vehicles/hour that can travel through a network at a given time. It is computed by designing an algorithm to optimize the assignments of flows to edges in the network.

Maximal flow value - The value which is a solution to a Maximal flow problem. Note, that it is possible for there to be several different network flows which realize a given maximal flow value.

Min-cost flow network- A flow network with costs (times to traverse) as well as flows associated with its edges. In this paper in order to determine the cost associated with a flow the following procedure is followed: 1) The flow rate on each flow path solution through the network is multiplied by its time of traversal and the result summed over all paths in order to obtain a total cost associated with a given maximum flow solution. This is the measure of effectiveness which determines the optimality of the solution. The total cost of the flow can then be divided by the total maximum flow to obtain an average cost per vehicle to travel through the network.

Min-cost flow problem- The problem of determining from all the possible flows which realize a given network maximal flow value, those which do it with minimal cost.

Network - A collection of nodes and edges that represent movement possibilities over a given terrain area.

Node - A point of reference in a network from which edges are drawn from and into.

Path - An ordered list of edges each of which has the same starting node as the preceeding edge's ending node.

An artificial intelligence network algorithm is methodology based on searches for paths from start nodes through a network to ending goal nodes using the methods of logic programming. The search mechanism proceeds in an orderly fashion unifying the variables in the search predicates from one level of search in the network to another. The algorithm used must save the partial solutions in an environment list so it can backtrack its way through the previous variable bindings in order to generate all possible ways of reaching the goal state. This differs from many network algorithms that use labels (instead of a list of partial search paths) at the nodes to store information as the steps in the algorithm proceed. Thus, after the labeling algorithms are through generating solutions, information is not kept on "how" the solutions were reached.

The search algorithm discussed in Section 2 below will print out ordered lists of shortest paths with and without the presence of obstacles. These lists reveal the critical nodes or weak links that most affect the optimal paths in the network. In order to do this and compute movement possibilities across cross-corridors an algorithm has to keep track of more information than can be stored on just a single label per node in the network or on a single search tree. One needs to store the same kind of list of partial solutions that a logic programming unification algorithm does when it tries to satisfy goal predicates.

Similarly, in developing programs to compute network flow rates which identify the critical nodes in the solution, it is important to compute the maximal or min-cost flows in terms of an ordered list of paths from the start node (or set of nodes) to the goal node (or set of nodes). The solution can be displayed just as a logic programming interpreter displays in turn the list of predicate variable identifications which satisfy the specified goal. The question then becomes whether this approach is feasible in terms of search time bounds and how it is implemented.

These questions are answered in this paper which contains five sections. In Section 2 the main search algorithm used to compute shortest paths or maximal flow paths and give the derivation of the number of search steps required to generate all these solutions is presented. In Section 3 an explanation of how this algorithm can be used to solve the maximum flow problems associated with certain types of networks is discussed. In section 4 simple modifications to this algorithm are presented that can be used to solve the corresponding min-cost flow problem. Section 5 contains a short discussion of the appropriateness of these algorithms for the transportation problem and the assignment problem, and the next and final section contains the conclusions.

ACKNOWLEDGEMENT

The tests described and the resulting data presented herein, unless otherwise noted were obtained from research conducted under the MILITARY RESEARCH AND DEVELOPMENT TEST AND EVALUATION PROGRAM of the United States Army Corps of Engineers by the US Army Waterways Experiment Station. Permission was granted by the Chief of Engineers to publish this information. Unlimited Distribution/Public Release.

The Main Search Algorithm

As mentioned above, outputting the full search path allows the user to determine the effect of weak links or choke points on the solution. For example, in on- or off-road movement networks based upons digitized maps, it is important to know the effects of minefields, anti-tank ditches, abatis, and road craters on the overall possible vehicular flow rate vehicles traverse across the terrain. Network path-generating algorithms based upon dynamic programming, dynamic tree structures, or node labeling do not save the information on the movement possibilities through cross-corridors in the terrain. This increases computational speed in many cases, but important information about the vulnerability or sensitivity of the solution to degrading factors is lost. The best algorithm for these purposes is one that provides a way to measure the effect of changing flow rates and times in certain parts of the network on the overall solution.

An example of such an algorithm is given below. The search procedure presented keeps track of the next best choices in a sorted priority queue. This is necessary so the algorithm can backtrack quickly to find another solution after it has determined the shortest path or failed to reach a goal in a given direction. In order to do this, it was convenient to write the program in Prolog. An algorithm that does this is described in the book by Winston [13] . The description of the algorithm is as follows :

Step 1 Form a queue of partial paths. Let the initial queue consist of the zero-length, zero-step path from the start node to nowhere.

```
Step 2 Until the queue is empty or the goal has been reached
determine if the first path in the queue reaches the goal
node.

    Step 2a If the first path reaches the goal node,
    do nothing.

    Step 2b If the first path does not reach the goal
    node:

      Step 2b1 Remove the first path from the queue

      Step 2b2 Form new paths from the removed path
      by extending them one step

      Step 2b3 Add the new paths to the queue

      Step 2b4 Sort the queue by cost accumulated so
      far, with least cost paths placed in front.

Step 3 If the goal node has been found, announce success;
    otherwise, announce failure.
```

The algorithm as given terminates when the shortest incomplete path is longer than the shortest complete path. In this situation there are no further paths needing to be investigated for optimality. Since the paths which could never be optimal have been pruned out at an earlier stage, the queue remaining (which has been sorted at each stage) contains at its head the optimal path.

The Prolog source code and Pascal source code for one particular implementation of the algorithm is given in Harrell's report [9] and it can be implemented in the C language using essentially the same code. There is a way to implement the algorithm using a dynamic tree structure to keep the environment of partial solutions which it is able to backtrack through (see the book by Bratke). However, as mentioned above, a tree can store information about only one partial path from its root to each leaf or subtree node.

The question then becomes whether the list of all partial paths accumulated using the search algorithm becomes so large that it is impractical to manipulate. The theorems and the lemmas listed below prove that under certain restrictions, such as: 1) no dead ends in the network, 2) the maximum numbers of nodes going in and out of a vertex bounded above, and 3) the maximum number of nodes which are critical in the sense defined below is bounded, the time it takes to finish this type of algorithm is not longer than for the algorithms which compute shortest paths to create maximal and min-cost flows according to the approaches of Edmonds and Karp [4].

[illegible]

Let:
ni = the number of nodes with i edges proceeding from them,
nji = the number of nodes with j edges entering them and i edges proceeding from them,
maxe = the maximum number of edges proceeding from any node in the network,
emax = the maximum number of edges entering any node in the network.

Call a node a critical backtracking node if it has more than 1 edge proceeding from it and more than one edge entering it.
Let ncrit = the number of critical bactracking nodes in a network.
Call a node of the network a q stage lth critical path backtracking node if it is a critical path backtracking node and it is preceeded in the network by q levels of backtracking nodes, each having more than 1 edge entering them. Moreover, there must be 1 of these backtracking nodes with more than 1 edge at the preceeding search level to the given node.

Let $n_{ql}ji$ = the number of q stage ,lth critical path backtracking nodes with j edges proceeding into them and i edges leaving them.
n1crit = the number of critical path backtracking nodes which are not q level lth critical for q or l > 1.
Examples of these definitions will be given in the course of the following discussion.

Theorem 1 Given a connected directed graph with a starting node and a goal node and no dead ends other than the goal node. Moreover, if there are at most ncrit critical path backtracking nodes with at most a q level instance of prior influence, then the number of different paths (containing no cycles) from the starting node to the ending node is bounded by the expression

$$1 + (emax -1)*(n -n1-ncrit) + (maxe*emax -1)*n1crit \\ +((maxe*emax)^{q} - 1)*(ncrit -n1crit)$$

Proof: This theorem is proved by following through the steps of the above algorithm and counting the number of ways new paths are generated. Step 2b4 which insures the solutions will be generated in order of shortest length is not necessary if the algorithm is only being used to generate all possible paths. At Step 2b, new paths are added to the queue of partial paths each time the search predicate finds a node following the current node which does not form a cyclic path. Since 1) the graph is finite, 2) there are no dead ends, 3) the graph is connected, 4)no cycles are permitted, then each new path will eventually reach the goal node.

During the generation of the list of partial paths, nodes with only one edge proceeding into them and one edge leaving them expand the current path but do not add any additional combinatorial search possibilities to keep track of during the backtracking process.

Then, the number of paths which the non-critical backtracking nodes enter into is:

$$1 + 1*n2 + 2*n3 + 3*n4 + (j-1)*nj + \ldots (emax - 1)*nemax \quad (1)$$

Using the fact that $n = n1 + n2 + \ldots nemax$ we note that the above number is bounded by:

$$(emax - 1)*(n - n1) + 1$$

The example below (Figure 1) illustrates how equation (1) counts paths in a network without any critical backtracking nodes.

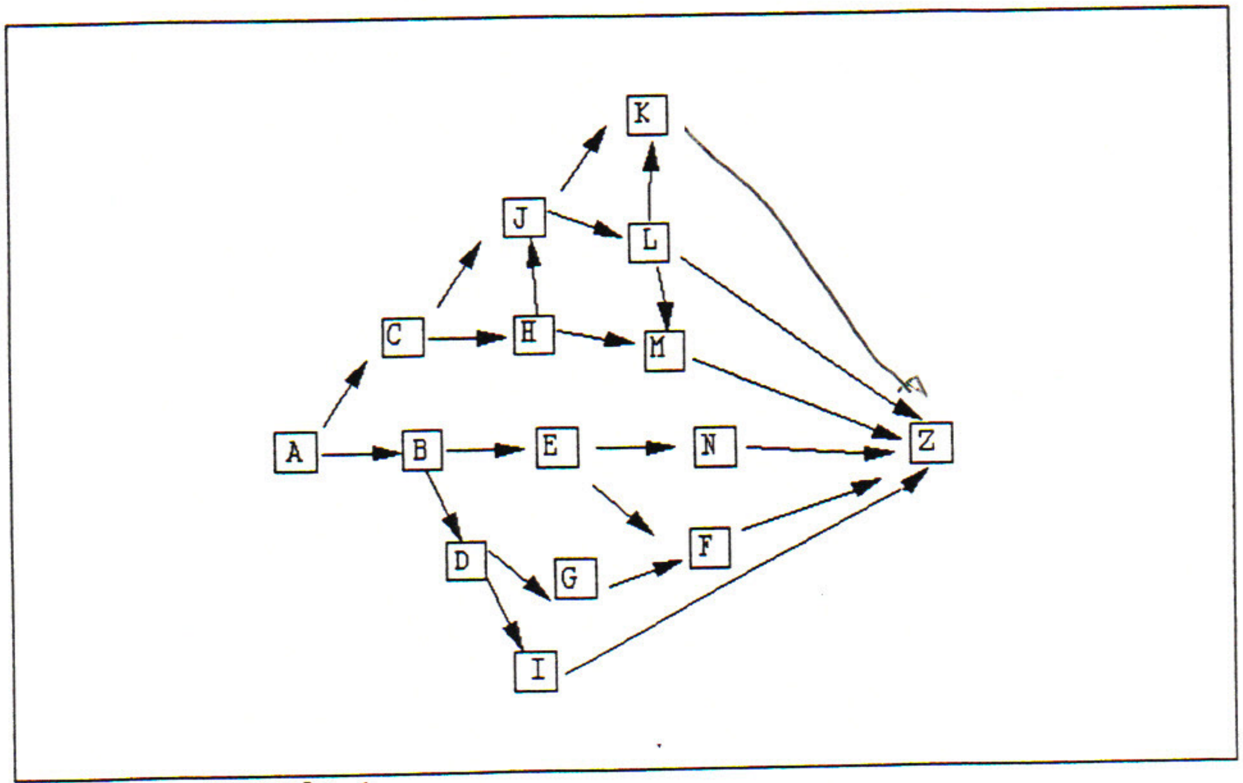


Figure 1. Example 1

There are 14 nodes,
not counting Z.

$n1 = 6$, $n2 = 7$

$n3 = 1$

$emax = 3$

number of paths = $(7 + 1)*1 + 2*(1) = 10$

path	search steps used to generate path[1]
ABDIZ	4
ABDGZ	1
ABDGFZ	1
ABEFZ	2
ABENZ	1
ACHMZ	3
ACJLMZ	3
ACJLZ	0
ACJLKZ	1
ACJKZ	1

If the network we are considering has q stage lth critical path bactracking nodes but none with q or l greater than 1 then the search algorithm will generate an additional:

$$3*n_{11}22 + 5*n_{11}23 + \dots 5*n_{11}32 + \dots (j*i - 1)*n_{11}ji + \dots (maxe*emax - 1)*n_{11}maxeemax \quad \text{paths.}$$

This number is bounded by:
(maxe*emax - 1)*(n1crit).

Example 2 - consider the following movement network, having two starting nodes A1 and A2 and three goal nodes E1, E2, and E3:

Solution:

[1] A search step is defined to be one cycle of search through the database of edges to determine which nodes are connected to a given edge. It is assumed the network information is stored in a vector structure in which each edge along with its starting and ending node and value are kept. Since in generating the queue of search paths a new path uses the nodes from the prior search paths, it is not necessary to search through the database for all the prior nodes in creating the new paths.

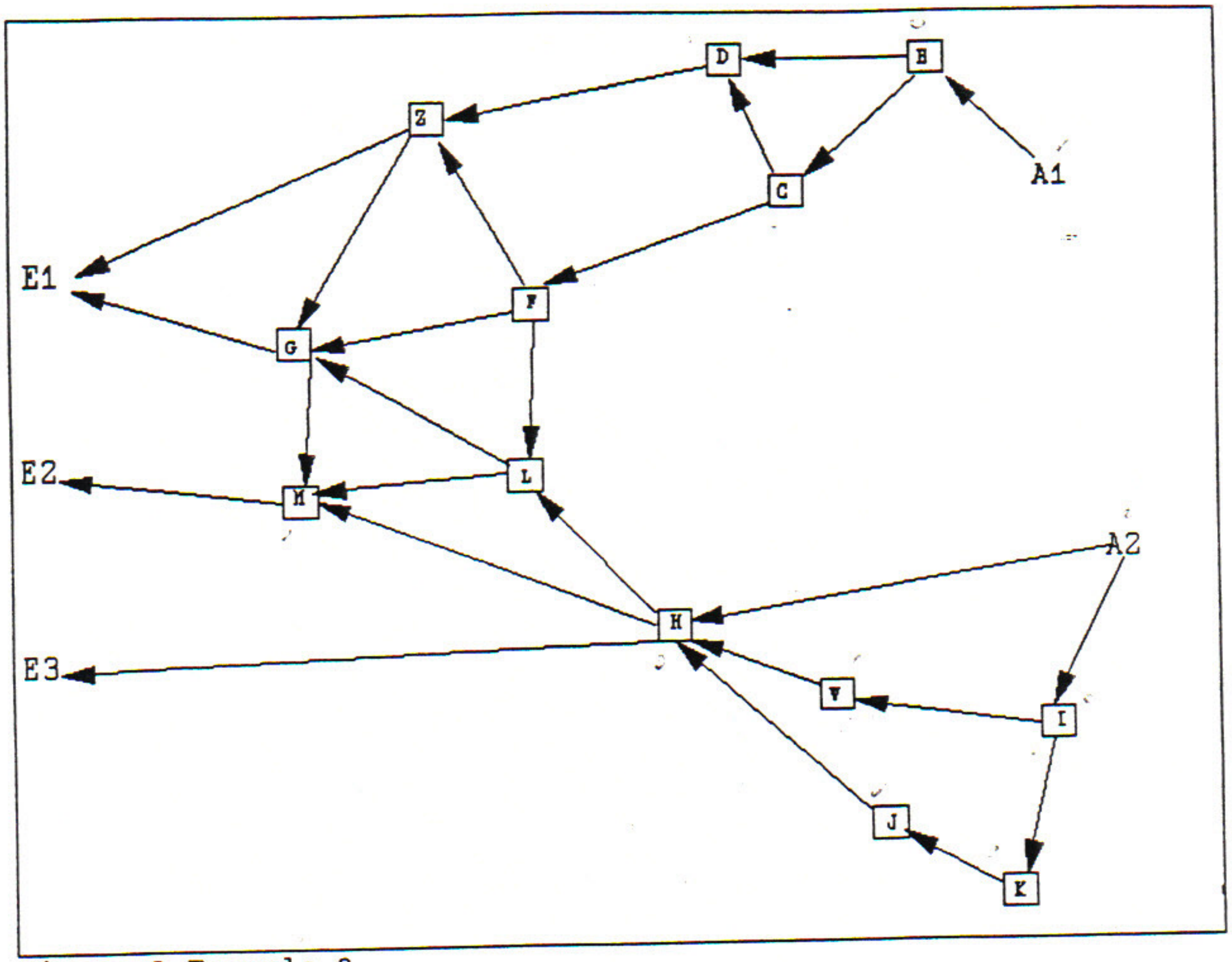


Figure 2 Example 2

To compute all the paths for this network break it up into six connected components corresponding to each possible combination of starting node and ending node. Figures 3 through 7 show this.

1) A1 - E1

ABDZE	8 solutions Z is a 1-stage 1th critical path backtracking node
ABCFGE	
ABCDZE	$n2 = 2 \quad n3 = 1 \quad n_{11}22 = 1$
ABCFLGE	
ABDZGE	$8 = 1 + n2*1 + n3*2 + n_{11}22*(2*2 -1)$
ABCFZE	$= 1 + 2 + 2 + 3$
ABCDZGE	
ABCFZGE	

2) A1 - E2

ABCFGME
ABCFLME
ABCFLGME
ABDZGME
ABCDZGME
ABCFZGME

6 solutions   n3 = 1   n2 = 3

$6 = 1 + 1*n2 + 2*n3 = 1 + 3 + 2$

3) A1 - E3   no solutions

4) A2 - E1

AHLGE
AIWHLGE
AIKJHLGE

3 solutions   n2 = 2

$3 = 1 + 1*n2 = 1 + 2$

5) A2 - E2

AHME
AIWHME
AIKJHME
AHLME
AHLGME
AIWHLME
AIWHLGME
AIKJHLME
AIKJHLGME

9 solutions   n2 = 3 $n_{11}32 = 1$

$$9 = 1 + n2*1 + n_{11}32*(3*2 - 1)$$
$$= 1 + 3 + 5$$

6) A2 - E3

AHE
AIWHE
AIKJHE

3 solutions   n2 = 2

$3 = 1 + n2*1 = 1 + 2$

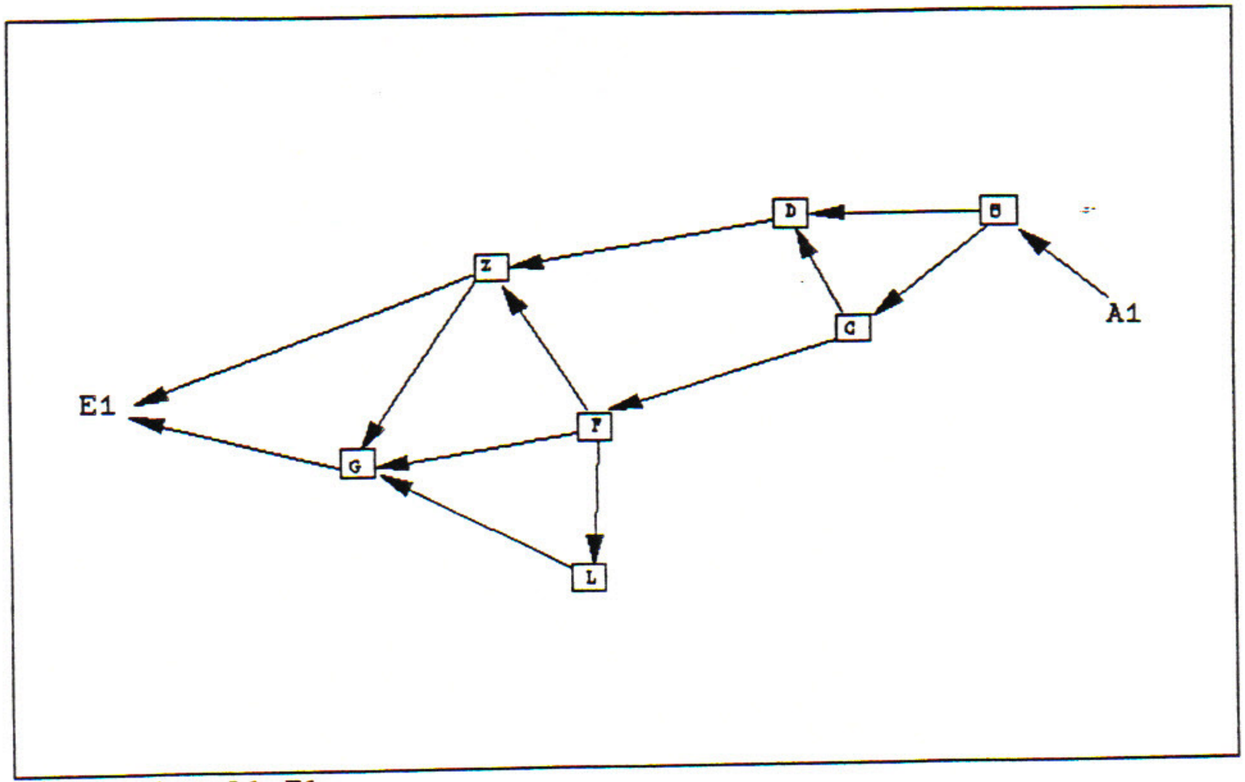


Figure 3   A1-E1

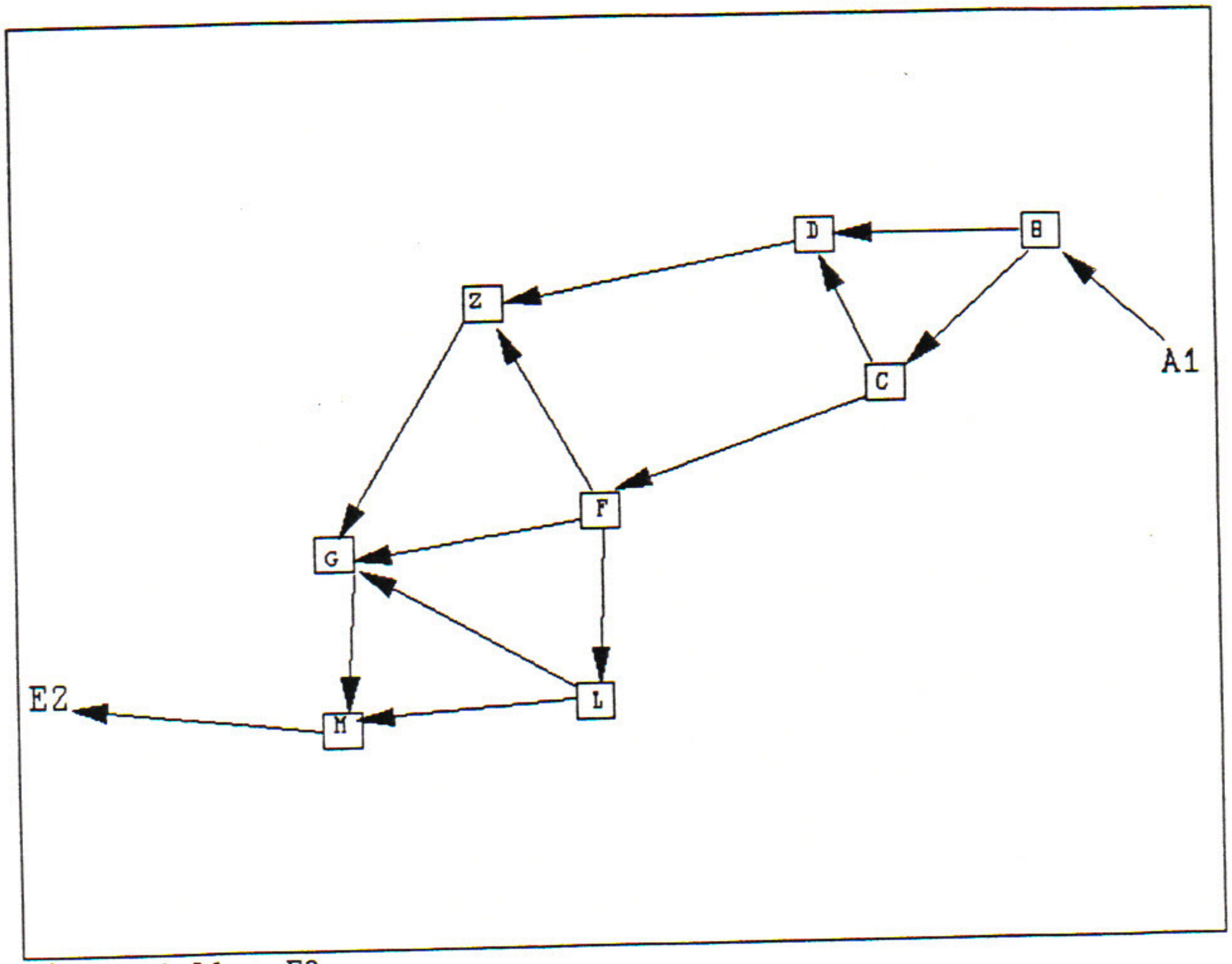


Figure 4 A1 - E2

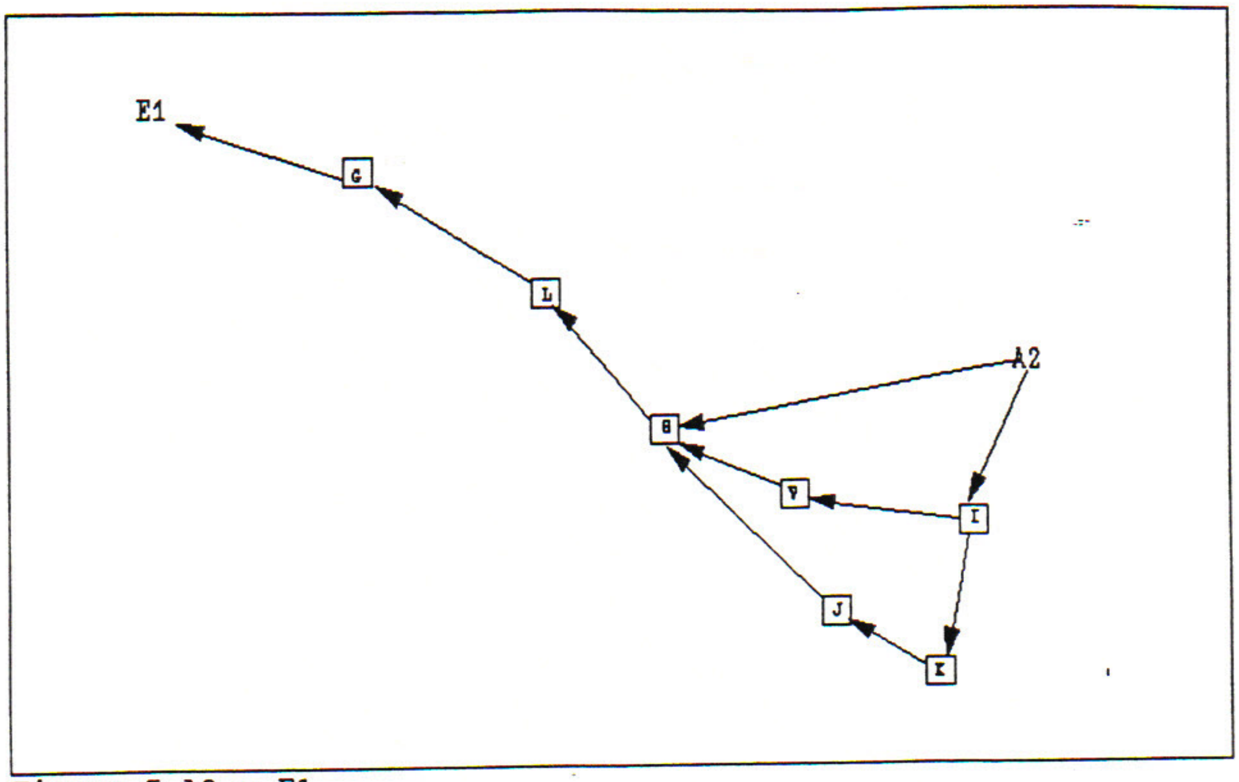


Figure 5 A2 - E1

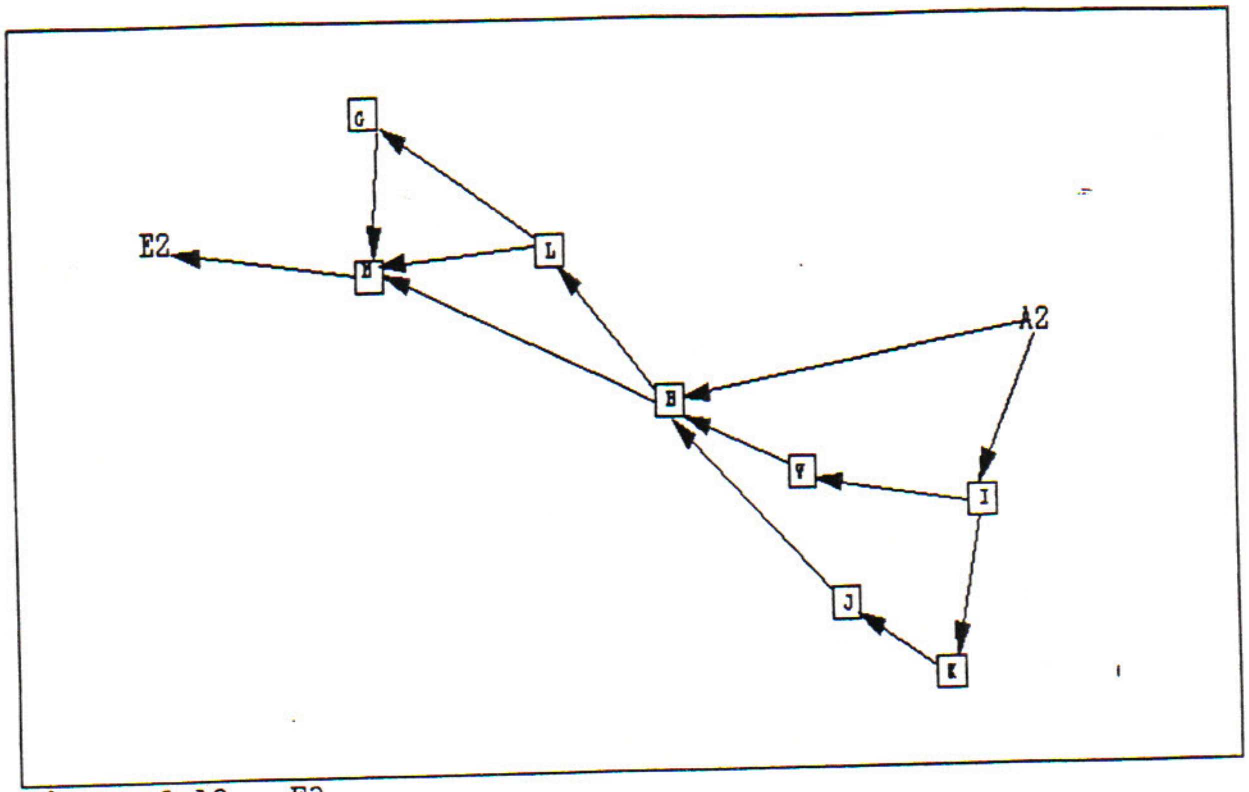


Figure 6 A2 - E2

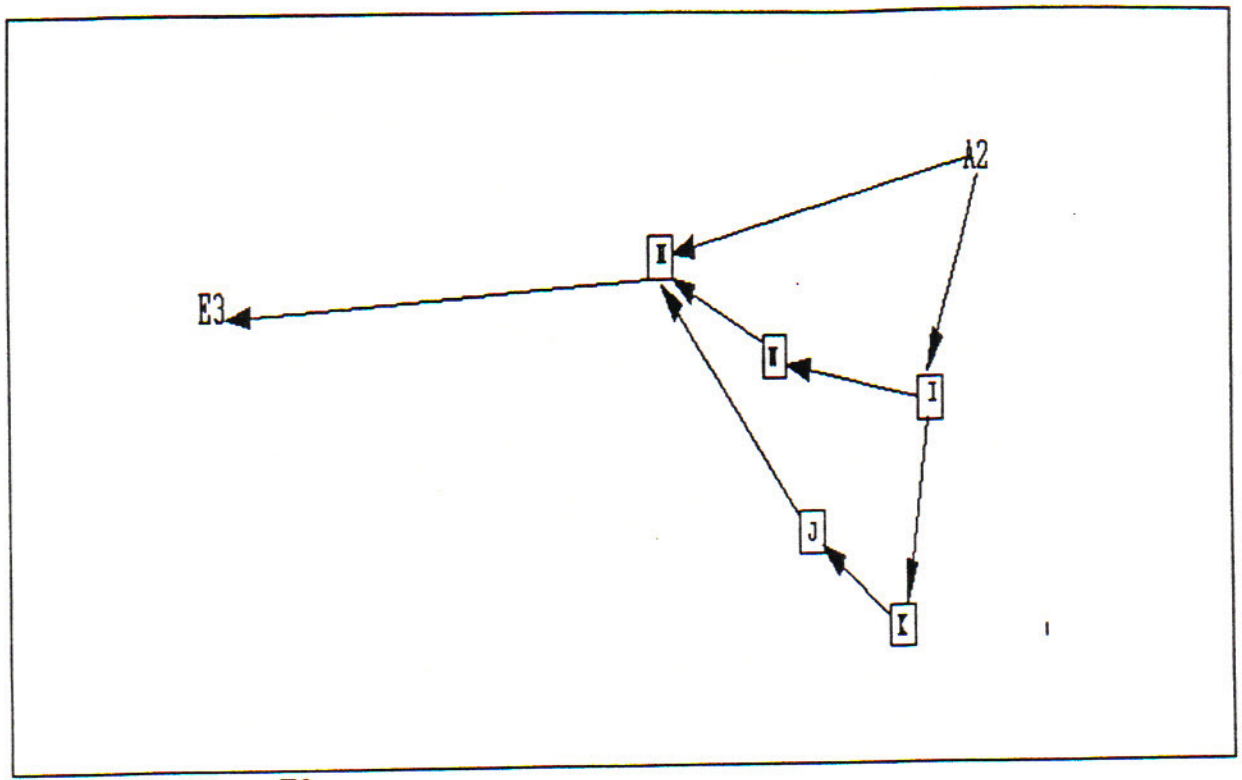


Figure 7 A2 - E3

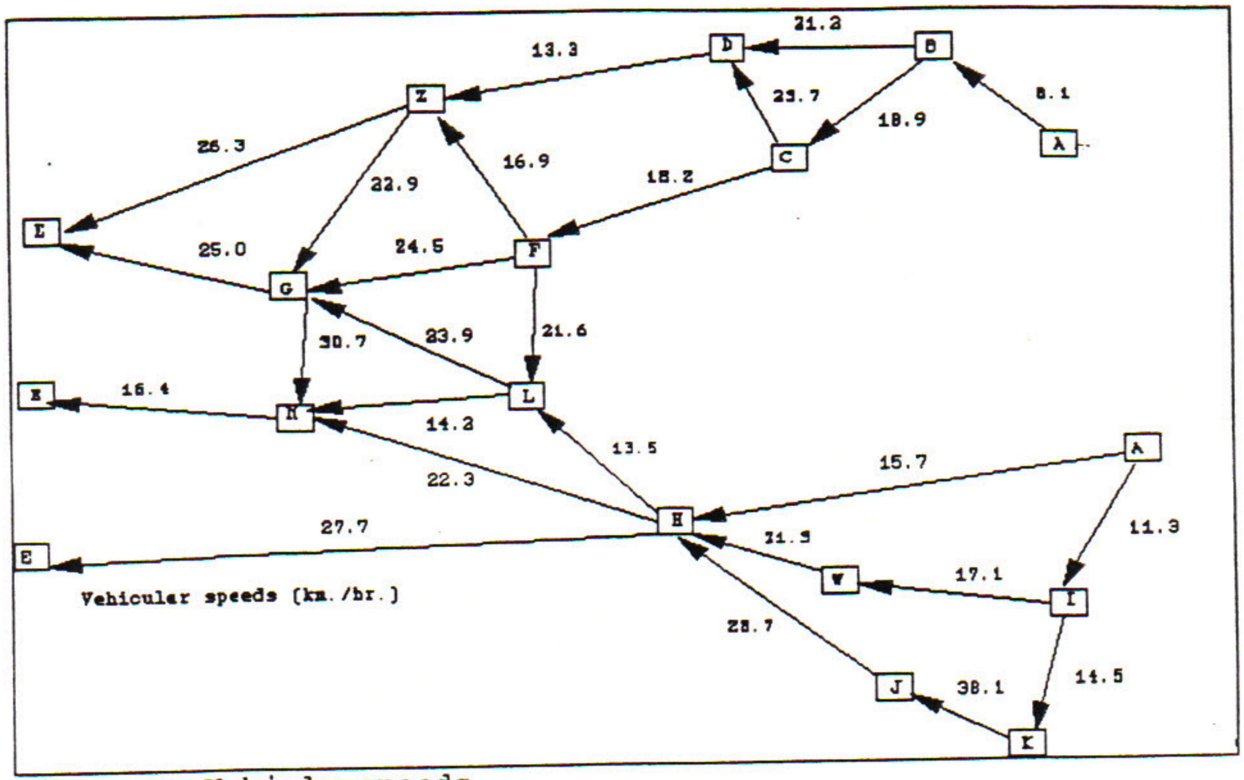


Figure 8. Vehicle speeds

If vehicular speeds are added to the edges in the network of example 2 as shown in Figure 8, the shortest path search algorithm may be used to produce the full list of non-cyclic paths ordered by their length in time (minutes to traverse). Figure 9 shows the one shortest path and the full ordered list is shown below.

```
SHORTEST PATHS

AHE time(min)=      48.7
AIWHE time(min)=     56.8
AHME time(min)=     63.0
AIKJHE time(min)=     66.2
ABDZE time(min)=     67.2
ABCFGE time(min)=     67.2
AIWHME time(min)=     71.0
ABCFGME time(min)=     74.1
```

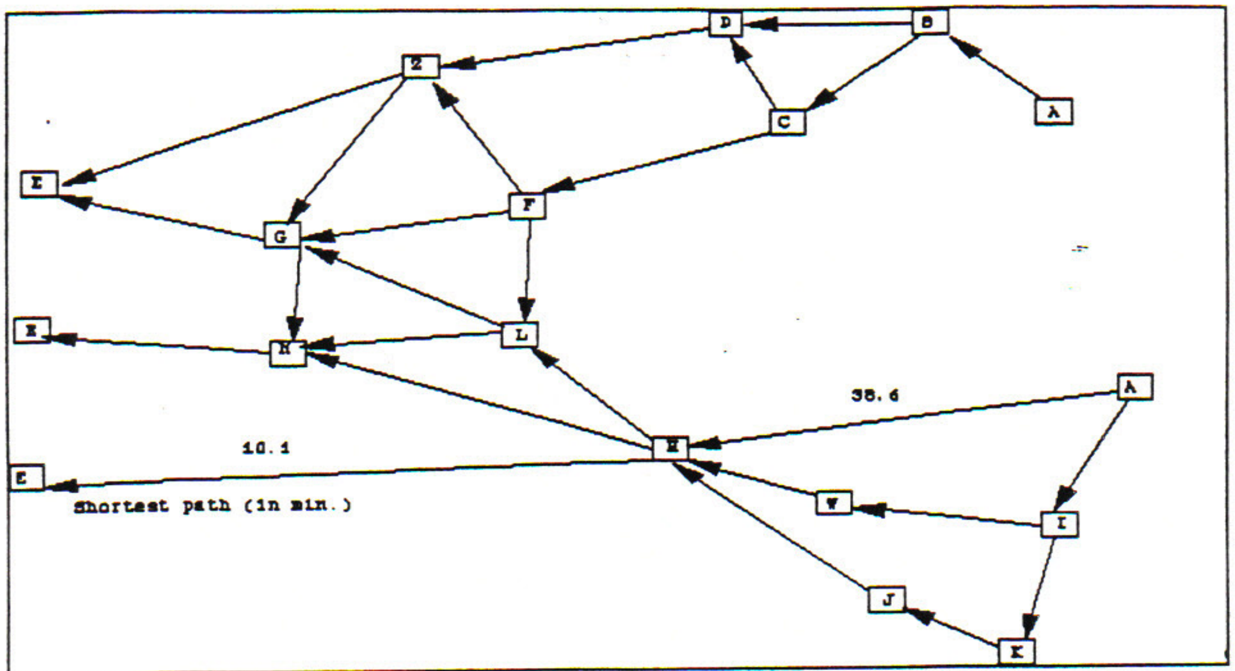


Figure 9. Shortest-path, cross country movement network

```
ABCDZE time(min)=      74.9
ABCFLGE time(min)=      76.2
ABCFLME time(min)=      78.9
AHLGE time(min)=      79.8
AIKJHME time(min)=      80.5
ABDZGE time(min)=      80.6
ABCFZE time(min)=      81.4
AHLME time(min)=      82.5
ABCFLGME time(min)=      83.0
AHLGME time(min)=      86.7
ABDZGME time(min)=      87.4
AIWHLGE time(min)=      87.9
ABCDZGE time(min)=      88.4
AIWHLME time(min)=      90.6
AIWHLGME time(min)=      94.7
ABCFZGE time(min)=      94.8
ABCDZGME time(min)=      95.2
AIKJHLGE time(min)=      97.4
AIKJHLME time(min)=     100.0
ABCFZGME time(min)=     101.7
AIKJHLGME time(min)=     104.2
```

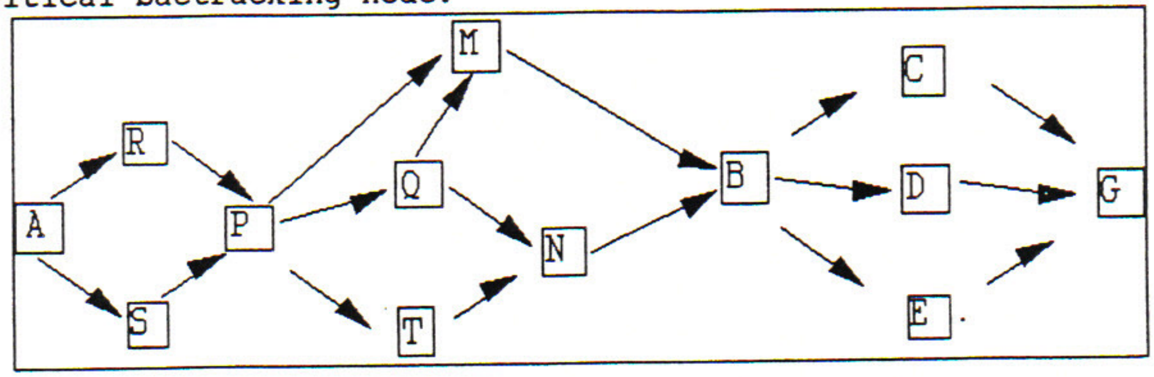
If the network contains q level lth critical path backtracking nodes with q or l greater than 1, then the computation of the number of possible paths becomes more complex. Given $n_{ql}ji$ q level the mth among the q levels of that sth node which are preceeded by backtracking nodes with more than 1 edge entering them and the rth among the l edges entering the node, let $PRIOR_{m.r.s}$ be the number of edges entering that prior critical backtracking node. Then, the number of new paths generated by these nqlji q level lth critical backtracking nodes is bounded by:

$$\sum_{s=1}^{n_{ql}ji} \left(\prod_{m=1}^{q+1} \left(\prod_{r=1}^{l} PRIOR_{m.r.s}\right)\right) * i_s - 1$$

i_s = number of paths leaving the sth critical backtracking node (which is bounded by maxe)

So, the total number of such additional paths will be bounded by (ncrit - n1crit)*($emax^q$ *maxe) - 1. Note, in this computation, a critical backtracking node may preceed (by occuring closer to the start node in the network) more than one other such node a certain number of times. In this case this formula will overcount the number of paths by that same factor. See the example below for an illustration of how this can occur. But whatever the case, these additional paths exhaust all the ways in which the algorithm can possibly backtrack to produce solutions. Thus, adding this bound to the previous one we have the expression given in the statement of the theorem. Q.E.D.

Example 3) In the network below the node B is a 2 level 2th critical bactracking node.



ARPMBCG	ARPNBCG
ASPMBCG	ASPNBCG
ARPQMBCG	ARPTNBCG
ASPQMBCG	ASPTNBCG
ARPMBDG	ARPNBDG
ASPMBDG	ASPNBDG
ARPQMBDG	ARPTNBDG
ASPQMBDG	ASPTNBDG
ARPMBEG	ARPNBEG
ARPQMBEG	ASPNBEG

24 solutions $s = 1$, $q = 2$, $l = 2$, $i_1 = 3$

$$48 = p_{3.1.1}\{p_{2.1.1} * p_{1.1.1} + p_{2.2.1} * p_{1.2.1}\} * 3$$

$$= 2\{2*2 + 2*2\}*3 > 24$$

$$PRIOR_{m.r.s} = p_{m.r.s}$$

ARPMBEG	ARPNBEG
ARPQMBEG	ASPNBEG
ASPNBEG	ARPTNBEG
ASPQMBEG	ASPTNBEG

~~PRIOR$_{m.r.s}$ = p$_{m.r.s}$~~

In this example B is a two stage 2th critical backtracking node. This is because the 2 backtracking nodes M and N which precede B in the network have more than 2 edges entering them and and there are 2 levels of critical backtracking nodes P and then M,N which preceed B. Because the node P preceeds the nodes M and N, the formula overcounts by a factor of 2.

In the applications that these theorems are used we have some freedom in how many nodes and edges are included in the network. The network will be a representation or model of some physical situation or process. In practice it becomes more and more unlikely in a random physical situation that planar graphs containing many q level lth critical backtracking nodes where q or l >> 1 will occur. In movement networks based upon digitized maps such nodes correspond to critical choke points at which the routes diverge to go around an obstacle and then reconverge.

Theorem 2 Considering the same type of graph as in Theorem 1, now allow paths to travel in directions opposed to the way the edges are directed. Then the bound for the number of possible paths is increased to :

$$1 + emax*(n - n1 - ncrit) + ((emax + maxe)*maxe - 1)*n1crit + (((emax + maxe)*maxe)^{q} - 1)*(ncrit - n1crit)$$

Proof: Count the number of paths using the algorithm as in theorem 1. For those nodes with only one edge entering them, the new number of possible edges which proceed outward has been increased by one. These edges will create :

$$1 + 2*n2 + 3*n3 + i*ni \ldots emax*nemax \quad \text{paths.}$$

This number is bounded by 1 +emax*(n - n1 -ncrit). For the nodes with j > 1 edges entering them, the new maximum number of edges leaving the node is emax + maxe. The rest of the formula follows from the previous calculations.

Theorem 3 The number of search steps required to compute all the paths of the type of graphs mentioned in theorem's 1 and 2 is bounded by:

$$1 + emax *(n - ncrit) + [(emax)^{(q+1)}]* ncrit$$

where q = the maximum level of any q level lth critical path backtracking nodes in the network, Proof: This can be verified by tracing through the algorithm and adding a marker to a node's count each time a search step is performed. In following through the algorithm note that since the network is connected each node is encountered during the searching and backtracking exactly the sum of the number of times that there are different edges proceeding into it multiplied by the number of times the preceding nodes to that edge have already been encountered. For the nodes which are not critical path backtracking nodes we have:

1*(1n +1) + 2*(2n) + .. emax*(~~emaxn~~ *(emax)n) search steps,

where in = the number of nodes with exactly i edges entering into them. This number is bounded by the number 1 + emax(n - ncrit). For a node which is a q level lth critical backtracking node, the sum of the number of times that there are different edges proceeding into it multiplied by the number of times the preceeding nodes to that edge have already been encountered is bounded by $(emax)^{(q+1)}$. Q.E.D.

Example ~~2~~ 1 (continued)

In this graph, which has no critical path backtracking nodes, we have 1n = 10 and 2n = ~~4~~ 3 so the number of search steps should be (10 +1) + 2*3 = 17. This can be verified by tracing through the algorithm and keeping a count as below:

node	search encounters	node	search encounters
A	I	H	I
B	I	M	II
D	I	L	I
C	I	J	II
I	I	K	II
G	I	N	I
F	II		
E	I		

total number of encounters = 17

Example 3 (continued)

This graph has a two level 2nd critical backtracking node B. The number of search encounters for the various nodes is as below:

node	search encounters	node	search encounters
P	II	B	IIIIIIII
Q	II	C	IIIIIIII
T	II	D	IIIIIIII
M	IIII	E	IIIIIIII
N	IIII		

total number of encounters = 46

The following theorem holds for the same reasons as the preceeding discussions.

Theorem 4 The number of search steps required to compute all the paths in the case where the paths are allowed to travel in opposite directions is bounded by:

$$1 + (emax + maxe)*(n - ncrit) + \{(emax + maxe)^{(q+1)}\}*(ncrit - n1crit)$$

Methodology for Solving Maximal Flow problems

Given that there is a method of generating all the shortest paths through the network, it is easy to modify the predicates to generate a list of maximal flow paths through the network. We now assume that each route segment has an associated maximal flow capacity. At each stage, simply choose the direction of maximal flow to expand the paths, and perform a sort on the maximal flow of the route segments instead of minimum length. Then, write a predicate that each time we reach the goal node with a route, go back and subtract that route's flow values from the network capacities. When the Prolog backtracking search does not generate any more solutions, all directed paths through the network have at least one edge which is already filled to capacity. One other way exists to increase the flow in the network. The paths which contain edges that point backward along the allowed route segments can be considered. Then, when the goal node is reached, proceed back to the source and modify the flow capacities, and add flow capacity along those segments, instead of subtracting it. As explained by Sedgewick [10] when the above procedure reaches a situation in which all paths have either full forward edges or nonempty back edges, then the Ford Fulkerson theorems says the maximal flow of the network has been reached. Many of the algorithms presently in use Goldberg [6] and Tarjan [12] for solving the maximal flow problem do not save lists of partial paths but instead use a labeling process to update information at each node. This increases computational speed, but makes it difficult to pick in order the main routes that contribute to the optimal solution. Since it may be desirable to do sensitivity analyses which locate the points in the network which most affect all the possible solutions, the above approach gives more information after the process is completed. Thus, it is possible to print out in order of flow the paths which contributed to the max-min cut situation. This then can be used to plan barriers for the defense or attack routes for the offense. The description of the algorithm is as given below:

Step 1. Search for the best(maximal flow) route from the starting to the ending node. Only search in directions in which there is either a forward edge with positive unused flow capacity, or a backward edge with positive existing flow. If no route exists, terminate the algorithm.

Step 2. Subtract the value of that flow from the capacity slots in the route's definition and add the flows (or subtract the flows if headed in a backward direction along an edge). Go to step 1.

Further explanation of the algorithm and examples of its use are given in Harrell's paper [6]. The source code of its implementation is in the technical report [9]. If flow rates are added to the edges in Example 2 the above procedure will generate the maximum vehicular flow across the network and compute the sensitivity of the solution to changes in flow rates at critical nodes.

Off-road vehicle flow capacities may be estimated from vehicular movement formations and speeds in the terrain corridor that each edge corresponds with. Suppose that the terrain will support a certain number of units as shown in Figure 11 and the movement speeds are as in Figure 8. Figure 12 shows the number of vehicles per square kilometer that correspond to a particular movement formation. Multiplying [1/(time it takes a group of vehicles to traverse the edge)]* number of vehicles in the group determines the flow rate associated with an edge. Then Figure 13 shows a maximum flow solution and Figure 14 shows the changes in the solution caused by changing the speeds on three edges.

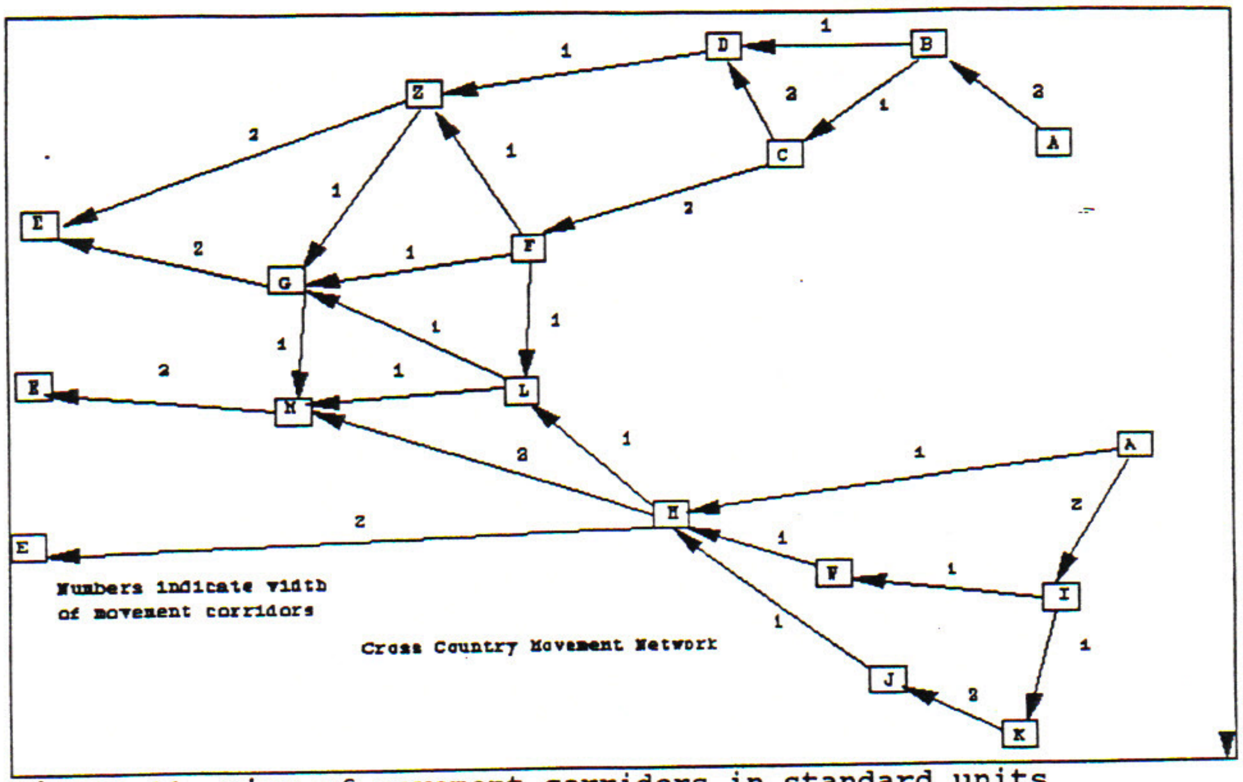


Figure 11. size of movement corridors in standard units

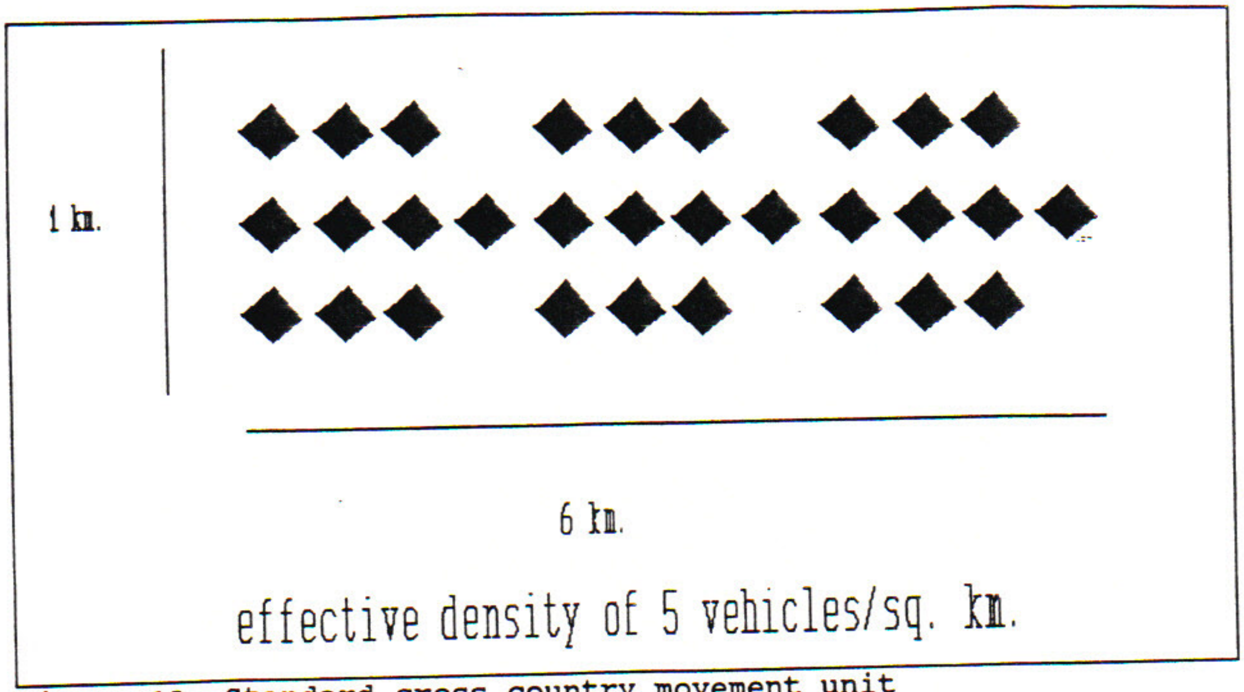


Figure 12. Standard cross country movement unit

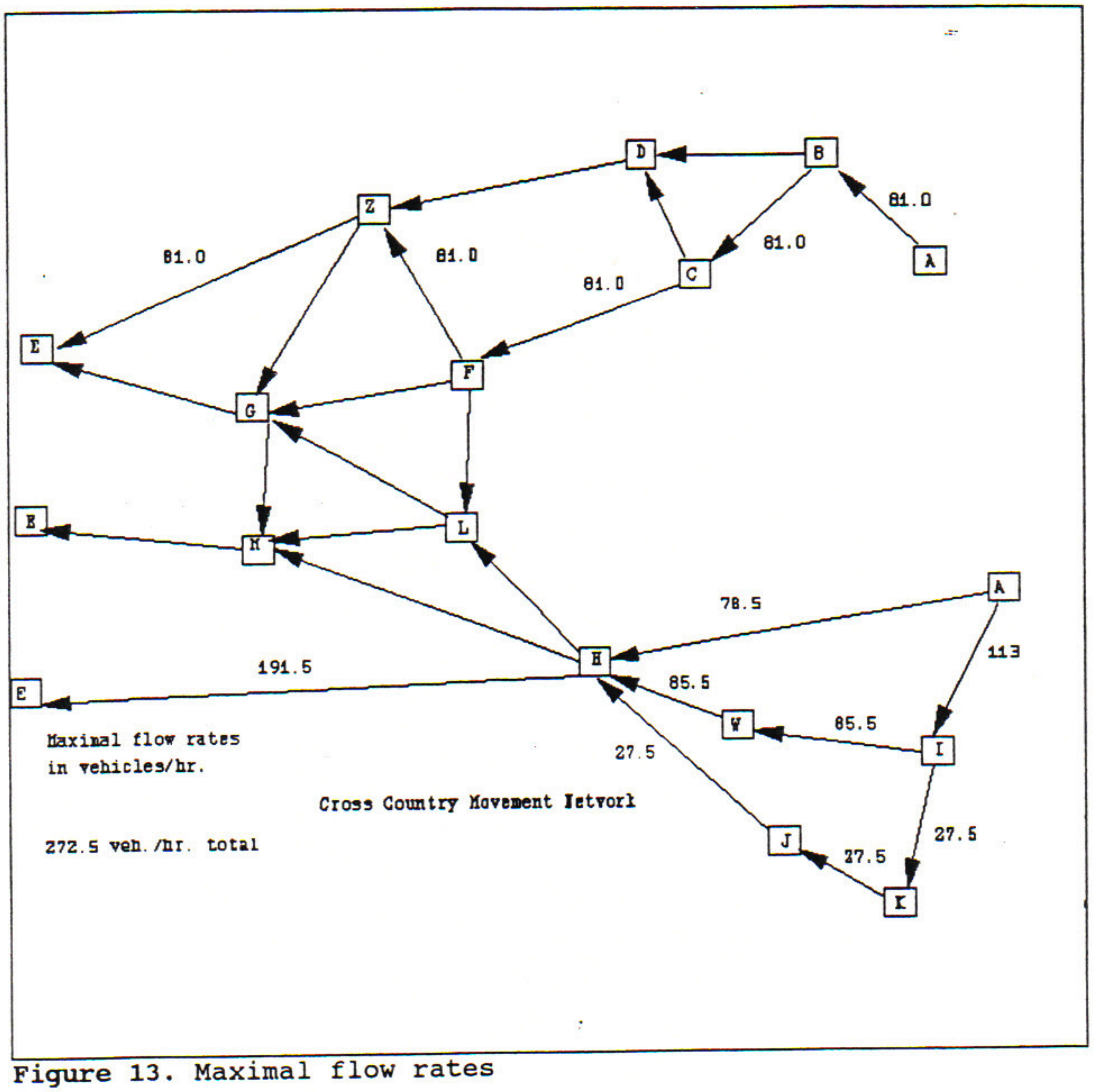


Figure 13. Maximal flow rates

The flow paths and their maximum capacities that are associated with the solutions of Figures 13 and 14 are :

Maximal flows without obstacles

```
MAXIMAL FLOWS

AIWHE flow veh/hr =   85.5
ABCFZE flow veh/hr =   81.0
AHE flow veh/hr =   78.5
AIKJHE flow veh/hr =   27.5
 total flow 272.5 veh./hr.
```

If:

a. Four anti-tank ditches with corresponding parallel tank bumps,

b. One standard(conventional) rectangular minefield
Maximal Flows with obstacles,

c. Four scatterable minefields,

are emplaced the maximal flows are reduced.

Maximal flows with obstacles

```
MAXIMAL FLOWS

ABCFZE flow veh/hr =   81.0
AHE flow veh/hr =   78.5
AIWHE flow veh/hr =   15.5
AIKJHE flow veh/hr =    6.5
 total flow 181.5 veh./hr
```

Theorems 1 through 4 solve the problem of determining how many steps it takes this algorithm to generate all the paths that create a max-min cut. As outlined in Example 3, after each search step the new edges must be placed in a sorted priority queue of current partial search paths. The maximum number of insertions required after each search step is emax + maxe. Each insertion requires the checking of the lengths of at most:

$$1 + emax*(n - n1 - ncrit) + ((emax + maxe)*maxe - 1)*n1crit + (((emax + maxe)*maxe)^{q} - 1)*(ncrit - n1crit)$$

partial paths {according to Theorem 2} against the lengths of the new paths created by adding an edge onto the active search path. Let e* = maxe + emax, and assume e* <= some constant C1,then the above expression is less than or equal to:

$$1 + ncrit*\ C1^{2q} + n*2C1^{2}$$

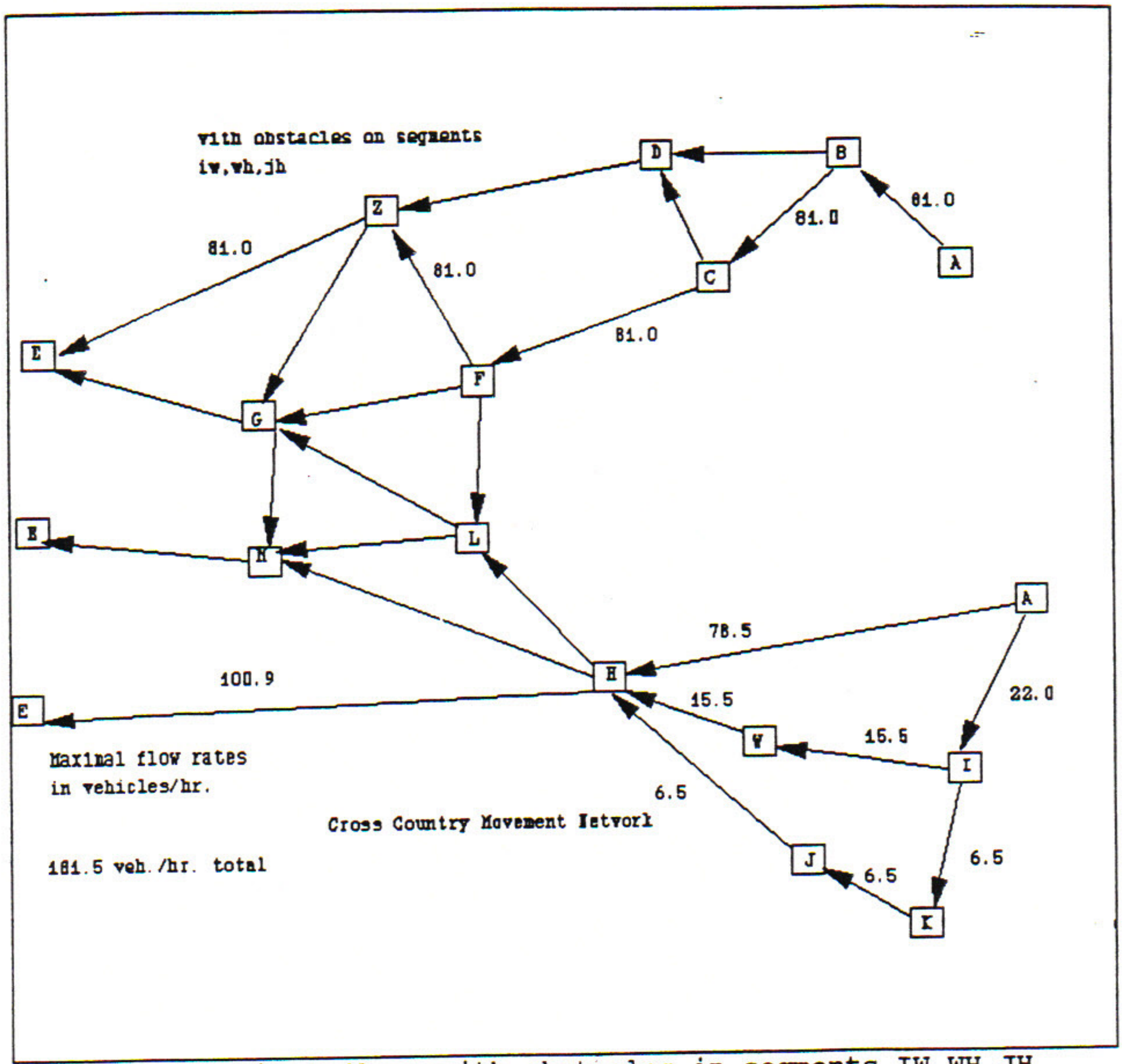


Figure 14. Maximal flows with obstacles in segments IW,WH,JH

The total number of steps is then bounded by:

$$\{1 + (emax + maxe)*(n - ncrit) + (emax + maxe)^{(q+1)}*(ncrit - n1crit)\} * [emax + maxe]^{2} * \{1 + ncrit*C1^{2q} + n*2C1^{2}\}.$$

This is less than or equal to:

$$\{C1 + n*C1^{2} + ncrit*C1^{(q+2)}\}\{1 + ncrit*C1^{2q} + n*2C1^{2}\}.$$

And this expression is of the order:

$$\{n^{2}\}*2*C1^{4} + ncrit * C1^{(3q+2)}\}$$

Theorem 5 If in designing the networks which represent the movement possibilities, we limit ourselves to the case:

$$(emax + maxe)^{4} <= n \quad \text{and} \quad (emax + maxe)^{(3q+2)} <= n^{2}, \quad q<=2$$

then the number of steps needed to solve the maximal flow problem is of the order of n^{3}, the number of vertices cubed.

Also, if we are not interested in generating the saturating flow paths in priority order of increasing flow, then the partial paths do not need to be sorted after each search step. By examining the above expressions we see that this reduces significantly the time the algorithm takes to compute a maximal flow.

4. Methodology for Solving the Minimal Cost Network Flow Problem

As a further benefit of the above approach, the procedures developed can be used to solve the minimal cost network flow problem. The minimal cost network flow problem is a generalization of the transportation network problem in operations research. In its formulation each edge is assumed to have a cost as well as a flow capacity associated. In this paper in order to determine the cost associated with a flow the following procedure is used: 1) The flow rate on each flow path solution through the network is multiplied by its time of traversal and the result summed over all paths in order to obtain a total cost associated with a given maximum flow solution. This is the measure of effectiveness which determines the optimality of the solution. The total cost of the flow can then be divided by the total maximum flow to obtain an average cost per vehicle to travel through the network. The min-cost flow problem is then the problem of determining from all the possible flows which realize a given network maximal flow value,those which do it with minimal cost. This measure of effectiveness is important in wargaming because it represents the amount of target exposure which is required for an offensive force to reach its objectives. As in the shortest path algorithm presented earlier, the algorithm that will be used to compute total cost of the flow expands the paths at each stage in the direction of shortest time. The resulting expanded pathlist will be sorted on time of the routes (in the maximal flow algorithm the maximal flow possibilities were sorted on). Only those directions in which the maximum flow algorithm says there is either a forward edge with positive unused flow capacity or a backward edge with positive existing flow should be chosen. As in the maximum flow algorithm, when the goal node is reached we proceed back to the source and subtract the maximum flow that least cost incremental flow route can handle(in the maximal flow algorithm, the route is not necesarily the least cost incrementing flow).

The following theorem from Ford and Fulkerson [5], which is also noted in Deo [3], insures that this process will generate the optimum solution.

Theorem 6 Let f be the minimal cost flow pattern of value w from start to finish. The flow pattern f' obtained by adding delta <= 0 to the flow in the forward edges of a minimal cost unsaturated path, and subtracting delta from the flow in the backward edges of the path is a minimal cost flow of value w + delta for the original network.

The same source code predicates can be used to implement this algorithm:

Step 1: Search for the best(minimal cost) route from the starting to the ending node. Only search in directions in which there is either a forward edge with positive unused flow capacity, or a backward edge with positive existing flow. If no such route exists, then terminate the algorithm.

Step 2: Subtract the value of that flow from the capacity slots in the route's definition and add the flows (or subtract the flows if going in a backward direction) along the edges. Go to Step 1.

If we again consider the network in Example 2, it is now possible to solve the problem of determining which of the several maximum flow solutions costs less in the above sense.
Minimal cost flows for the same network and same vehicle/weather conditions are shown below in Figure 15. The maximum throughput for the minimal cost flows is the same as that which results from the maximal flow algorithm. The paths followed to acheive this throughput is, however, different in the minimal cost flows from those which result from running the maximal flow algorithm. This is to be expected since in the minimal cost case the algorithm chooses the direction of shortest time to expand the search path. In the maximal flow case, the algorithm chooses the direction of

maximum flow to expand the search paths. In the list of the minimum cost flow paths, we have included an average cost for each flow. This is defined to be the average of the sum of the amount of each flow path (vehicles) times the cost of it (minutes). Note that this average does not change much with and without the presence of obstacles. This is because what the obstacles affect (being employed over only a part of the network) is primarily the maximum throughput, and not the time through the network.

```
MIN_COST FLOWS
         Minimum cost flows without obstacles


AHE      flow veh/hr =    78.5 cost time(min)=    48.7
AIWHE    flow veh/hr =    85.5 cost time(min)=    56.8
AIKJHE   flow veh/hr =    27.5 cost time(min)=    66.2
ABDZE    flow veh/hr =    66.5 cost time(min)=    67.2
ABCFGE   flow veh/hr =    14.5 cost time(min)=    67.2

total flow = 272.5 veh\hr. total cost = 78.5*48.7 + 85.5*56.8 +
27.5*66.2 + 66.5*67.2 +14.5*67.2.  average cost = 58.5 minutes per
vehicle
```

Explanation of How this Algorithm Can Be Used To Solve the Transportation Network Problem

By the transportation network problem the following is meant: Consider n points located on a map as origins of logistical material. Each point has associated with it a supply of a[i] units of the material. In addition, there are m destination points, with each destination point requiring b[i] unit of the material. Associated with each link in a network between the sources and the destinations there is a unit cost of transportation and a flow capacity. The problem is to determine the shipping pattern from origins to destinations that minimizes the total cost under the constraints imposed by the flow capacities on each link. By defining n paths each with a flow capacity equal to a[i] from a notional starting point, and m paths each with a flow capacity equal to b[i] from the destinations to a notional ending points, this problem can be considered as a special case of the minimal cost network flow problem discussed in the previous section. The algorithm given to solve that problem will in the process of computing the maximal flow in the network just defined, produce the minimal shipping cost solution which satisfies most of the total requirements at the destinations. With simple modifications to the starting requirements for the search routines the algorithm

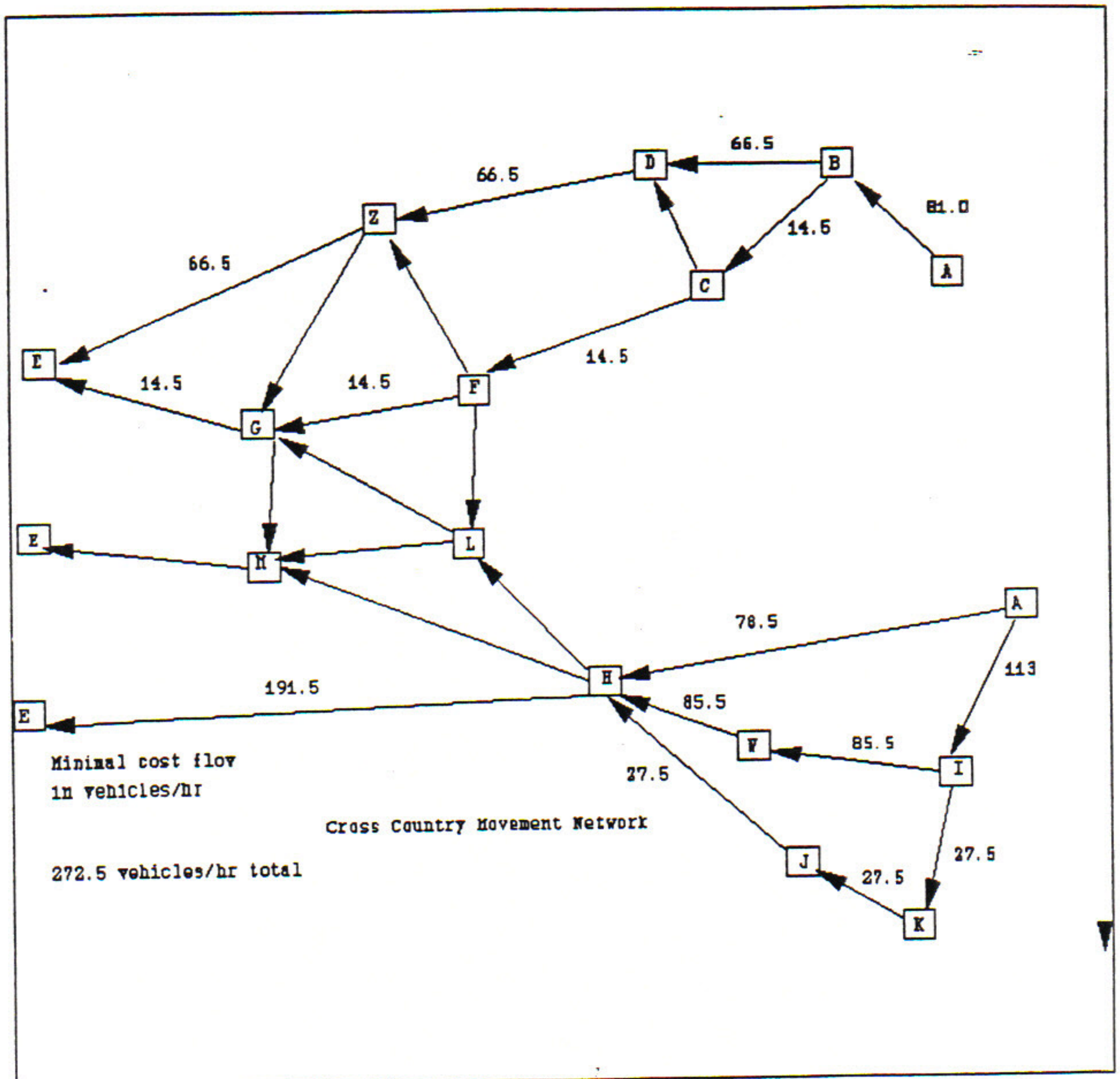


Figure 15. Minimal cost flows

[7] Harrell, A. W., "The Concept of Individual Vehicular and Unit Mobility and its Effect on Wargaming", Proceedings 27th Army Operations Research Symposium,Vol. II,pp. 3-993 to 3-1006,1988.

[8] Harrell, A. W., "The Concept of Individual Vehicular and Unit Mobility and its Effect on Wargaming", Proceedings 56th Military Operations Research Symposium, 1989.

[9] Harrell, A. W., "Evaluating the Effect of Off-Road Obstacles on Unit Movement",Technical Report GL-89-4, Waterways Experiment Station,Vicksburg, Ms., 1989.

[10] Sedgewick, Robert, Algorithms, Addison-Wesley, Reading, MA. ,1983.

[11] Sterling, Leon and Shapiro, Ehud, The Art of Prolog, The MIT Press, Cambridge, MA.,1986.

[12] Tarjan, R. E., Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadephia, PA,1983.

[13] Winston, Patrick Henry, Artificial Intelligence 2nd ed., Addison Wesley, Reading, MA.,1984.

[13] ......, Lisp 2nd ed., Reading, MA.,1984.

will produce solutions which satisfy the list of destinations in any given prioritized sequence.

Conclusions

6. We have given definitions and examples of some artificial intelligence network terminology and discussed several ways in which sorted priority queue depth-first searches can be used to solve shortest path, network maximal flow, and min-cost network flow problems. We have shown that the time bounds for these algorithms depend on the number of critical path backtracking nodes in the following sense: If there are no critical backtracking nodes, and the network is directed, then there are at most (emax -1)*(n - n1) + 1 paths in the whole search space. If there are critical path backtracking nodes then the number of paths is bounded by:

$$1 + (emax -1)*(n - n1 - ncrit) + (maxe*emax -1)*n1crit +((maxe*emax)^{q} -1)*(ncrit -n1crit)$$

We obtained similiar expressions for the case in which the paths can go either forward or backward along edges in the network. We used these expressions to obtain time bounds for the total number of steps to solve the maximal flow and min-cost flow problems.

BIBLIOGRAPHY

[1] Borlund Int., Turbo Prolog,Version 2.0,User's Guide, Scotts Valley, CA,1988.

[2] Bratko, Ivan, Prolog Programming for Artificial Intelligence, Addison Wesley Publishing Co, Reading, MA,1986.

[3] Deo,Narsingh, Graph Theory with Applications to Engineering and Computer Science., Prentice Hall, Englewood Cliffs, NJ, 1974.

[4] Edmonds, J. and Karp, R. M.,"Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," Journal for the Association of Computing Machinery, Vol. 19, no. 2, pp.248-264, New York, N.Y.

[5] Ford, L. R. and Fulkerson, D. R., Flows in Networks, Princeton University Press, Princeton, N.J.,1962.

[6] Goldberg, A. E. and Tarjan, R. E., "A New Approach to the Maximum-Flow Problem", Journal for the Association of Computing Machinery, vol.35,no. 4,pp. 921-940, New York, N.Y.,1988.